CSCI 322/522 **Assignment 2: ConstraintLayout Exercise**
50 points

This assignment is more of an exercise than an assignment but you will earn 50 points of credit for turning in a working version of what is produced following the steps in this assignment.

Instructions

- The first step is to create a new Android Studio project.
- Open Android Studio and create a **New Project**.
- In our class example so far, we have used the **Empty Views Activity** template but, for this assignment, we will learn how to create an entirely new activity and corresponding layout resource file manually, so select the **No Activity** option and then click **Next**.
- The project name should be **ConstraintLayout Exercise** and specify the package name as edu.niu.android.constraintlayoutexercise.
- Before clicking on **Finish**, change the Minimum API level setting to API 21 ("Lollipop"; Android 5.0) and change the language to Java if not already Java.

Creating a New Activity

- Once the project creation process is complete, the Android Studio main window should appear.
- The first step in the project is to create a new activity and doing so is a valuable learning exercise since there are many instances in the course of developing Android applications where new activities need to be created from the ground up without using the template choices provided at the beginning of creating a new project.
- If not already open, click on the **Project** side tab in the upper left corner to display the **Project** tool window.
- Once displayed, expand the hierarchy by clicking on the right facing arrows next to the entries in the Project window (you do not need to expand the **Gradle Scripts**).
- The objective here is to gain access to the **app > java > edu.niu.android.constraintlayoutexercise** folder in the project hierarchy.
- Once the package name is visible, right-click on it and select the **New > Activity > Empty Views Activity** menu option as illustrated in Figure 1 below (it shows **Empty Activity** in Figure 1).
- Alternatively, select the **New > Activity > Gallery...** option to browse the available templates and make a selection using the New Android Activity dialog.
- When the New Android Activity window opens, be sure the Activity Name is MainActivity, that the Generate a Layout File box is checked, that the Layout Name is activity_main, that the Layout Activity box is checked and then click Finish.

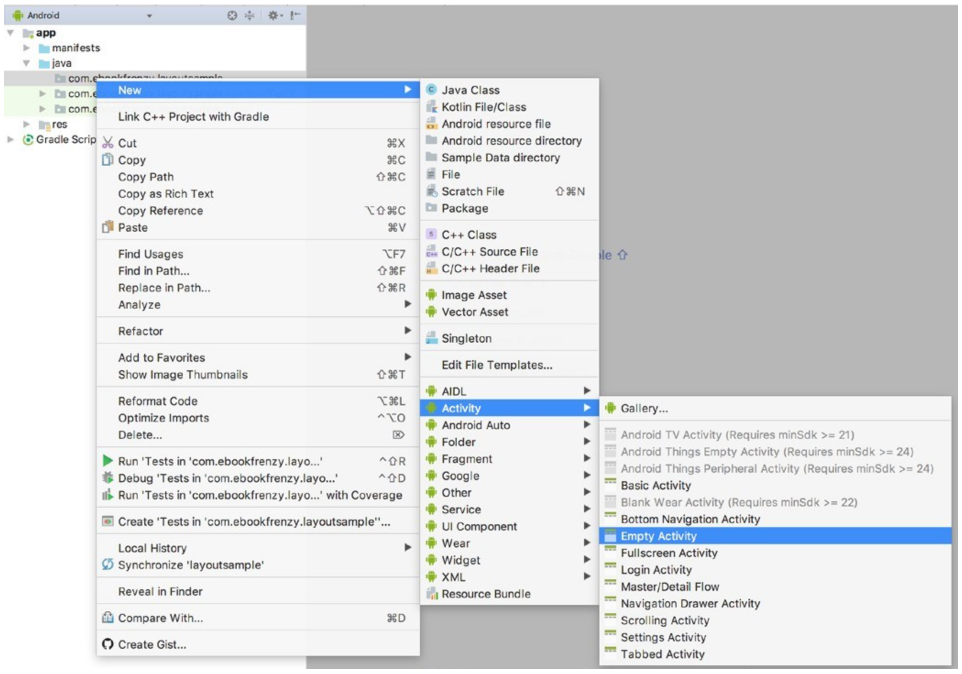


Figure 1

- As an FYI, in order for an application to be able to run on a device it ***must*** have an activity designated as the launcher activity because the operating system must know which activity to start up when the application first launches.

- At this point Android Studio should have added two files to the project.

- The Java source code file for the activity named `MainActivity` should be located in the **app** > **java** > **edu.niu.android.constraintlayoutexample** folder.

- In addition, the XML layout file for the user interface named `activity_main.xml` should have been created in the new **app** > **res** > **layout** folder created by adding the new Activity.

- Note that the **Empty Views Activity** template was chosen for this activity so the layout is contained entirely within the `activity_main.xml` file and there is no separate content layout file.

- Finally, the new activity should have been added to the `AndroidManifest.xml` file and designated as the launcher activity.

- The manifest file can be found in the project window under the **app** > **manifests** folder and should contain the following XML:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools">
  <application
    android:allowBackup="true"
    android:dataExtractionRules="@xml/data_extraction_rules"
    android:fullBackupContent="@xml/backup_rules"
    android:icon="@mipmap/ic_launcher"
```

```
      android:label="@string/app_name"
      android:roundIcon="@mipmap/ic_launcher_round"
      android:supportsRtl="true"
      android:theme="@style/Theme.ConstraintLayoutExercise"
      tools:targetApi="31">
      <activity
        android:name=".MainActivity"
        android:exported="true">
        <intent-filter>
          <action android:name="android.intent.action.MAIN" />

          <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
      </activity>
    </application>
  </manifest>
```

Preparing the Layout Editor Environment

- Locate and double-click on the `activity_main.xml` layout file located in the **app > res > layout** folder to load it into the **Layout Editor** tool.
- Since the purpose of this tutorial is to gain experience with the use of constraints, turn off the **Autoconnect** feature using the button located in the **Layout Editor** toolbar.
- Once disabled, the button will appear with a line through it as shown in Figure 2 below:

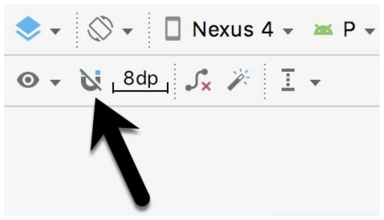


Figure 2

- If the default margin value to the right of the **Autoconnect** button is not set to 8dp, click on it and select 8dp from the resulting panel.
- The user interface design we want will also make use of an `ImageView` object to display an image.
- Before proceeding, save the image file named `galaxys6.jpg` found in the Assignment 2 folder on Blackboard to your computer.
- Within Android Studio, open the **Resource Manager** side tab near the upper left corner of Android Studio to open its tool window.
- Click, drag and drop `galaxy6s.jpg` from your computer onto the **Resource Manager** tool window in Android Studio.

- In the resulting dialog, click **Next** and then click **Import** to add the image to project and the image should now appear in the **Resource Manager** as shown in Figure 3 below:

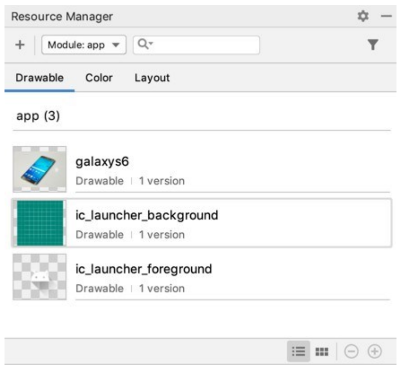


Figure 3

- Close the **Resource Manager** side tab by clicking on it again and then click on the **Project** side tab to open it again.

- The image now appears in the **res > drawable** folder of the **Project** tool window as shown in Figure 4 below:

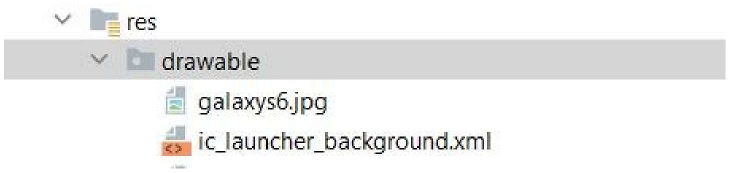


Figure 4

Adding the Widgets to the User Interface

- If not already so, open file `activity_main.xml` in **Design** mode.

- From within **Palette**, open the **Common** selection and drag an `ImageView` object into the center of the display view.

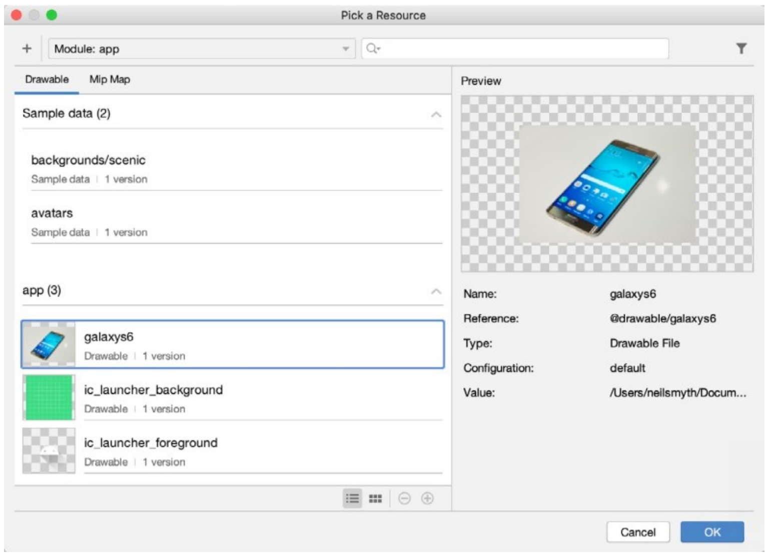
- Note that horizontal and vertical dashed lines appear indicating the center axes of the display.

- When centered, release the mouse button to drop the view into position.

- Once placed within the layout, the **Pick a Resource** dialog will open so that you can pick which resource to add to the `ImageView`.

- Click on **galaxys6** to select it as illustrated in Figure 5 below:

(continued)



Name:	galaxys6
Reference:	@drawable/galaxys6
Type:	Drawable File
Configuration:	default
Value:	/Users/neilsmyth/Docum...

Figure 5

- Click on OK to assign it to the `ImageView` object.
- If necessary, adjust the size of the `ImageView` using the resize handles and reposition it in the center of the layout.
- At this point the layout should match Figure 6 below:

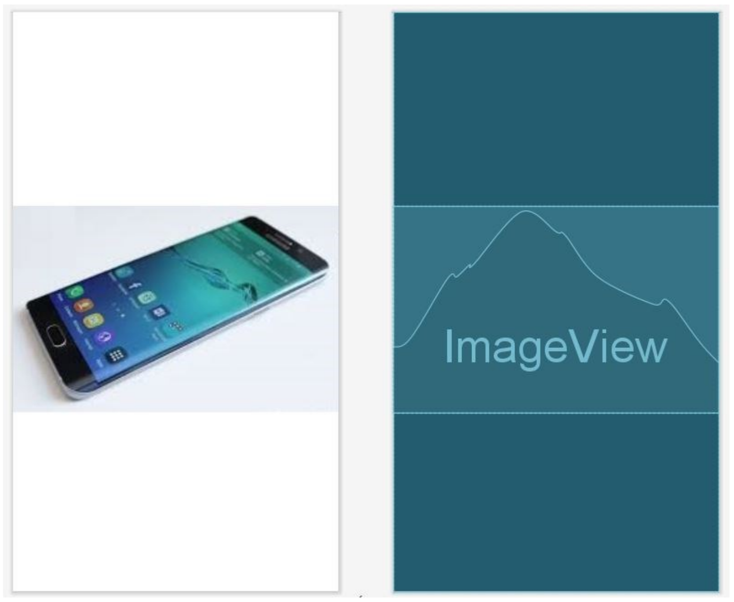


Figure 6

(continued)

- Next, click and drag a `TextView` object from the **Text** selection of the **Palette** and position it so that it appears above the `ImageView` as shown in Figure 7 below:

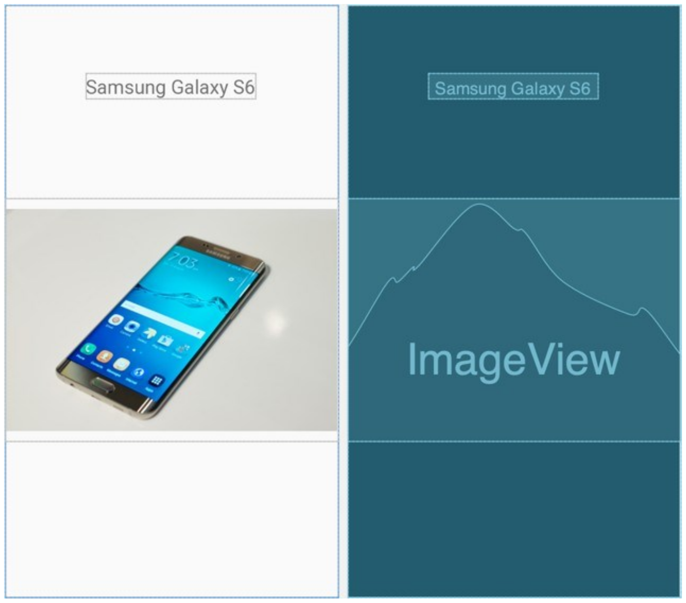


Figure 7

- Using the **Attributes** panel, expand the `textAppearance` attribute entry in the **Common Attributes** section, change the text to "Samsung Galaxy S6", the `textSize` property to 24sp and center the text.

- Next, add three `Button` widgets along the bottom of the layout and set the text attributes of these views to "Buy Now", "Pricing" and "Details". The completed layout should now match Figure 8:

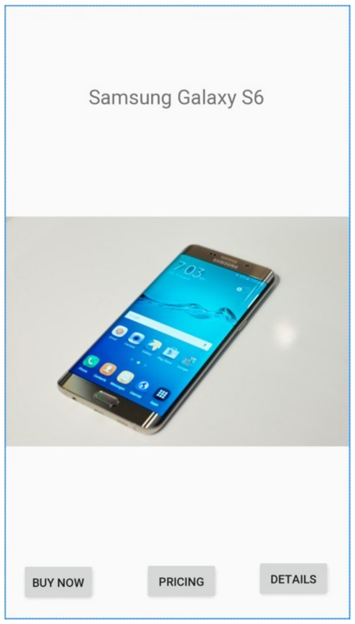


Figure 8

- At this point, the widgets are not sufficiently constrained for the layout engine to be able to position and size the widgets at runtime.

- Were the app to run now, all of the widgets would be positioned in the top left-hand corner of the display.
- With the widgets added to the layout, use the device rotation `Button` located in the **Layout Editor** toolbar (indicated by the arrow in Figure 9) to view the user interface in landscape orientation:

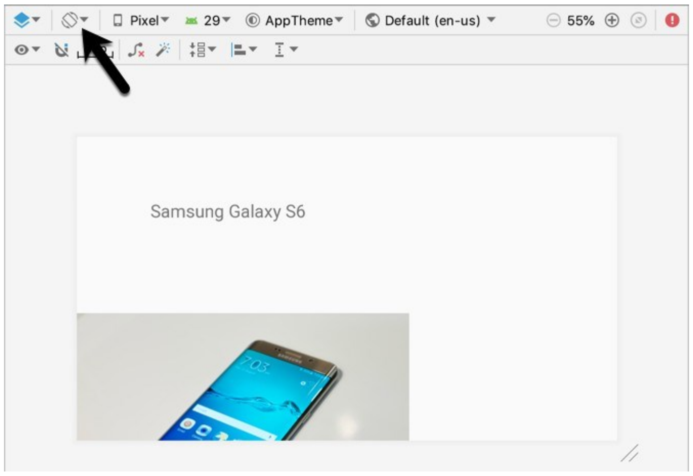


Figure 9

- The absence of constraints results in a layout that fails to adapt to the change in device orientation, leaving the content off center and with part of the image and all three `Buttons` positioned beyond the viewable area of the screen.
- Clearly some work still needs to be done to make this into a responsive user interface.

Adding the Constraints

- Constraints are the key to creating layouts that can adapt to device orientation changes and different screen sizes.
- Begin by rotating the layout back to portrait orientation and selecting the `TextView` widget located above the `ImageView`.
- With the widget selected, establish constraints from the left, right and top sides of the `TextView` to the corresponding sides of the parent `ConstraintLayout` as shown in Figure 10:

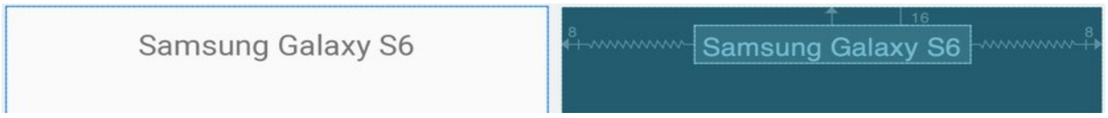


Figure 10

- With the `TextView` widget constrained, select the `ImageView` instance and establish opposing constraints on the left and right-hand sides with each connected to the corresponding sides of the parent layout.
- Next, establish a constraint connection from the top of the `ImageView` to the bottom of the `TextView` and from the bottom of the `ImageView` to the top of the center `Button` widget.

- If necessary, click and drag the `ImageView` so that it is still positioned in the vertical center of the layout.

- Next, establish constraints on the left and right hand sides of the `ImageView` to the corresponding sides of the `ConstraintLayout` container.

- With the `ImageView` still selected, use the Inspector in the attributes panel to change the top and bottom margins on the `ImageView` to 24 and 8 respectively and to change both the widget height and width dimension properties to `MATCH_CONSTRAINT` so that the widget will resize to match the constraints.

- These settings will allow the layout engine to enlarge and reduce the size of the `ImageView` when necessary to accommodate layout changes:

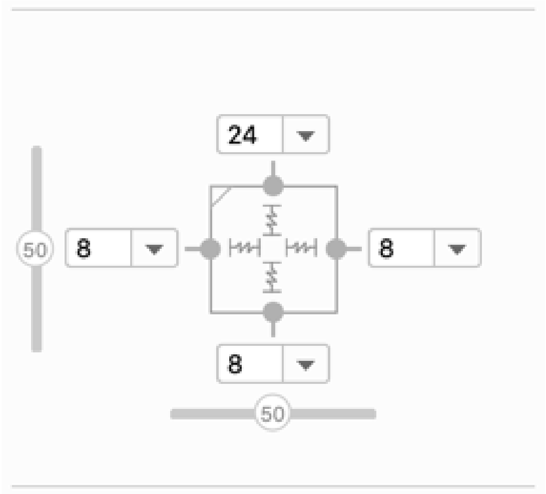


Figure 11

- Figure 12, shows the currently implemented constraints for the `ImageView` in relation to the other elements in the layout:

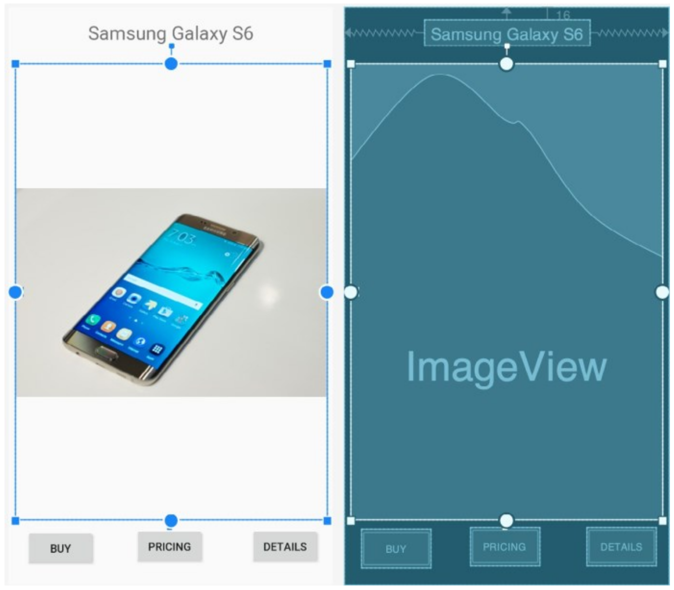


Figure 12

- The final task is to add constraints to the three `Button` widgets.
- For this example, the `Buttons` will be placed in a chain.
- Begin by turning on Autoconnect within the Layout Editor by clicking the toolbar `Button` highlighted earlier.
- Next, click on the Buy Now `Button` and then shift-click on the other two `Buttons` so that all three are selected.
- Right-click on the Buy Now `Button` and select the **Chains > Create Horizontal Chain** menu option from the resulting menu.
- By default, the chain will be displayed using the spread style which is the correct behavior for this example.
- Finally, establish a constraint between the bottom of the Buy Now `Button` and the bottom of the layout.
- Repeat this step for the remaining `Buttons`.
- On completion of these steps the `Buttons` should be constrained as outlined in Figure 13:

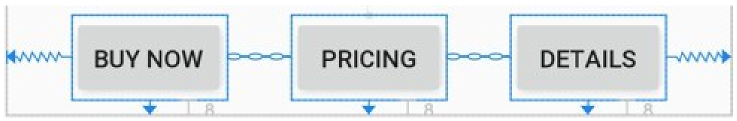


Figure 13

Testing the Layout

- With the constraints added to the layout, rotate the screen into landscape orientation and verify that the layout adapts to accommodate the new screen dimensions.
- While the Layout Editor tool provides a useful visual environment in which to design user interface layouts, when it comes to testing there is no substitute for testing the running app.
- Launch the app on a physical Android device or emulator session and verify that the user interface reflects the layout created in the Layout Editor.
- An Android Studio Layout Editor `ConstraintLayout` Tutorial Figure 14, for example, shows the running app in landscape orientation:

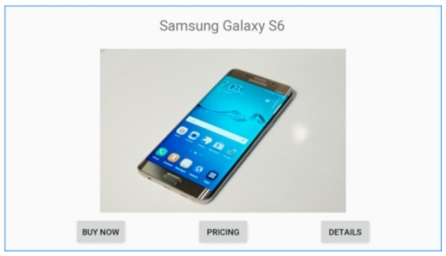
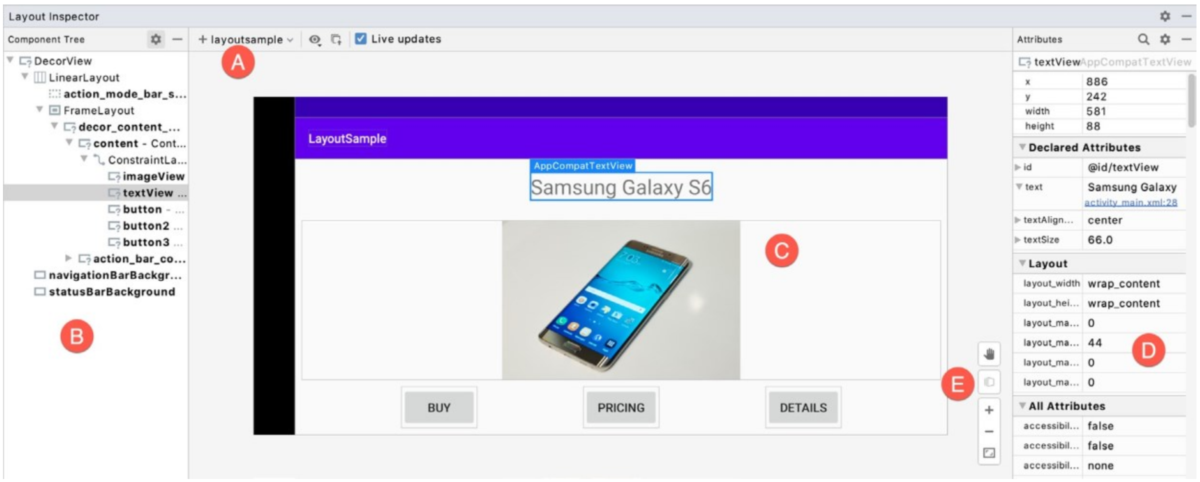


Figure 14

- The user interface design is now complete.
- Designing a more complex user interface layout is a continuation of the steps outlined above.
- Simply drag and drop views onto the display, position, constrain and set properties as needed.

Using the Layout Inspector

- The hierarchy of components that make up a user interface layout may be viewed at any time using the Layout Inspector tool.
- In order to access this information the app must be running on a device or emulator.
- Once the app is running, select the **Tools > Layout Inspector** menu option followed by the process to be inspected using the menu marked A in Figure 80 below).
- Once the inspector loads, the left most panel (B) shows the hierarchy of components that make up the user interface layout.
- The center panel (C) shows a visual representation of the layout design.
- Clicking on a widget in the visual layout will cause that item to highlight in the hierarchy list making it easy to find where a visual component is situated relative to the overall layout hierarchy.
- Finally, the rightmost panel (marked D in Figure 15) contains all of the property settings for the currently selected component, allowing for in-depth analysis of the component's internal configuration.
- Where appropriate, the value cell will contain a link to the location of the property setting within the project source code.



textView AppCompatTextView

x	886
y	242
width	581
height	88
Declared Attributes	
id	@id/textView
text	Samsung Galaxy activity_main.xml:28
textAlign...	center
textSize	66.0
Layout	
layout_width	wrap_content
layout_hei...	wrap_content
layout_ma...	0
layout_ma...	44
layout_ma...	0
layout_ma...	0
All Attributes	
accessibil...	false
accessibil...	false
accessibil...	none

Figure 15

- The `Button` marked E switches the view to 3D mode, "exploding" the hierarchy so that it can be rotated and inspected.

- This can be useful for tasks such as identifying obscured views:

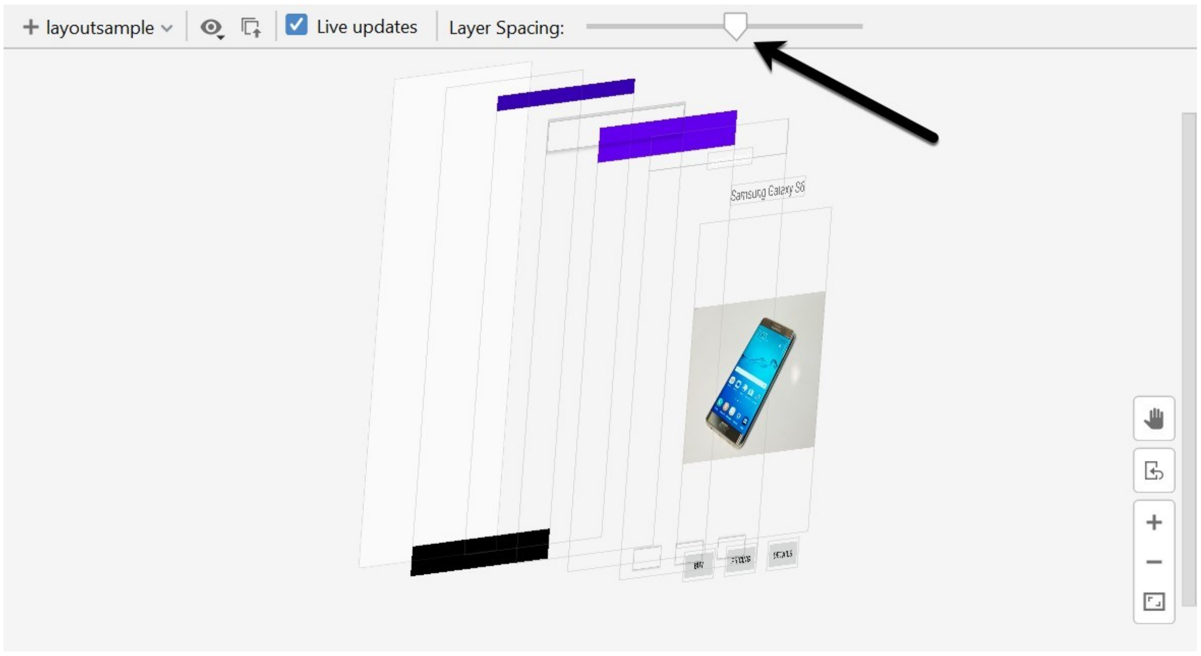


Figure 16

- Click and drag the rendering to rotate it in three dimensions, using the slider indicated by the arrow in the above figure to increase the spacing between the layers.

- Click the `Button` marked E in Figure 16 above to switch back to 2D mode.

- Finally, to follow conventions and good app development practice, go back and remove any hard-coded text, add the text to `strings.xml` and refer back to the names of the strings of text in the `activity_main.xml` file.

What to Turn In

Document any Java classes as described in *322 & 522 Android Coding and Documentation Guidelines* and zip any apps separately.

Attach your zipped file(s) to your assignment submission on Blackboard.