Android UI Development

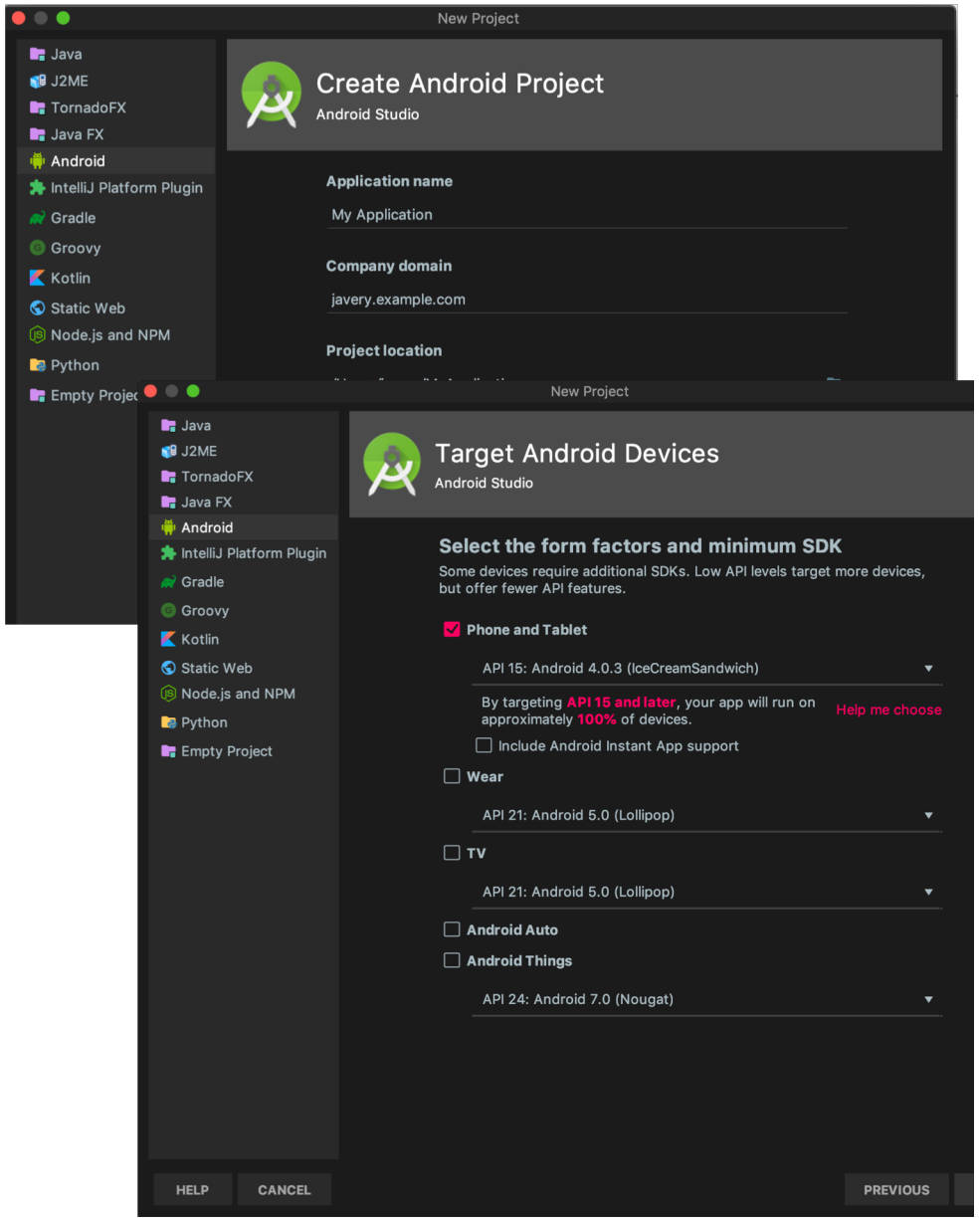
Project structure

Managing layout

Using UI widgets

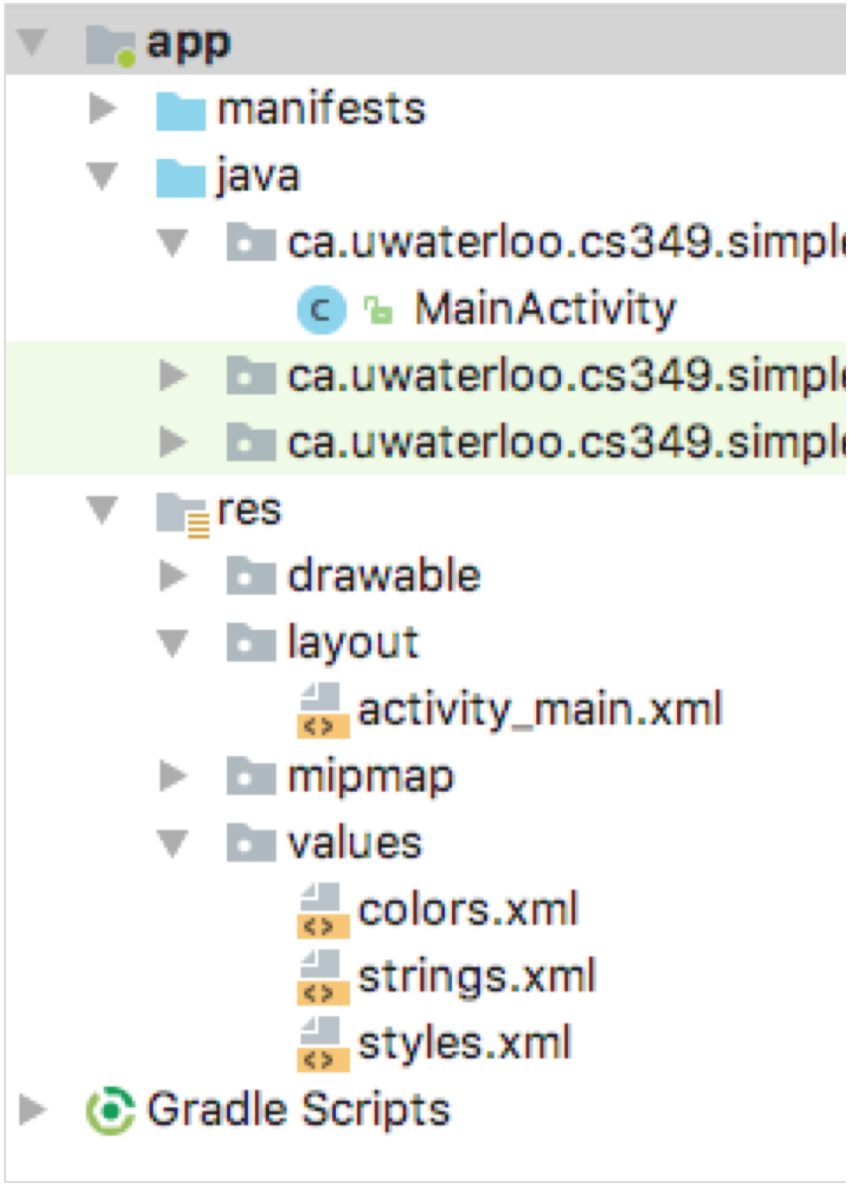
Getting Started: Project Wizard

- Company Domain
  - only important if you release your app, can just use something like: `cs349.uwaterloo.ca`
- API to target is the minimum Android version on target devices
  - Use API 15 for Phone and Tablet (we won't be doing anything restrictive)
- SDK version is the version of the dev tools, libraries etc.
  - Android Studio defaults to 28
- Activity: Whatever you start with, do NOT use fragments



Project Structure

- Manifest (app/manifests/)
  - Application setting
- Java (app/java/)
  - **(*.java) source code**
- Resources (app/res/)
  - **layout: (*.xml) UI layout and View definitions**
  - **values: (*.xml) constants** like strings, colours, …
  - also bitmaps and SVG images (mipmap*, drawable*, ….)



Manifest - Activities

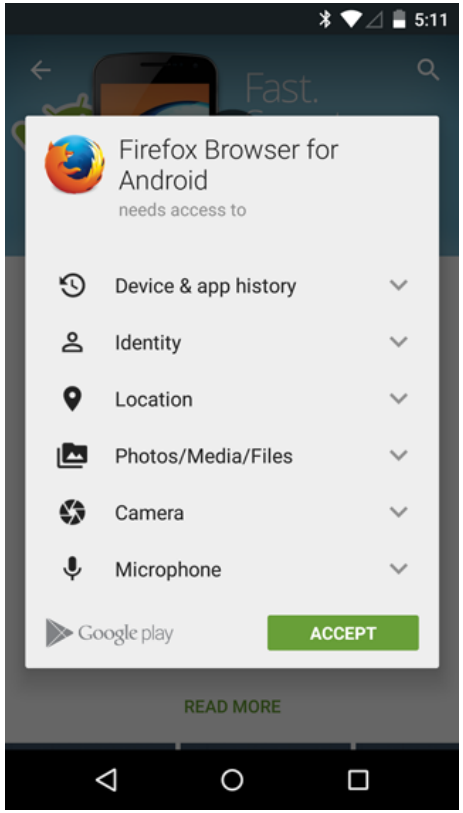
- Metadata about the app
- App components, Intent filters

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme">
    <activity android:name=".MainActivity">
        <intent-filter>
          <action android:name="android.intent.action.MAIN" />
          <category
              android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>
```

Manifest – Permissions

```
<manifest>
    <uses-permission
        android:name="android.permission.INTERNET" />
    <uses-permission
        android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission
        android:name="android.permission.SEND_SMS" />
</manifest>
```

- Android must request permission to access sensitive user data
- User is prompted once on application launch
- Do not request more than you need (please!)



App Resources

- Each type of resource in located a specific subdirectory of your project's res/ directory
- Access them using resource IDs that are generated in the project's R class

```
app/
        manifest/
        java/
        res/
                drawable/
                                graphic.png
                layout/
                                activity_main.xml
                mipmap/
                                icon.png
                values/
                                strings.xml
```

Building User Interfaces

`android.view.View`

- Base widget class (i.e. drawing and event handling)
- Subclasses:

```
android.widget.Button
android.widget.ImageView
android.widget.ProgressBar
Android.widget.TextView
...
```

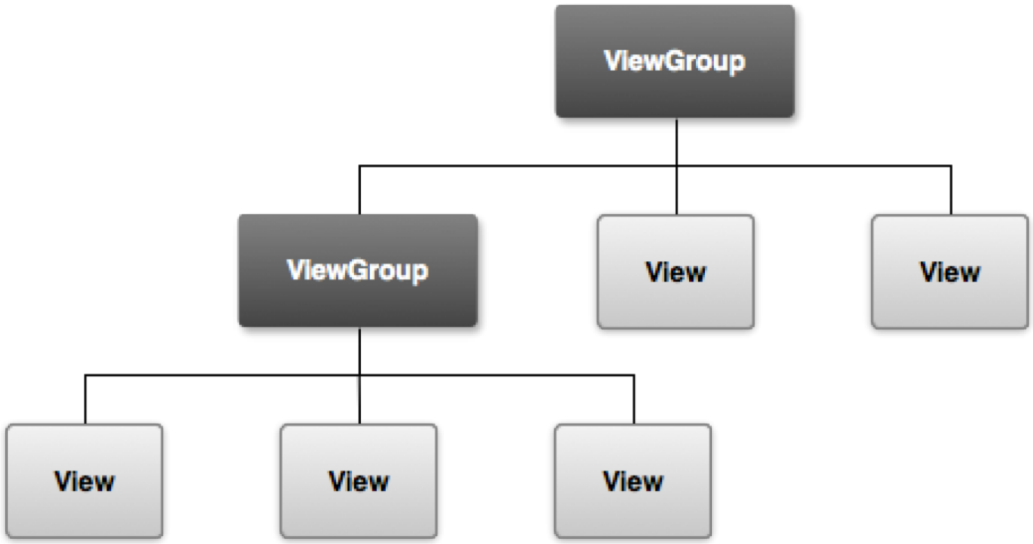
`android.view.ViewGroup`

- Abstract container class that *includes the layout*
- Subclasses:

```
LinearLayout, RelativeLayout, GridLayout, …
```

User Interface Classes

- UI is built using a hierarchy of `View` and `ViewGroup`
  - A `ViewGroup` is an invisible container that defines the layout structure for `View` and other `ViewGroup` objects
  - A `View` usually draws something the user can see and interact with



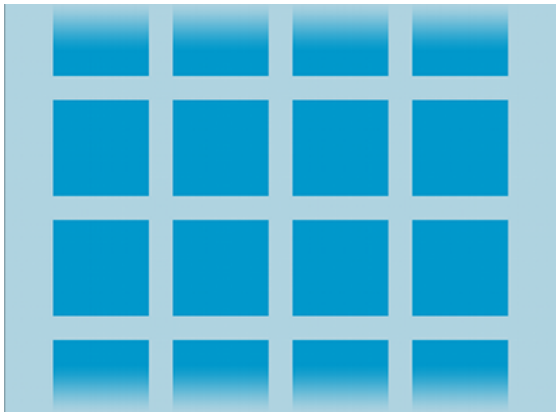
https://developer.android.com/guide/topics/ui/declaring-layout.html

Common Layouts

- Each subclass of the *ViewGroup* class provides a unique way to display the views you nest within it

Linear Layout
A layout that organizes its children into a single horizontal or vertical row

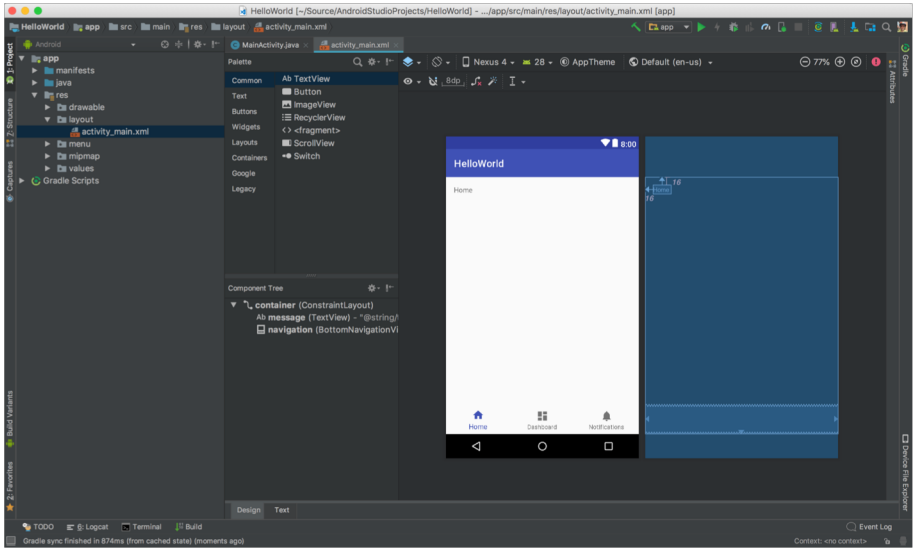
Relative Layout
Enables us to specify the location of child objects relative to each other or to the parent.



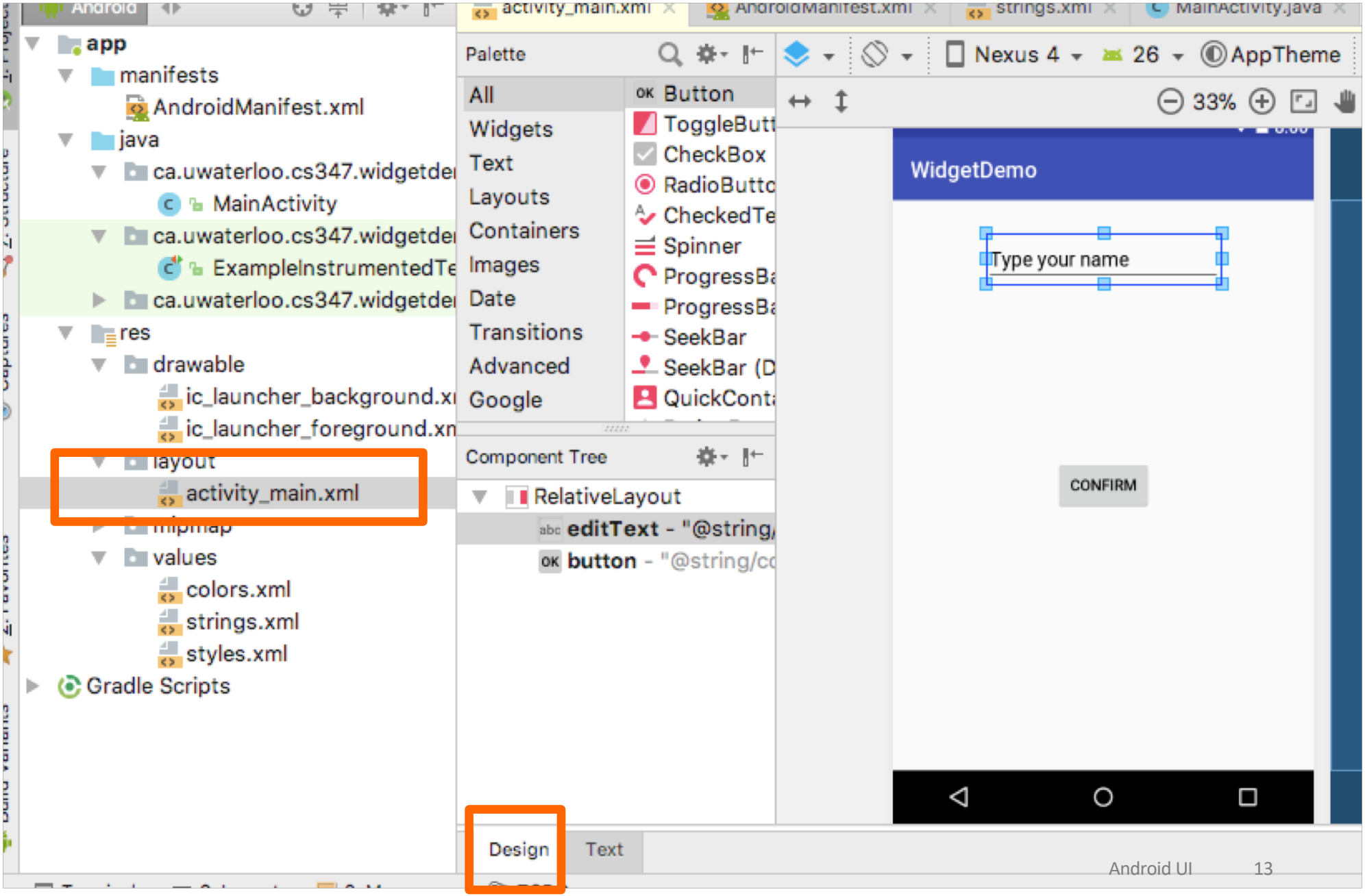
Grid View
Displays items in a two-dimensional, scrollable grid

UI Definition and Layout

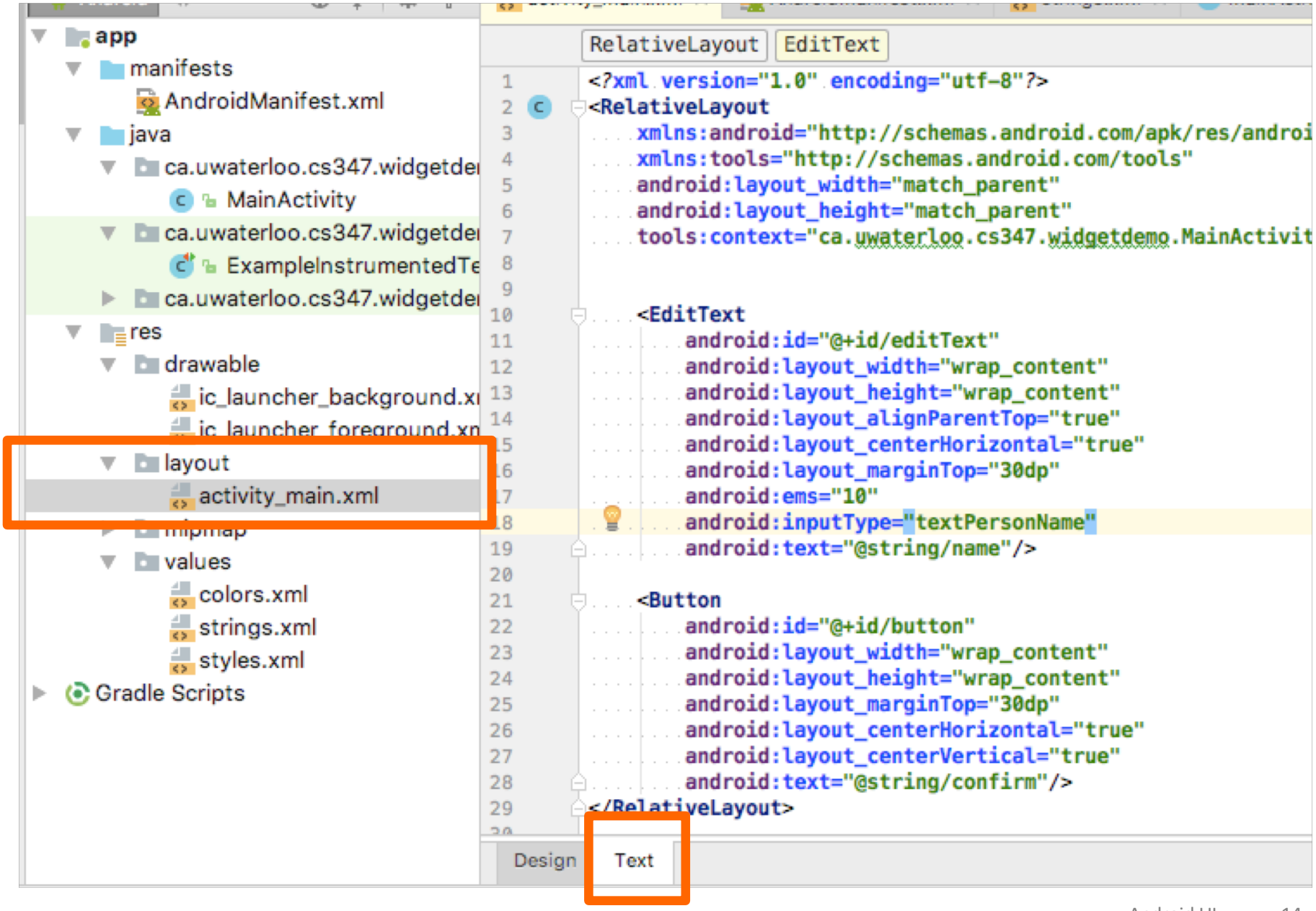
- Layout can be handled in one of two ways:
  - **Programmatic: write code.** You write code to instantiate ViewGroups, Views and bind them together (like in Java Swing).
  - **Declarative: use XML to describe your layout**. In XML describe the screen elements (view groups and views) along with properties, and then tell your application to dynamically load it.
- Using XML is the preferred way
  - Android Studio & IntelliJ both include a GUI builder to make this easier!



Layout: WYSIWYG Version

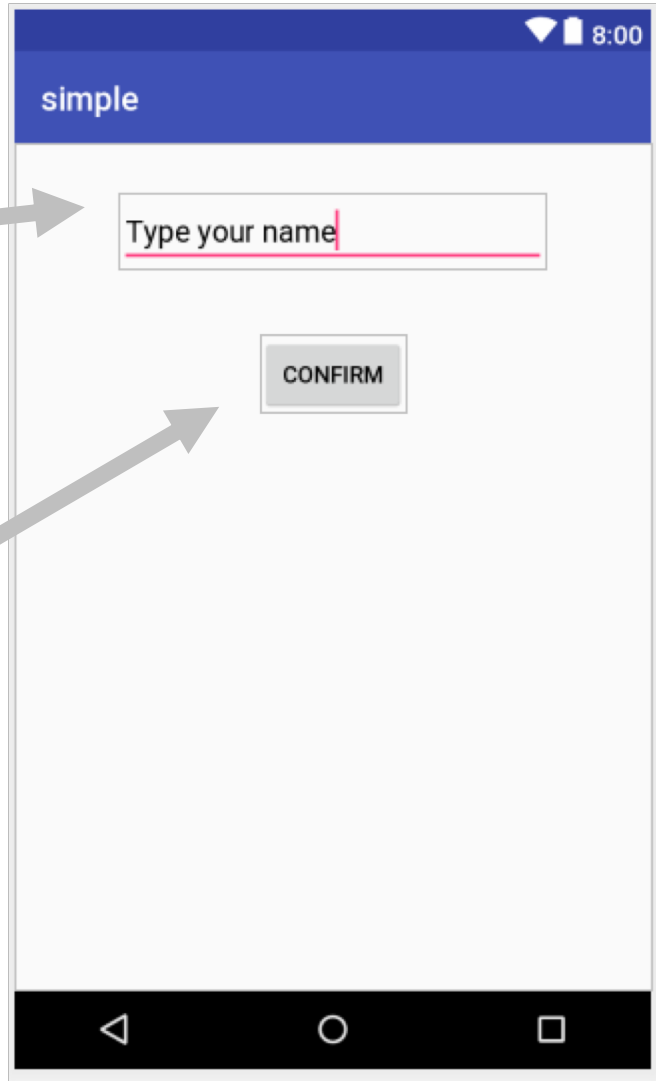


Layout: XML Version



Layout Example

```
<RelativeLayout xmlns:
    android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

     <EditText
          android:id="@+id/editTextName”
          android:layout_alignParentTop="true”
          android:layout_marginTop="30dp"
          android:ems="12"
          android:text="@string/name”
         …  />
     <Button
        android:id="@+id/btnConfirm"
        android:layout_below="@+id/editTextName"
        android:layout_marginTop="40dp"
        android:text="@string/confirm"
        …  />

</RelativeLayout>
```



Layout

- When you compile your app, each XML layout file is compiled into a View resource
- calling setContentView(), passing it the reference to your layout resource in the form of: R.layout.*layout_file_name*.
- app/java/MainActivity.java

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

Linear Layout

- **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
  - a vertical list will only have one child per row, no matter how wide is it
  - a horizontal list will only be one row high

https://developer.android.com/guide/topics/ui/layout/linear.html

Key Attributes

- **Orientation**
  - Should the layout be a column or a row? Use "horizontal" for a row, "vertical" for a column.

- **Fill model**
  - MATCH_PARENT: the view wants to be as big as its parent
  - WRAP_CONTENT: the view wants to be just large enough to fit its own internal content

- **Weight**
  - `android:layout_weight` attribute assigns an "importance" value to a view in terms of how much space it should occupy on the screen.

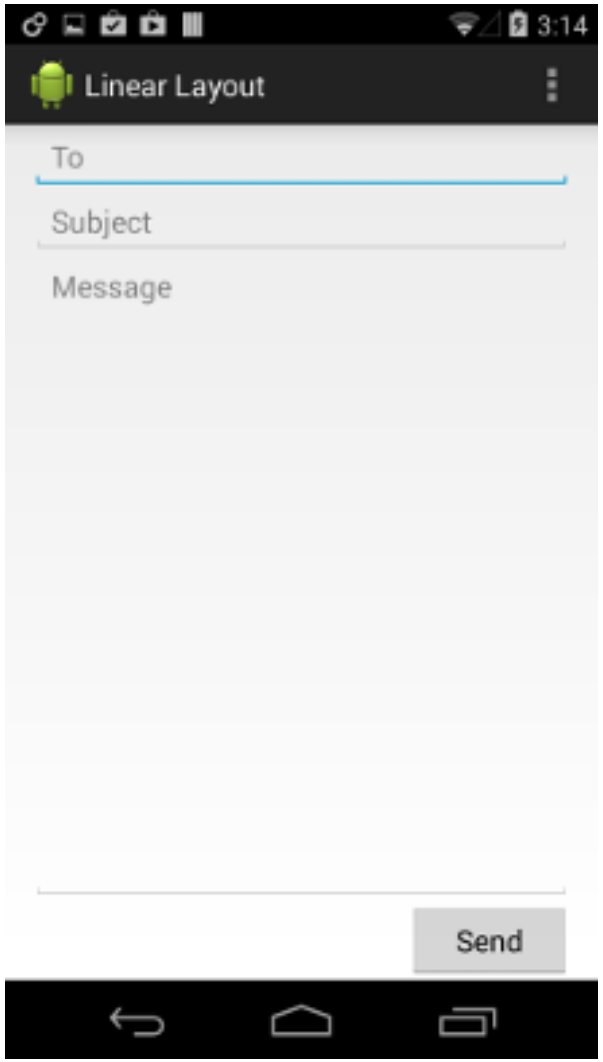
Attributes

- **Gravity**
  - Specifies how an object should position its content, on both the X and Y axes (top, bottom, center,…)

- **Padding/margin**
  - Setting padding/margin

LinearLayout

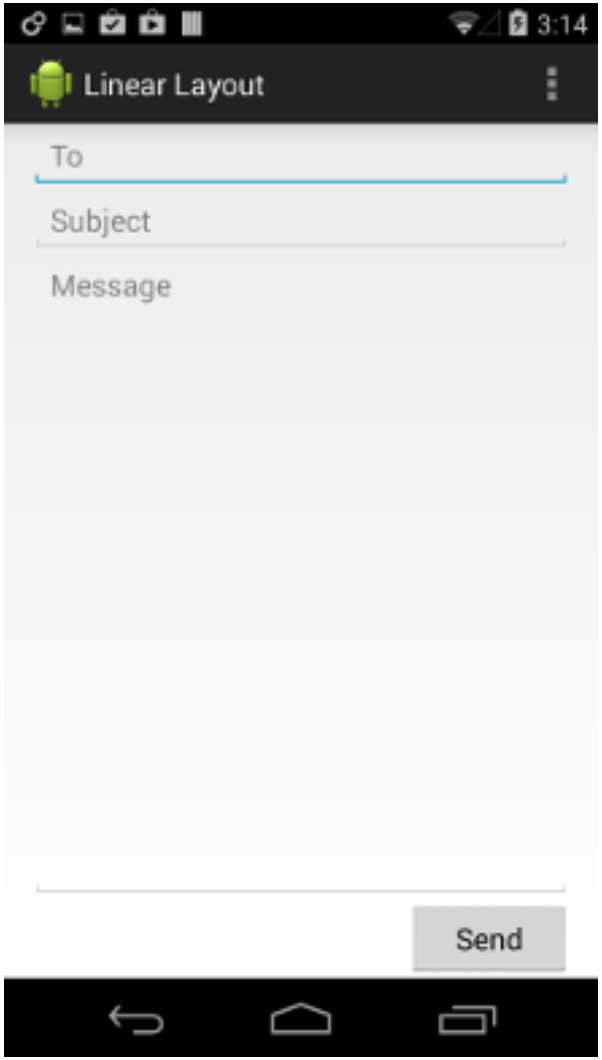
```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:orientation="vertical" >
    <EditText
        …
     />
     <EditText
        …
     />
    <Button
        …
     />
</LinearLayout>
```



LinearLayout

```
<LinearLayout …>
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/to" />
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/subject" />
    <EditText
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:gravity="top"
        android:hint="@string/message" />
    <Button
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:layout_gravity="right"
        android:text="@string/send" />
</LinearLayout>
```

Fill



Relative Layout

- `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)
  - in positions relative to the parent's area (such as aligned to the bottom, left or center).

https://developer.android.com/guide/topics/ui/layout/relative

View Positioning

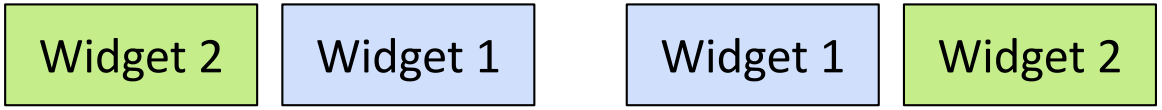
- RelativeLayout lets child views specify their position relative to the parent view or to each other (specified by ID).
- By default, all child views are drawn at the top-left of the layout
- Example of some layout properties :
  - android:layout_alignParentTop
  - android:layout_centerVertical
  - android:layout_below
  - android:layout_toRightOf
  - More: RelativeLayout.LayoutParams

https://developer.android.com/guide/topics/ui/layout/relative

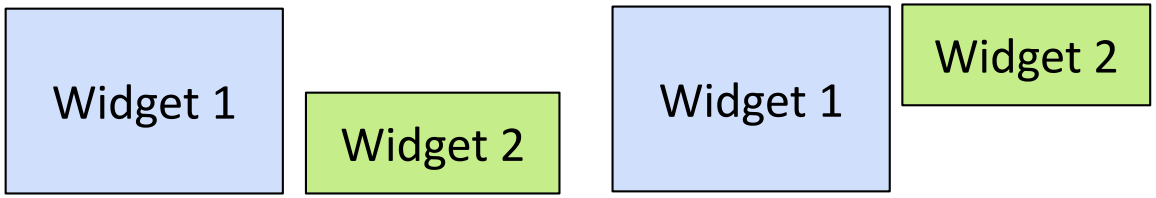
View Positioning in Relative Layout

android:layout_above
android:layout_below

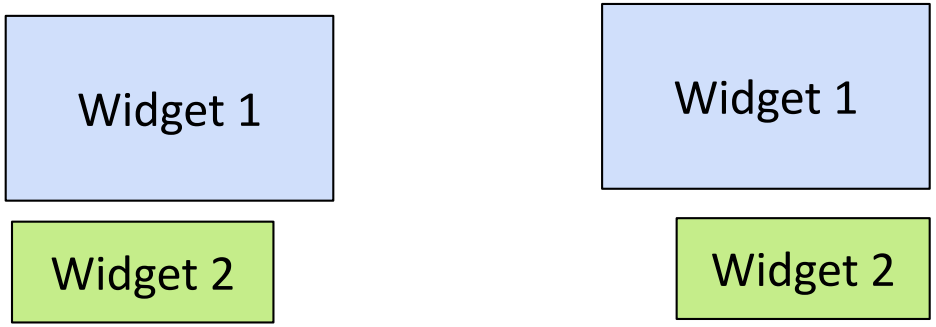
android:layout_toLeftOf
android:layout_toRightOf



android:layout_alignBottom
android:layout_alignTop



android:layout_alignLeft
android:layout_alignRight



Relative layout alignment parameters

android:layout_alignParentTop

android:layout_alignParentLeft

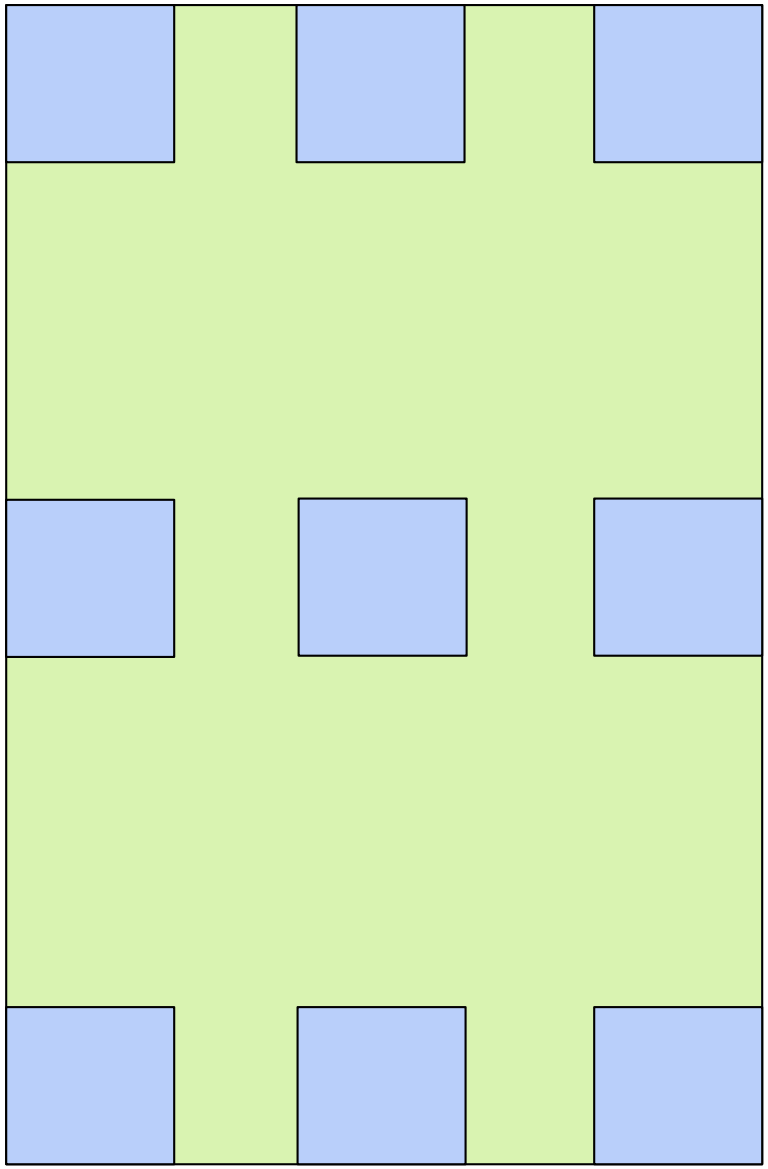
android:layout_alignParentRight

android:layout_centerInParent

android:layout_centerVertical

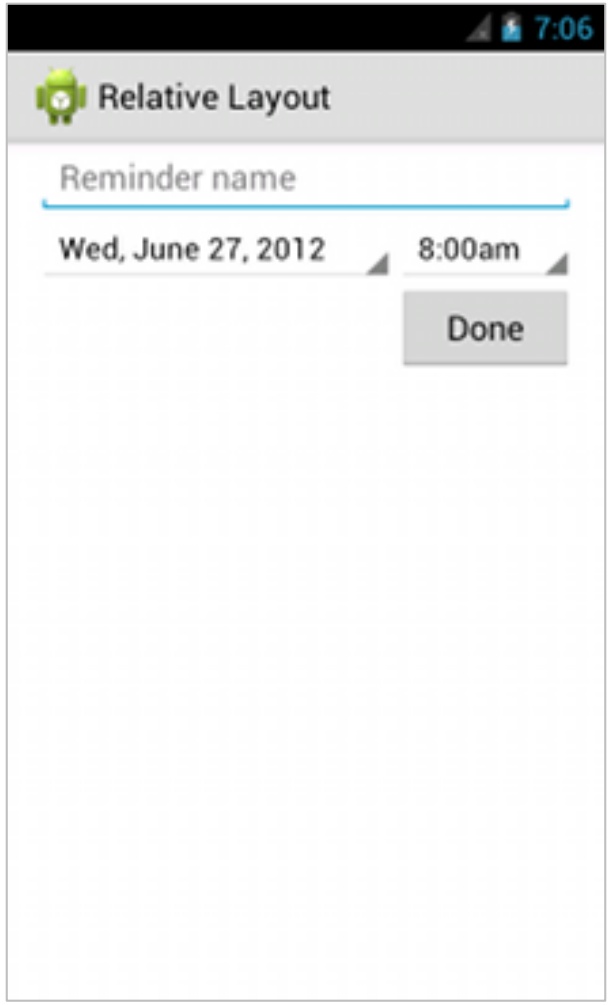
android:layout_centerHorizontal

android:layout_alignParentBottom

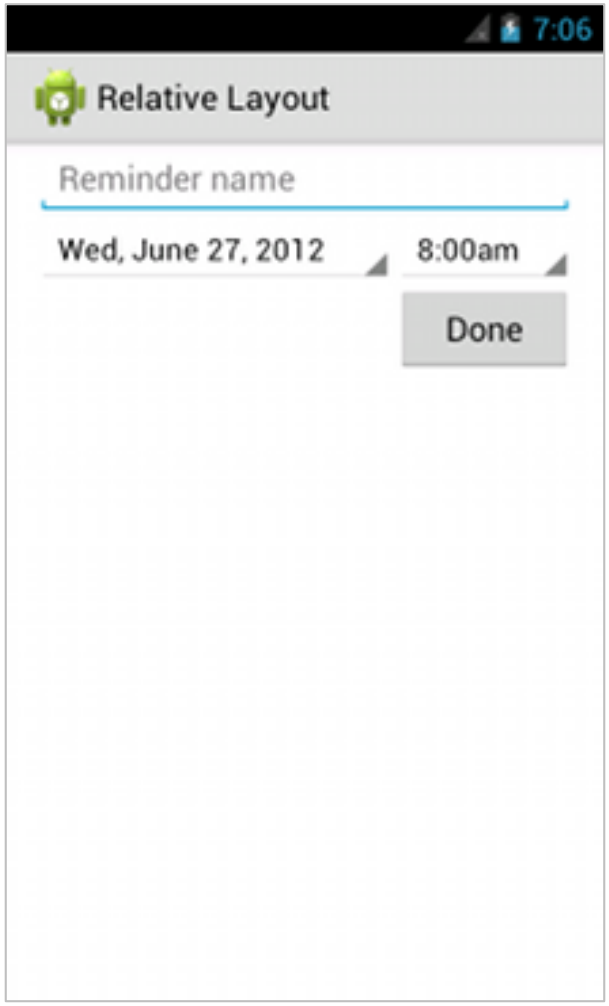


Relative Layout

```
<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
        …
    />
    <Spinner
        …
    />
   <Spinner
        …
    />
    <Button
        …
    />
</RelativeLayout>
```

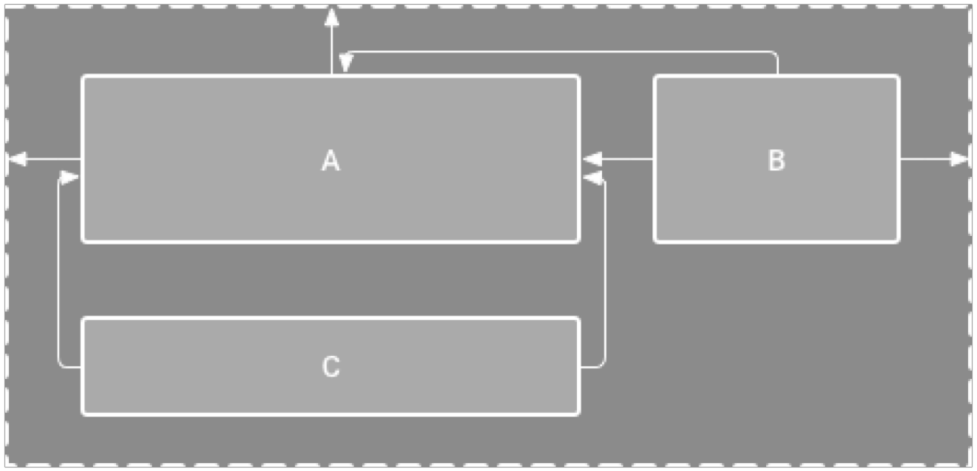


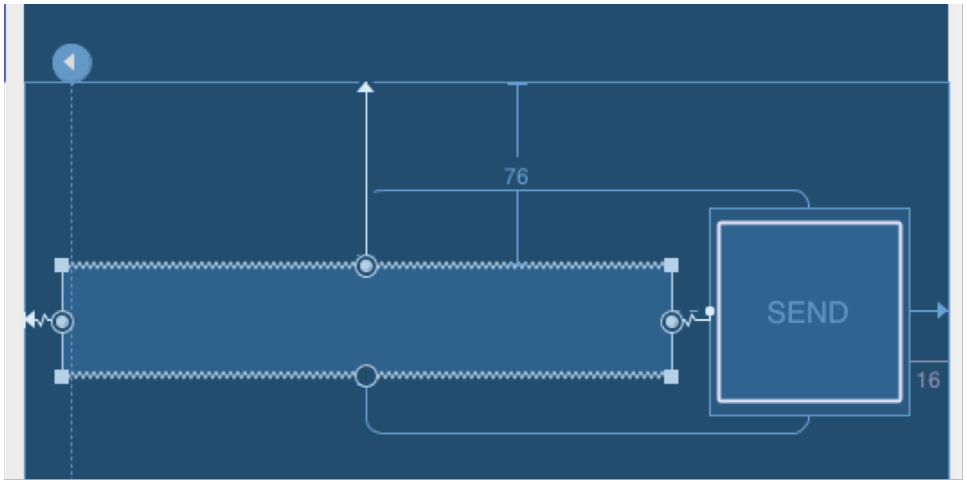
```
<RelativeLayout …
    <EditText
        android:id="@+id/name"
        android:layout_width="match_parent"
        android:layout_height="wrap_content” />
    <Spinner
        android:id="@id/times"
        android:layout_width="96dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/name"
        android:layout_alignParentRight="true" />
    <Spinner
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/name"
         android:layout_toLeftOf="@+id/times”
        android:layout_alignParentLeft="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>
```



Constraint Layout

- Similar to RelativeLayout
  - All views are laid out according to relationships with others
- Easier to use with Android Studio's Layout Editor
  - Default layout for a new layout





https://developer.android.com/training/constraint-layout/index.html

Nested Layout

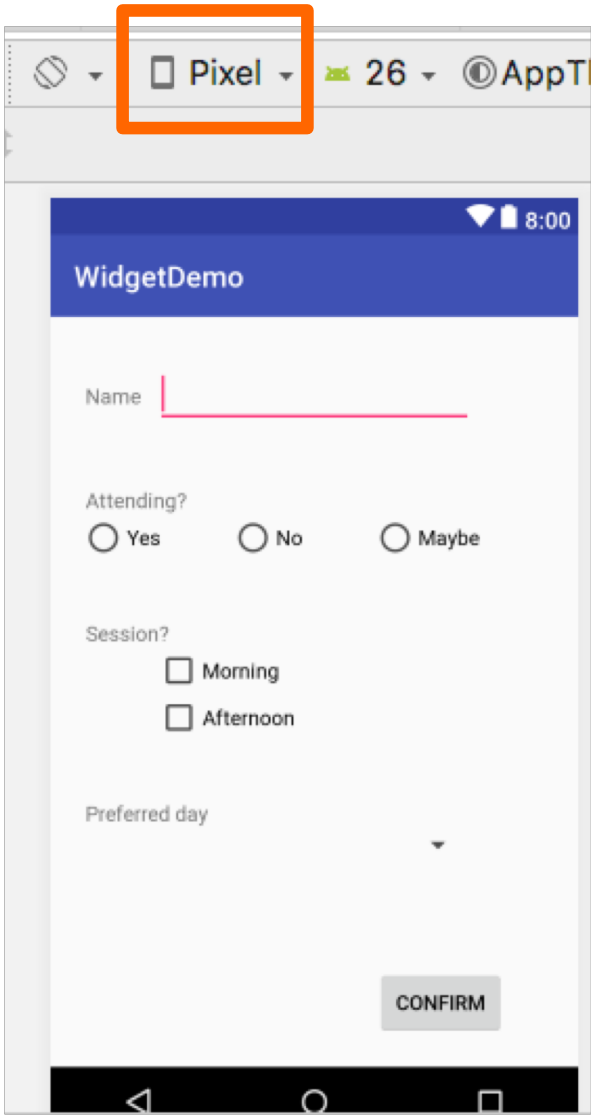
```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
       android:id="@+id/name_title"
       android:text="@string/name">
       ……
    </EditText>
       ……
    <LinearLayout
       android:layout_below="@+id/session">
       ……
        <CheckBox
            android:id="@+id/checkbox_morning"
            android:text="@string/morning"
           ……  />
        <CheckBox
           ……  />
    </LinearLayout>
</RelativeLayout>
```

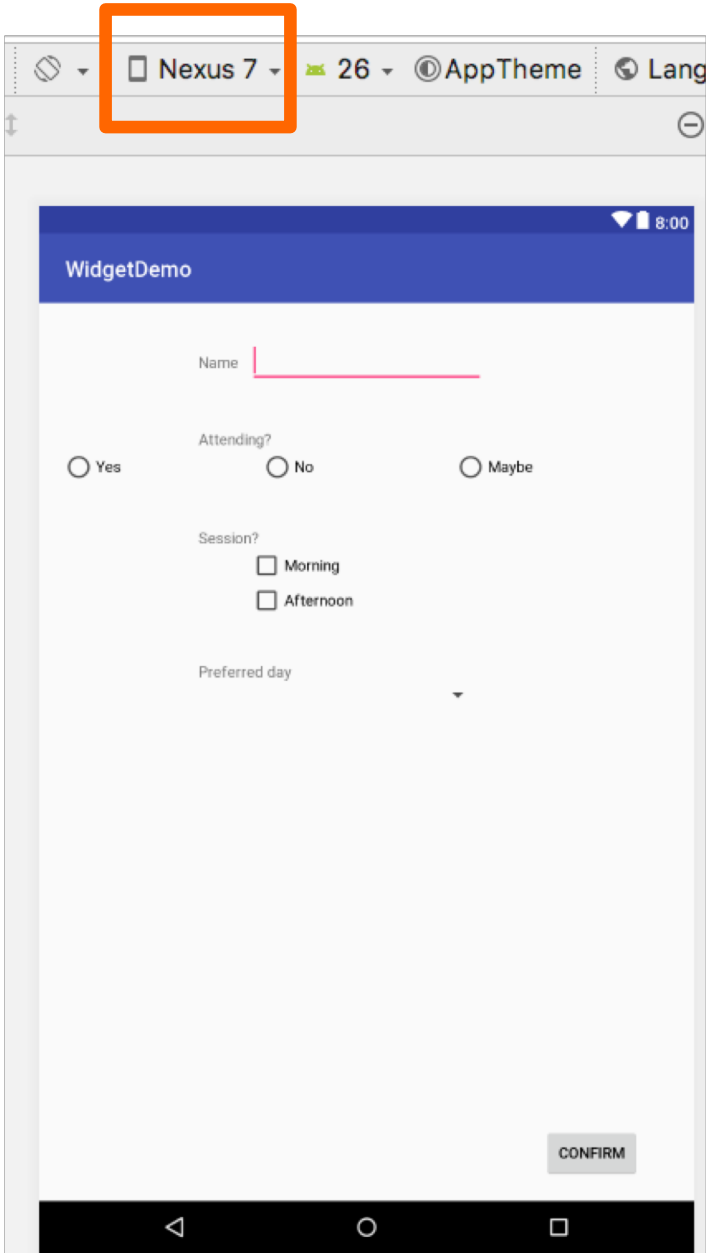
Views

Views

Layout test

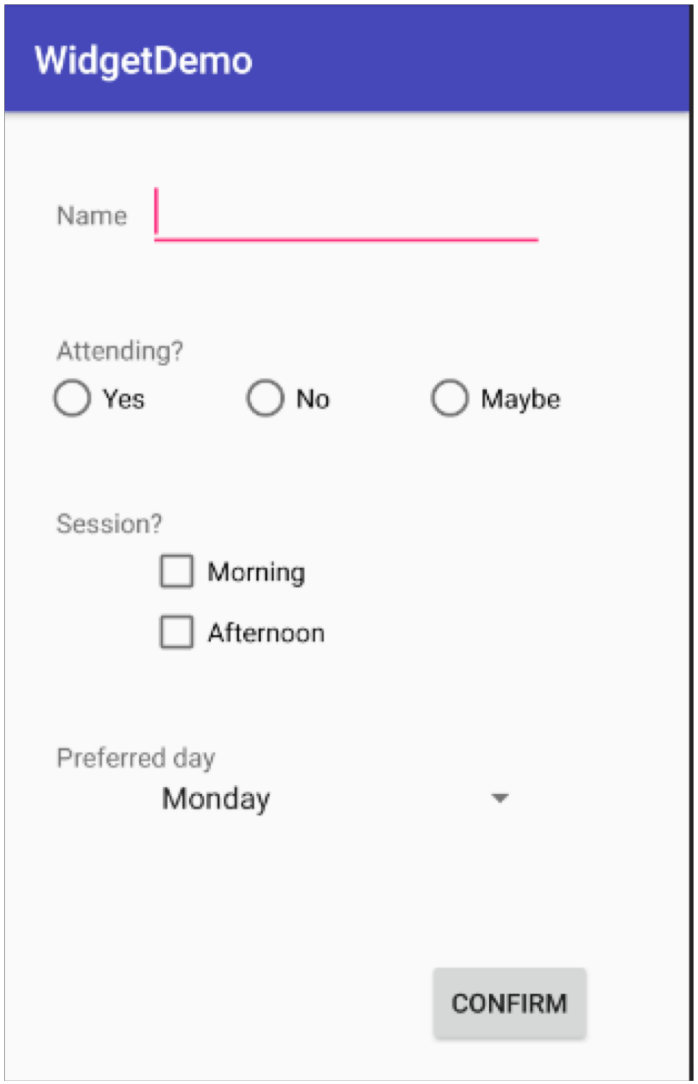
- Check the layout with multiple screen sizes





Code Demo: WidgetDemo

- Notes
  - TextView
  - EditText
  - RadioButton
  - CheckBox
  - Spinners
  - Relative Layout
  - Linear Layout
  - Nested Layout



View (Widget)

Properties:

- Background color, text, font, alignment, size, padding, margin, etc

Event Listeners and Handlers:

- respond to various events such as: click, long-click, focus change, etc.

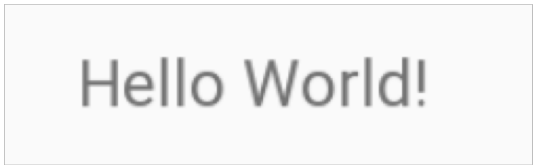
Set focus:

- Set focus on a specific view requestFocus() or use XML tag <requestFocus />

Visibility:

- You can hide or show views using setVisibility(…).

Views: TextViews




```
<TextView
    android:id="@+id/txtHello"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!" />
```

```
TextView helloTextView = findViewById(R.id.txtHello);
helloTextView.setText("CS349 W19");
```

Views: EditText


```
<EditText
    android:id="@+id/name"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content”
    android:inputType="textPersonName"
    android:text=”@string/name” >
    <requestFocus/>
 <EditText/>
```

```
EditText nameView = findViewById(R.id.name);
Text name = nameView.getText().toString();
```

Views: Buttons


```
<Button
    android:id="@+id/btnAlarm"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/alarm" />
```

https://developer.android.com/guide/topics/ui/controls/button.html

Responding to Events

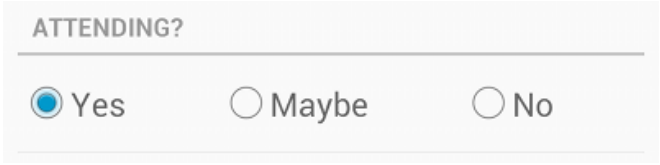
- *Option 1: Listeners*

```
Button button = (Button) findViewById(R.id.btnAlarm);
button.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        // Do something in response to button click
    }
});
```

- *Option 2: Registered in Layout file*

```
<Button
    android:id="@+id/btnAlarm"
    ……
    android:onClick="sendMessage"/>

/** Called in activity when the user touches the button */
public void sendMessage(View view) {
    // Do something in response to button click
}
```

Radio Buttons



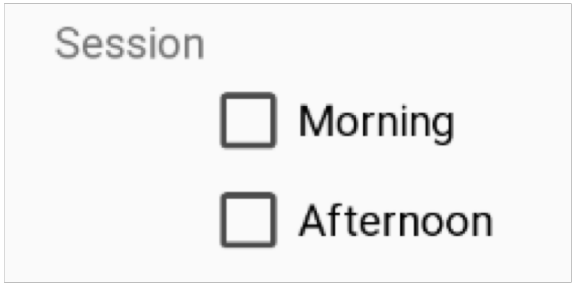

```
<RadioGroup
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal">
    <RadioButton
       android:id="@+id/radio_yes"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
        android:weight=”1"
       android:onClick="onRadioButtonClicked"
       android:text="@string/yes" />
    <RadioButton
       android:id="@+id/radio_no"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
        android:weight=”1"
       android:onClick="onRadioButtonClicked"
       android:text="@string/no" />
</RadioGroup>
```

Radio Buttons

```
public void onRadioButtonClicked(View view) {
    // Is this button checked?
    boolean checked = ((RadioButton) view).isChecked();

    // Check which radio button was clicked
    switch (view.getId()) {
        case R.id.radio_yes:
            if (checked)
                // code for yes
                break;
        case R.id.radio_maybe:
            if (checked)
                // code for may be
                break;
        case R.id.radio_no:
            if (checked)
                // code for no
                break;
    }
}
```

https://developer.android.com/guide/topics/ui/controls/radiobutton

Checkboxes




```
<CheckBox
    android:id="@+id/checkbox_morning"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="onCheckboxClicked"
    android:text="@string/morning" />

<CheckBox
    android:id="@+id/checkbox_afternoon"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="onCheckboxClicked"
    android:text="@string/afternoon" />
```

https://developer.android.com/guide/topics/ui/controls/checkbox.html

Checkboxes

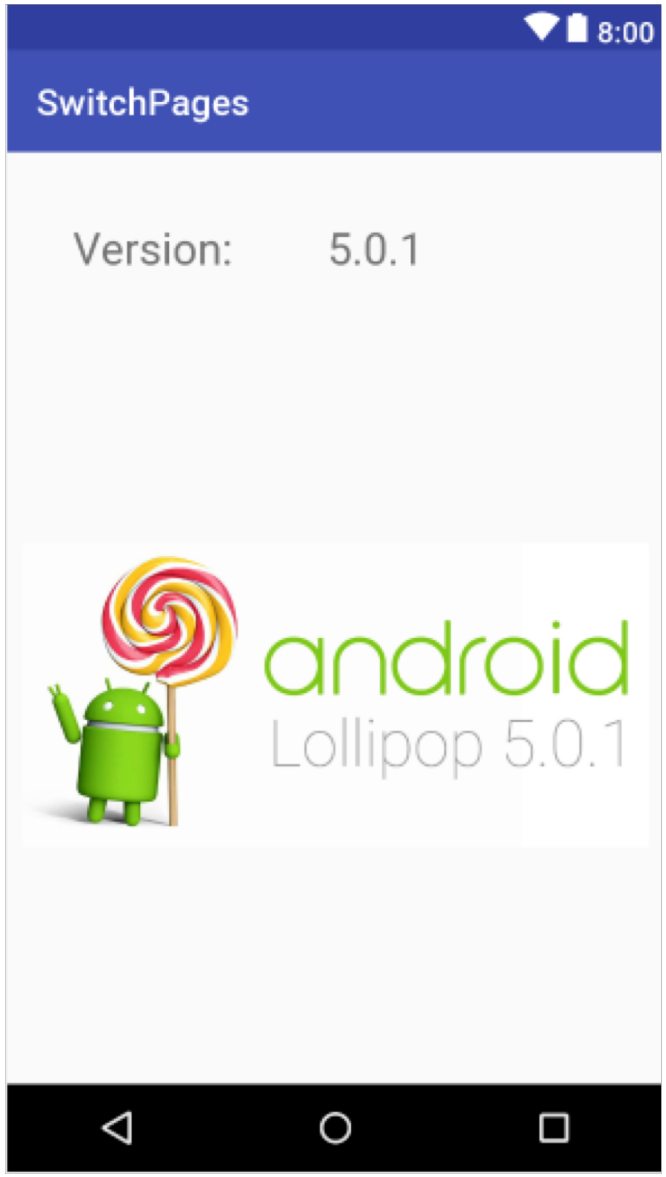
```
public void onCheckboxClicked(View view) {
    // Is the view now checked?
    boolean checked = ((CheckBox) view).isChecked();

    // Check which checkbox was clicked
    switch (view.getId()) {
        case R.id.checkbox_morning:
            if (checked)
            // Add morning session
        else
            // Remove morning session
            break;
        case R.id.checkbox_afternoon:
            if (checked)
            // Add afternoon session
        else
            // Remove afternoon session
            break;
    }
}
```

https://developer.android.com/guide/topics/ui/controls/checkbox.html

ImageView

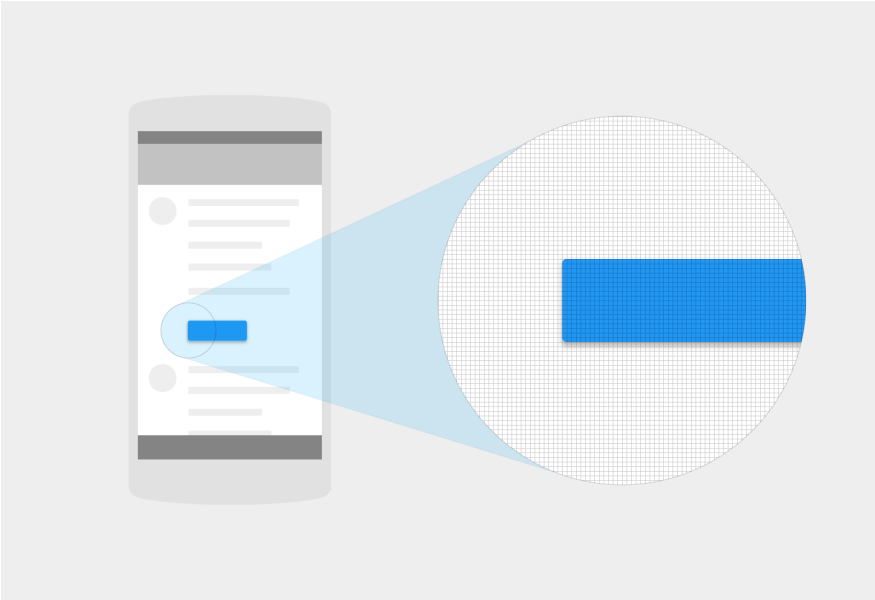
- Display images
- Save image resources to drawable folder

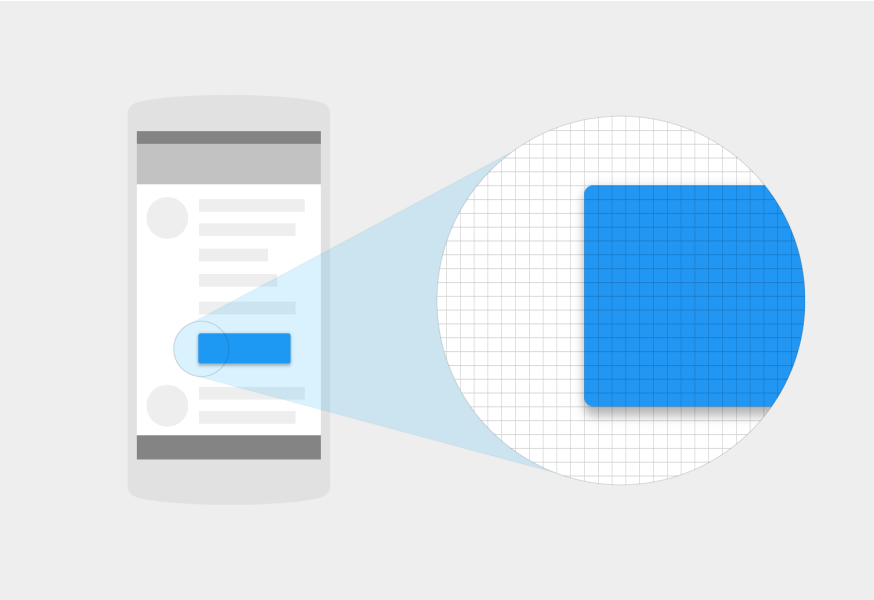
```
<ImageView
     android:id="@+id/imageView"
     android:layout_width="wrap_content"
     android:layout_height="wrap_content”
     android:src="@drawable/lollipop" />
```



Pixel density

- The number of pixels that fit into an inch is referred to as “pixel density.” ex. 160dpi = 160pixel per inch
- High-density screens have more pixels per inch than low-density ones.
- Screen density = screen width (or height) in pixels / screen width (or height) in inches





dp Layout Units

- Uses Density-independent pixels (dp), pronounced “dips”
- A dp is equal to one physical pixel on a screen with 160 dpi
- To calculate dp:

```
dp = (width in pixels * 160) / screen density
```

Screen density	Screen width in pixels	Screen width in density-independent pixels
120	180 px	240 dp
160	240 px	
240	360 px	