CSc 317
Activities and Buttons

Benjamin Dicken

Announcements

- PA 1 due Wednesday
- PA 2 due next week
- Quiz 1 grade published
- Quiz 2

ConstraintLayout vs LinearLayout

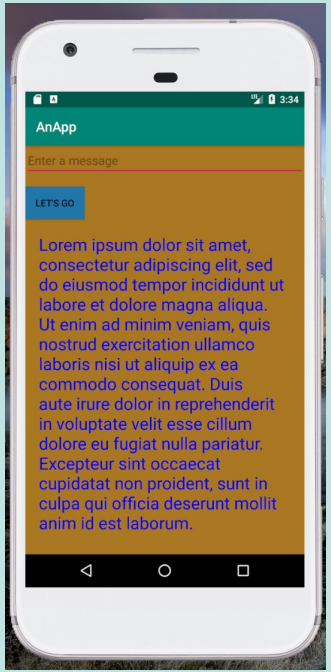
- **ConstraintLayout**: Layout out elements relative to one another
  - Notice things like:
    `app:layout_constraintEnd_toStartOf="@+id/button"`
- **LinearLayout**: Layout out sequentially, either vertically or horizontally

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

Change the Layout

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

- Use a LinearLayout
- Set to vertical layout
- Remove the **`app:layout_*`** and **`tools:layout_*`** attributes
- Ensure **`android:layout_width`** and **`android:layout_height`** are defined for the three nested elements



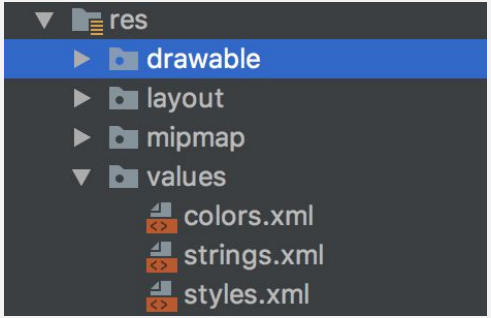
ImageView

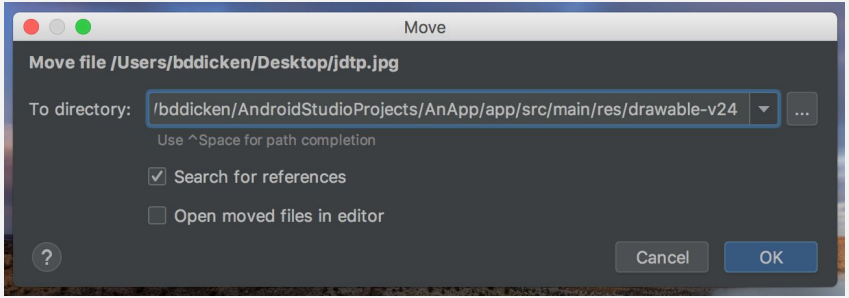
- **ImageView**: View for displaying an image
  - Specify an id
  - Specify the id of the image resource
    - **`android:src="@drawable/drawable_resource_id"`**
  - Can also specify, width, height, etc.
  - For instance:

```
<ImageView
        android:id="@+id/suns_logo"
        android:layout_width="200dp"
        android:layout_height="200dp"
        android:src="@drawable/suns_logo" />
```

Adding an image

- Click and drag an image to the **drawable** directory
- Change directory to be named **drawable**
- Click OK
- If you didn't change anything else, the ID should be the image name, not including the extension

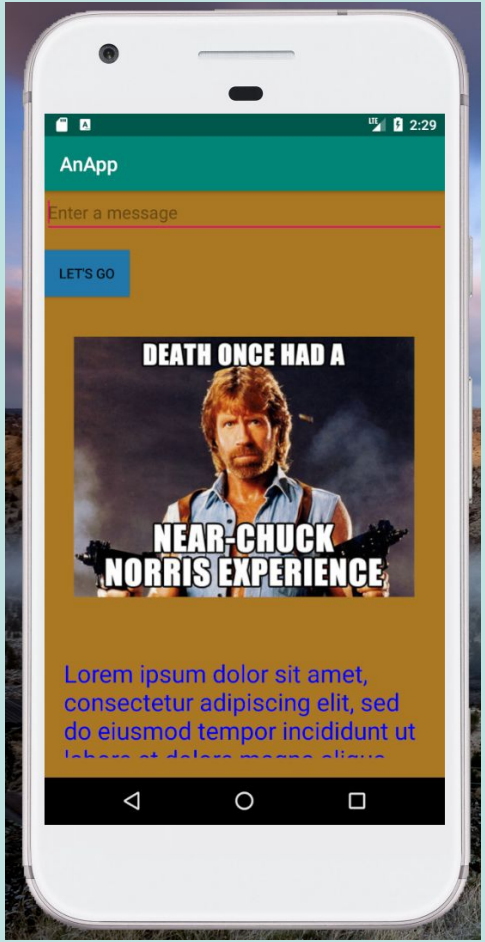




Add an image

- Download and add an image
- Use a ImageView
- Ensure **`android:layout_width`** and **`android:layout_height`** are defined

```
<ImageView
  android:id="@+id/some_id"
  android:layout_width="200dp"
  android:layout_height="200dp"
  android:src="@drawable/some_id" />
```



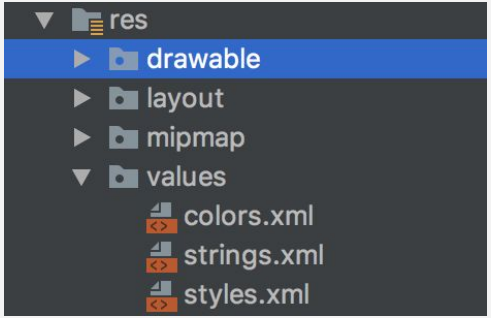
ImageView

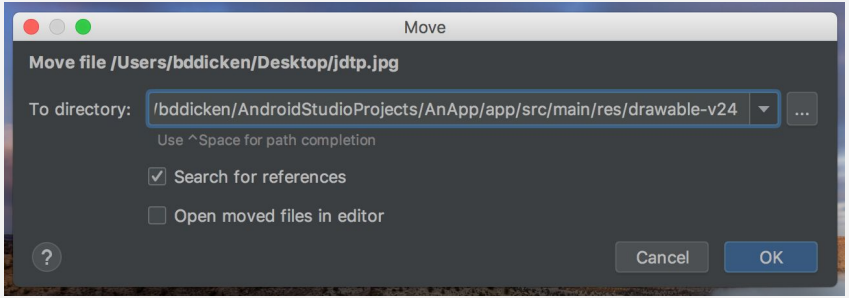
- **ImageView**: View for displaying an image
  - Specify an id
  - Specify the id of the image resource
    - **`android:src="@drawable/drawable_resource_id"`**
  - Can also specify, withd, height, etc.
  - For instance:

```
<ImageView
        android:id="@+id/suns_logo"
        android:layout_width="200dp"
        android:layout_height="200dp"
        android:src="@drawable/suns_logo" />
```

Adding an image

- Click and drag an image to the **drawable** directory
- Change directory to be named **drawable**
- Click OK
- If you didn't change anything else, the ID should be the image name, not including the extension

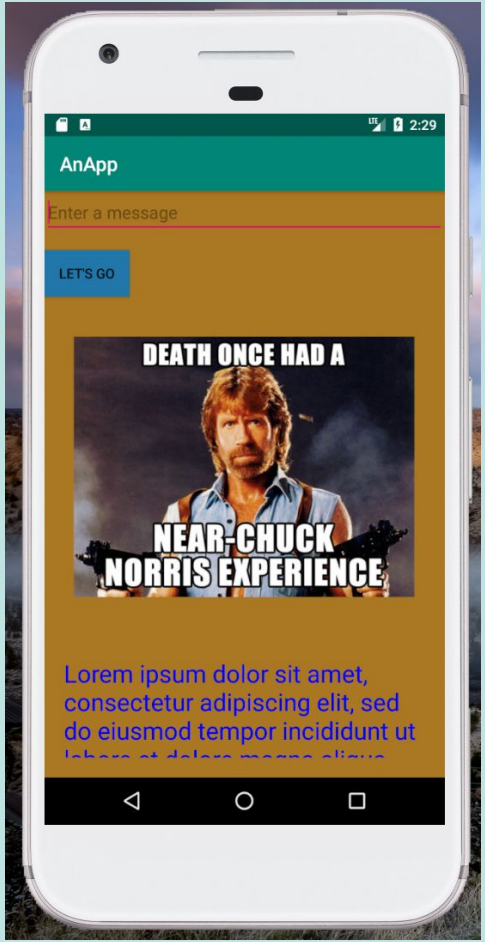




Add an image

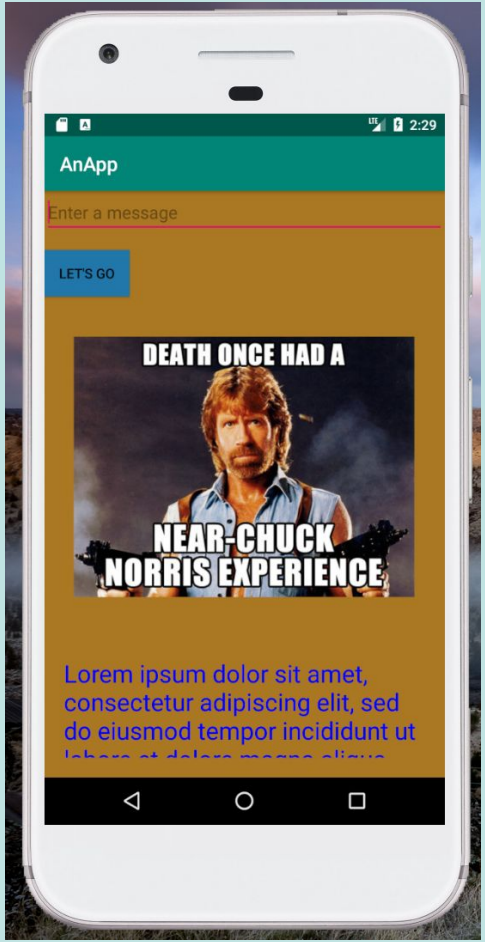
- Download and add an image
- Use a ImageView
- Ensure **`android:layout_width`** and **`android:layout_height`** are defined

```
<ImageView
  android:id="@+id/some_id"
  android:layout_width="200dp"
  android:layout_height="200dp"
  android:src="@drawable/some_id" />
```



Add an image

What about scrollability?



Scrolling

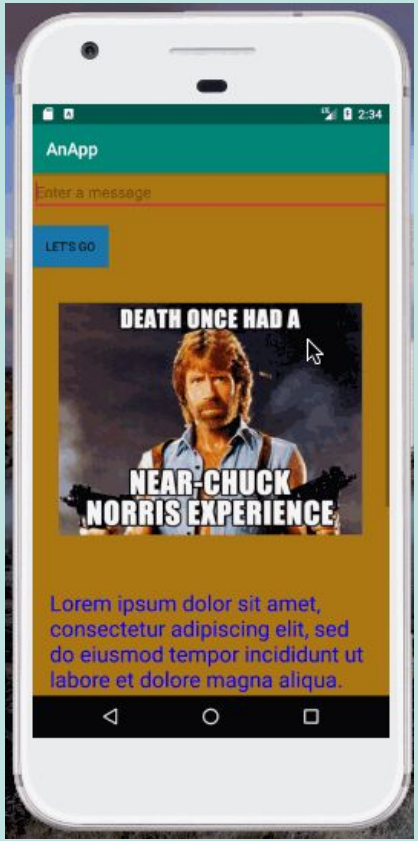
- **ScrollView**: View for displaying an image.
  - Can put a layout within a scrollview
  - Then, if necessary, can scroll through content.
  - For instance:

```
<ScrollView
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent">
```

Make it scrollable

- Use a ScrollView
- Ensure **`android:layout_width`** and **`android:layout_height`** are defined

```
<ScrollView
  xmlns:android="..."
  xmlns:app="..."
  xmlns:tools="..."
  android:layout_width="match_parent"
  android:layout_height="match_parent">
```



What about the button?

- Using the GUI, you should have gotten the button working such that when it is pressed, a new activity is started, that displays the entered text from the text box
- How did it work in the code/xml?

activity_main.xml

```
<Button
    android:id="@+id/button"
    android:onClick="sendMessage"
    android:layout_width="wrap_content"
```

MainActivity.java

```
public void sendMessage(View view) {
    Intent intent = new Intent( packageContext: this,
            DisplayMessageActivity.class);
    EditText editText = (EditText) findViewById(R.id.editText);
    String message = editText.getText().toString();
    intent.putExtra(EXTRA_MESSAGE, message);
    startActivity(intent);
}
```

DisplayMessageActivity.java

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_display_message);

    // Get the Intent that started this activity and extract the string
    Intent intent = getIntent();
    String message = intent.getStringExtra(MainActivity.EXTRA_MESSAGE);

    // Capture the layout's TextView and set the string as its text
    TextView textView = findViewById(R.id.textView);
    textView.setText(message);
}
```

Views in **Code** vs **XML**

- For the most-part, all of the view-building that you can do in XML, you can also do in-code
- Needed for PA 1
- Generally, stick to XML when you can
- If the UI needs to be dynamically changed, might want to do some in code

Views in **Code** vs **XML**

```xml
<LinearLayout
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:background="@color/sunsPurple"
  android:orientation="vertical"
  tools:context=".MainActivity">
```

```
LinearLayout linear = new LinearLayout(context);
linear.setLayoutParams(new
    LinearLayout.LayoutParams(LayoutParams.MATCH_PARENT,
                              LayoutParams.MATCH_PARENT));
linear.setOrientation(LinearLayout.VERTICAL);
linear.setBackgroundColor(Color.parseColor("#613489"));
```

Views in **Code** vs **XML**

```xml
<TextView
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_gravity="center"
  android:text="Phoenix Suns"
  android:textColor="@color/sunsOrange"
  android:textSize="50px" />
```

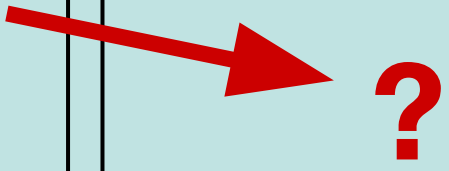
```
TextView tv = new TextView(getApplicationContext());
LinearLayout.LayoutParams tvlp = new
    LinearLayout.LayoutParams(
        LayoutParams.WRAP_CONTENT,
        LayoutParams.WRAP_CONTENT);
tv.setLayoutParams(tvlp);
tv.setGravity(Gravity.CENTER);
tv.setText("Phoenix Suns");
tv.setTextColor(Color.parseColor("#F06600"));
tv.setTextSize(50);
. . .
linear.addView(tv);
```

onCreate method outline

```
super.onCreate(savedInstanceState);
Context context =  getApplicationContext();

// Scrollview will contain a LinearLayout, which will contain content
ScrollView scroll = new ScrollView(context);
. . .

// Create the linear layout, which will hold image and text content (ImageView and TextView objects)
LinearLayout linear = new LinearLayout(context);
. . .

// Create your individual TextViews and ImageViews (about 7 total)
// make sure to add each one to the LinearLayout with linear.addView()
. . .

scroll.addView(linear);
setContentView(scroll);
```

XML to Code

```
<ImageView
  android:layout_width="200px"
  android:layout_height="200px"
  android:src="@drawable/suns_logo"
/>
```



?

?

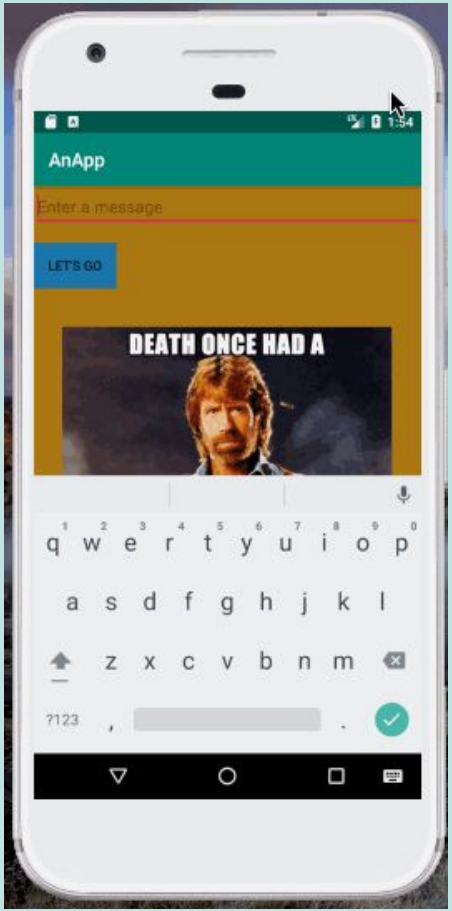
XML to Code

```xml
<ImageView
  android:layout_width="200px"
  android:layout_height="200px"
  android:src="@drawable/suns_logo"
/>
```

```java
ImageView suns_logo = new ImageView(this);
LayoutParams lp = new LayoutParams(200, 200);
suns_logo.setLayoutParams(lp);
suns_logo.setImageResource(R.drawable.suns_logo);
linear.addView(suns_logo);
```

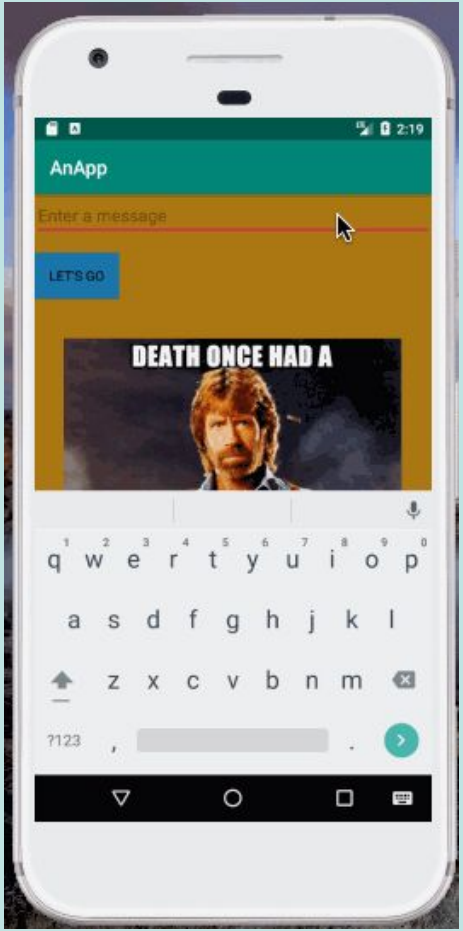
Change font size

- Change **`activity_display_message.xml`**
- Make the text display larger when the other activity is visible.



Change font size

- Change **`activity_display_message.xml`**
  - Make the text 40dp
- Also:
  - Update **`DisplayMessageActivity.java`** so that each time a value is entered, it keeps track of it, in a list format.



Four Main Building blocks of apps

- **Activities** - The entry point for interacting with the user. It represents a single screen with a user interface.
- **Services** - A general-purpose entry point for keeping an app running in the background for all kinds of reasons. For instance, a download or music.
- **Broadcast Receivers** - A component that enables the system to deliver events to the app outside of a regular user flow, allowing the app to respond to system-wide broadcast announcements.
- **Content providers** - Manages a shared set of app data.

Loosely taken from the android dev docs

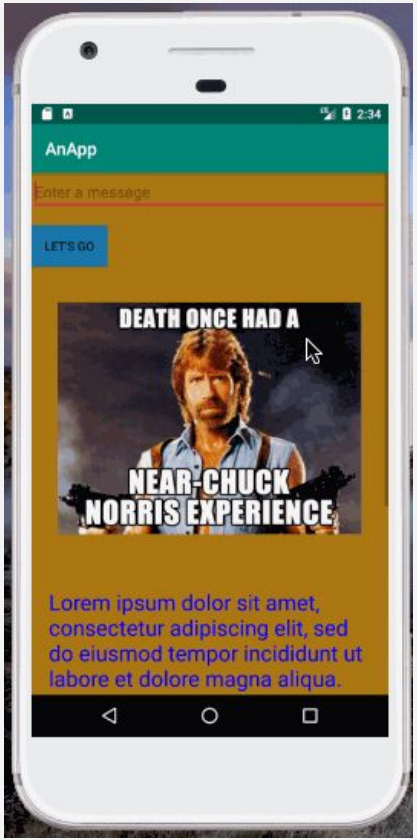
Activities

- Activities are one of the **building blocks** of android applications
- From the reading:

> *The Activity class is a crucial component of an Android app, and the way activities are launched and put together is a fundamental part of the platform's application model. Unlike programming paradigms in which apps are launched with a main() method, the Android system initiates code in an Activity instance by invoking specific callback methods that correspond to specific stages of its lifecycle.*

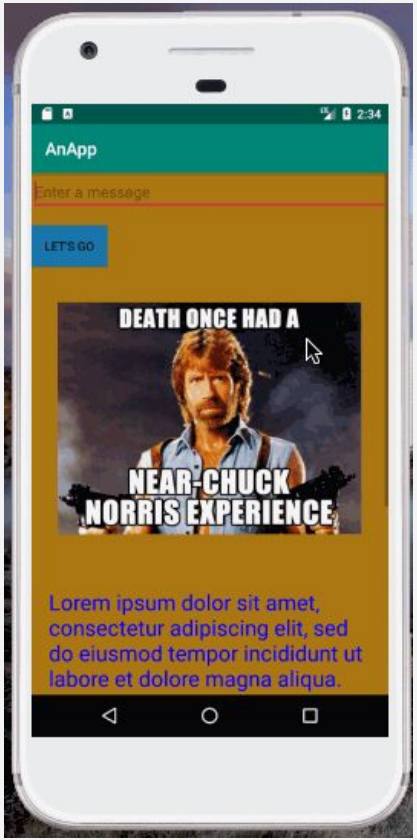
Activities

- For now, each main interface screen of your application will be built using an **Activity**
- A simple, one-page app might only need one activity
- Some apps may have many
- Each activity that has a UI should have an associated layout xml file



Activities

- How does one create an activity?
- How does one change from one active activity to another?



A new activity

- Create a new activity, and call it **MemeActivity**
  - Two new files should be created: `MemeActivity.java` and `activity_meme.xml`
- The new view should display one meme of your choice
- Do this in code/xml, not with the GUI editor