LAB 3 – Using Android Studio Layout Editor

At the end of this lab, students should be able to:

- Use Android Studio Layout Editor
- Create a ConstraintLayout-based user interface
- Use the Layout Inspector tool

Creating User Interfaces

Each element within a user interface screen of an Android application is a view that is ultimately subclassed from the android.view.View class.

Each view represents a rectangular area of the device display and is responsible both for what appears in that rectangle and for handling events that take place within the view's bounds. Multiple views may be combined to create a single composite view. The views within a composite view are children of a container view which is generally a subclass of android.view.ViewGroup (which is itself a subclass of android. view.View). A user interface is comprised of views constructed in the form of a view hierarchy.

The Android SDK includes a range of pre-built views that can be used to create a user interface. These include basic components such as text fields and buttons, in addition to a range of layout managers that can be used to control the positioning of child views. In the event that the supplied views do not meet a specific requirement, custom views may be created, either by extending or combining existing views, or by subclassing android.view.View and creating an entirely new class of view. User interfaces may be created using the Android Studio Layout Editor tool, handwriting XML layout resource files or by writing Java code.

Android Layout Managers

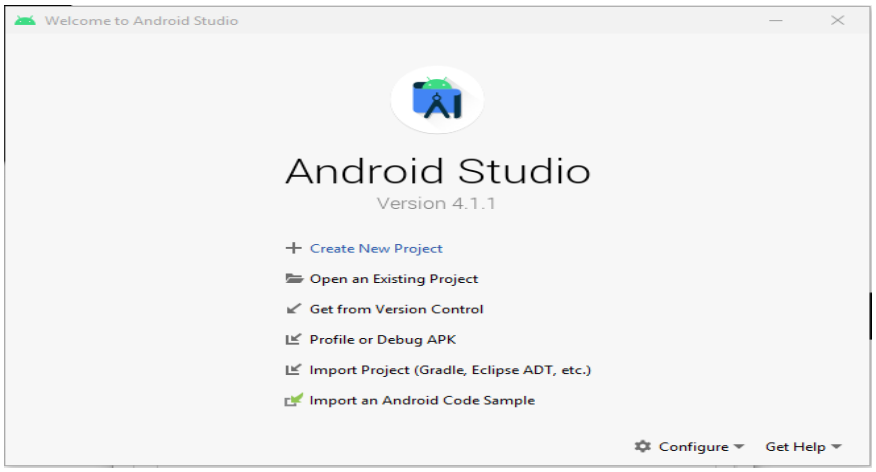
Inside the SDK also includes a set of views referred to as layouts. Layouts are container views (and, therefore, subclassed from ViewGroup) designed for the sole purpose of controlling how child views are positioned on the screen.

Layout Views	Description
ConstraintLayout	Introduced in Android 7, use of this layout manager is recommended for most layout requirements. ConstraintLayout allows the positioning and behavior of the views in a layout to be defined by simple constraint settings assigned to each child view. The flexibility of this layout allows complex layouts to be quickly and easily created without the necessity to nest other layout types inside each other, resulting in improved layout performance. ConstraintLayout is also tightly integrated into the Android Studio Layout Editor tool.

An Android Studio Layout Editor ConstraintLayout Tutorial

a) Creating the No Activity Project

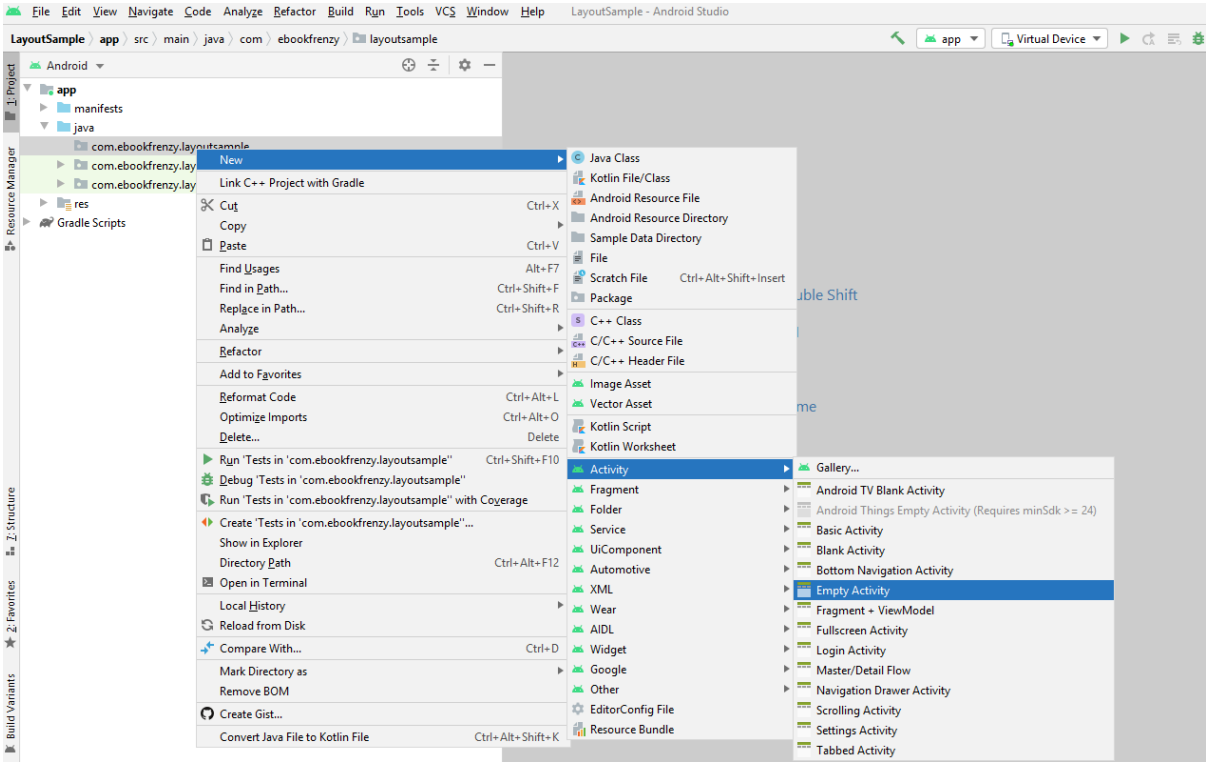
i. Go to File -> New Project



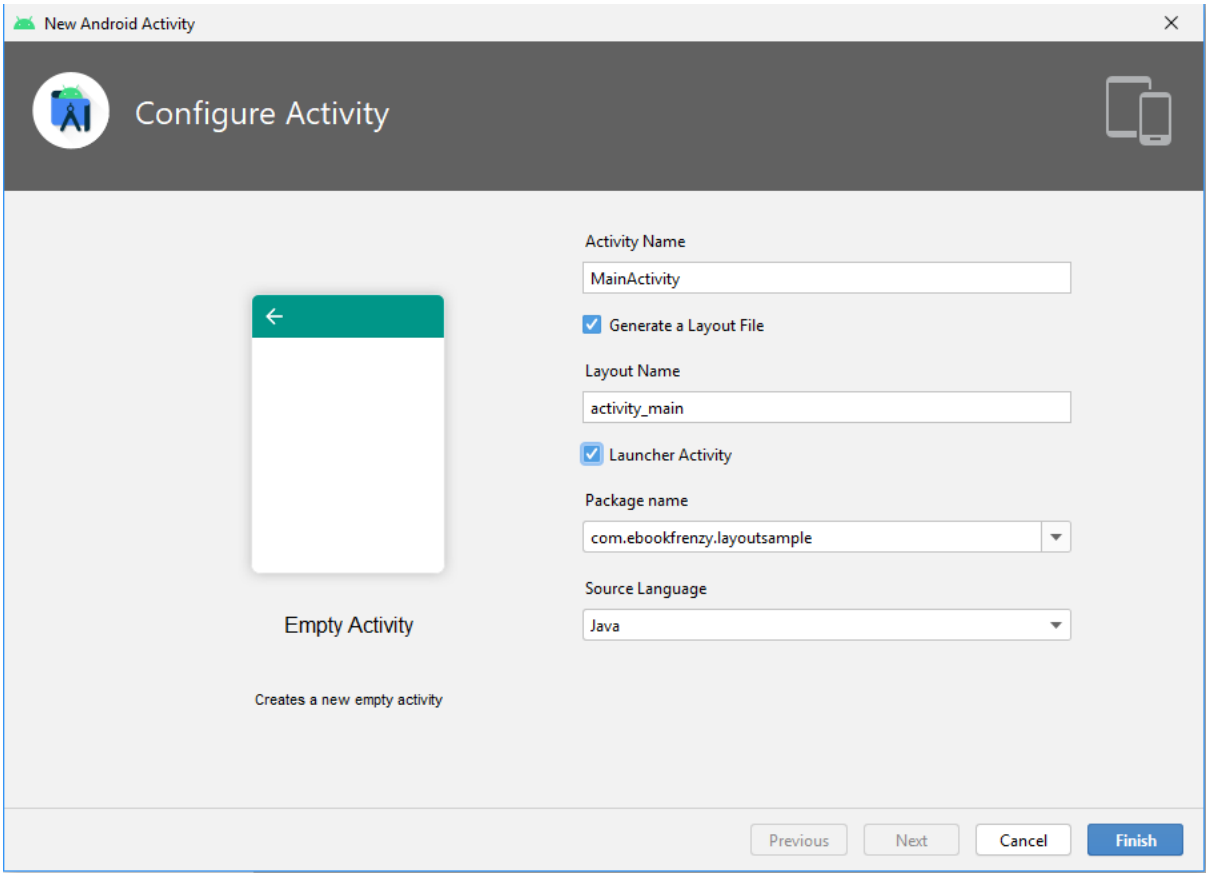
ii. Choose **No activity** template, then click Next

iii. Enter LayoutSample into the Name field and specify com.ebookfrenzy.layoutsample as the package name.
Change the Minimum API level setting to API26: Android 8.0(Oreo) and the Language menu to Java.
Click Finish.

iv. Once the package name is visible, right-click on it and select the *New -> Activity -> Empty Activity* menu option as illustrated in figure below:

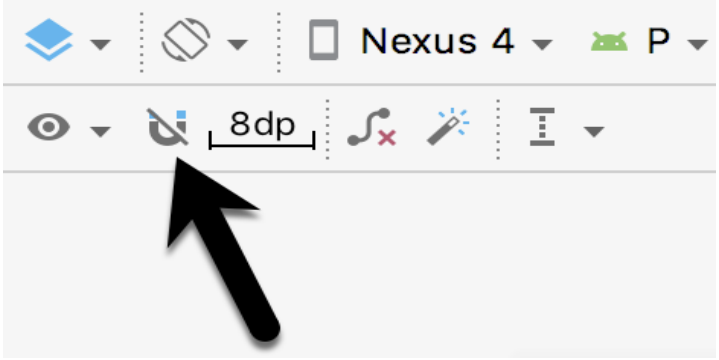


v. In order for an application to be able to run on a device it needs to have an activity designated as the launcher activity. Without a launcher activity, the operating system will not know which activity to start up when the application first launches and the application will fail to start. Since this example only has one activity, it needs to be designated as the launcher activity for the application so make sure that the Launcher Activity option is enabled before clicking on the Finish button.



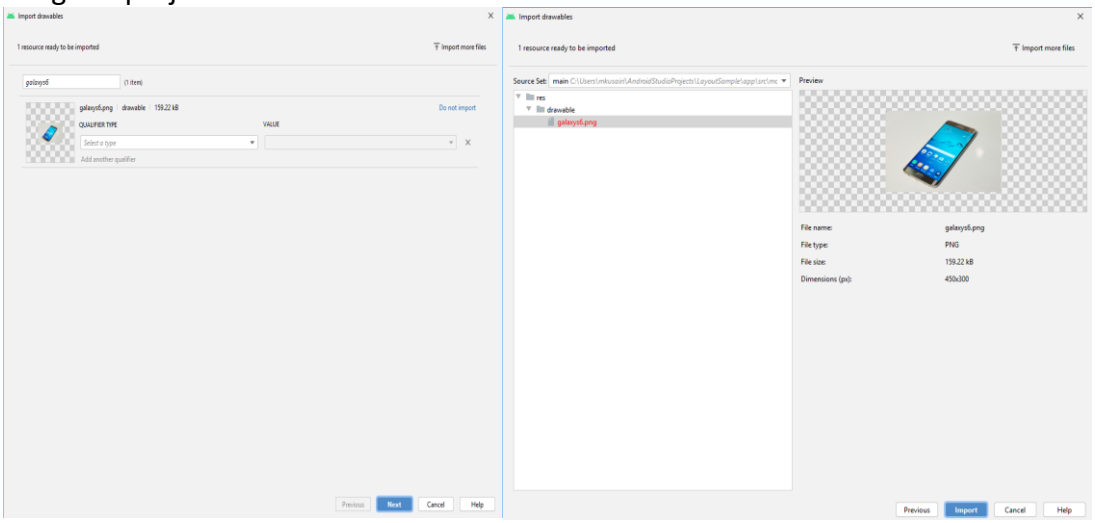
b) Preparing the Layout Editor Environment

i. Open activity_main.xml. Turn off the Autoconnect button if the Autoconnect button is on. 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.

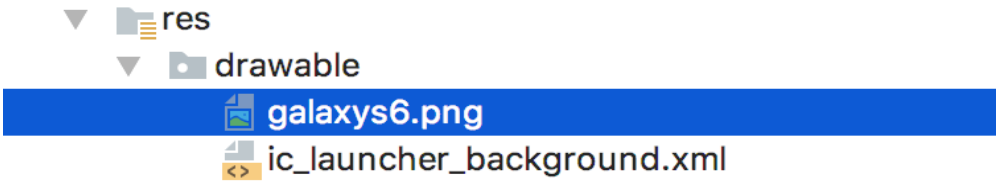


i. Within the Android Studio, display the Resource Manager tool window (View -> Tool Windows-> Resource Manager). Drag and drop the galaxy6s.png image onto the Resource Manager tool window.

ii. In the resulting dialog below, click *Next* followed by the *Import* button to add the image to project.

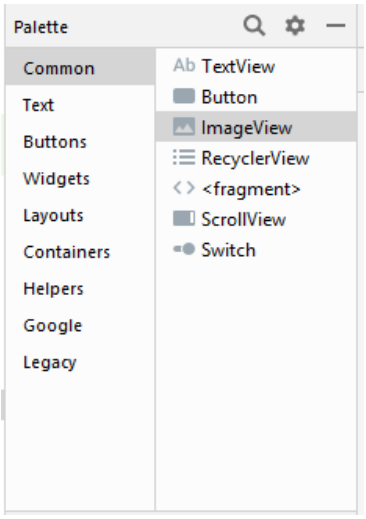


iii. The image will appear in the *res -> drawables* section of the Project tool window:

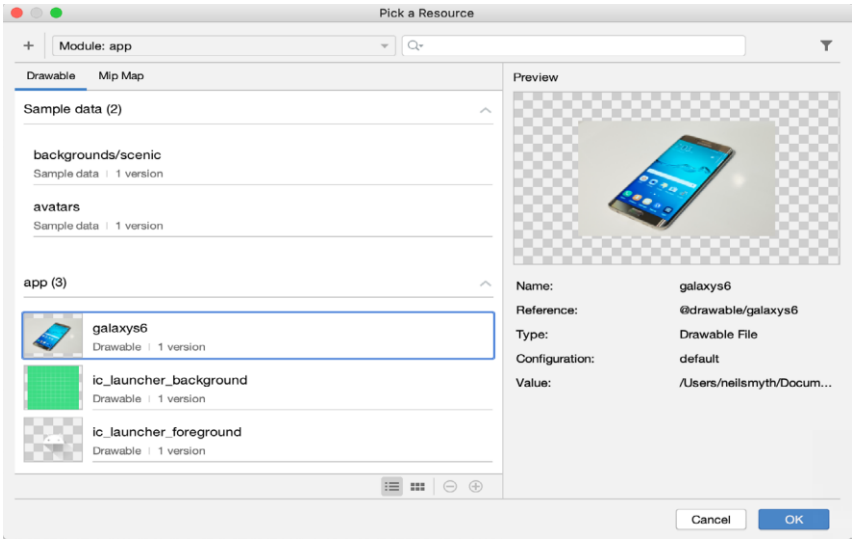


c) Adding the widget to the user interface

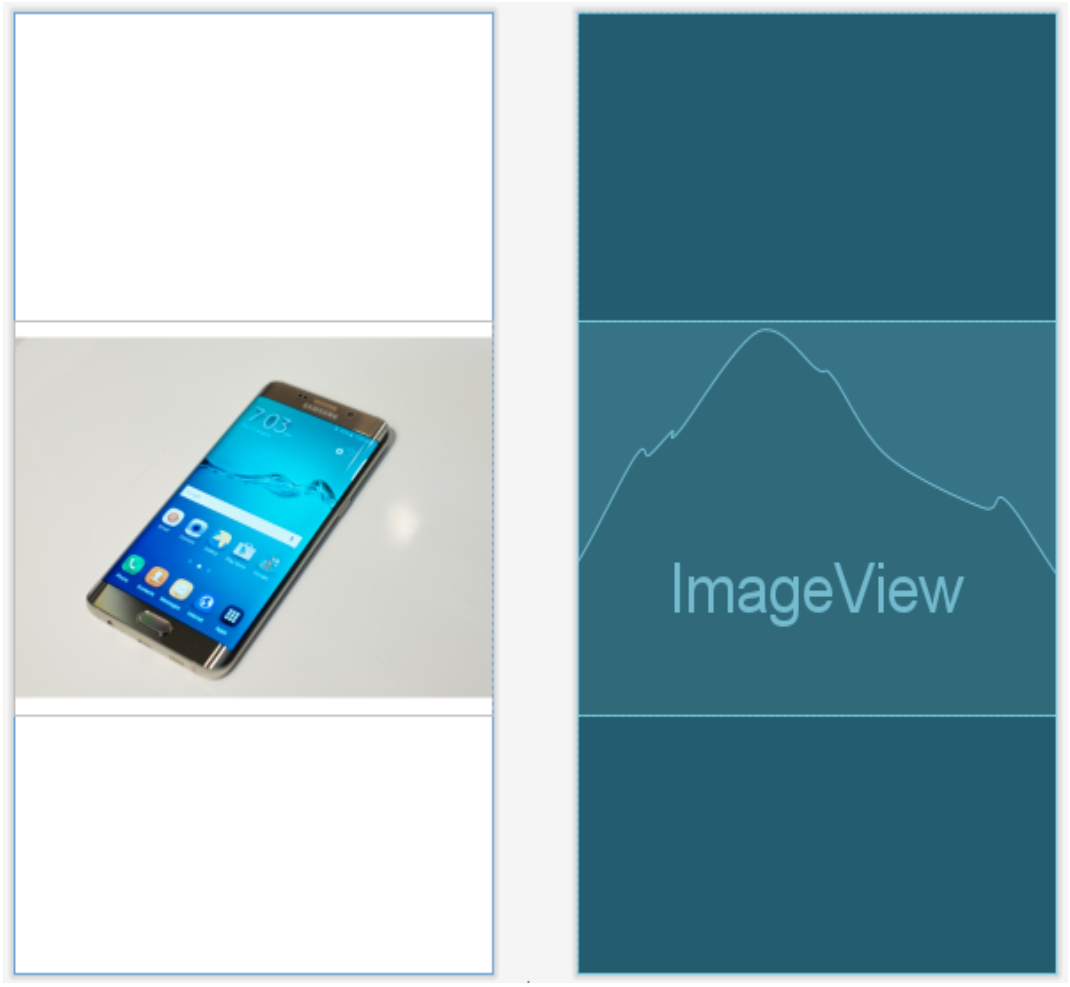
i. Within the Common palette category, drag an ImageView object into the center of the display view.



ii. Once placed within the layout, the Resources dialog will appear seeking the image to be displayed within the view. Select the galaxy6s.png and click OK.

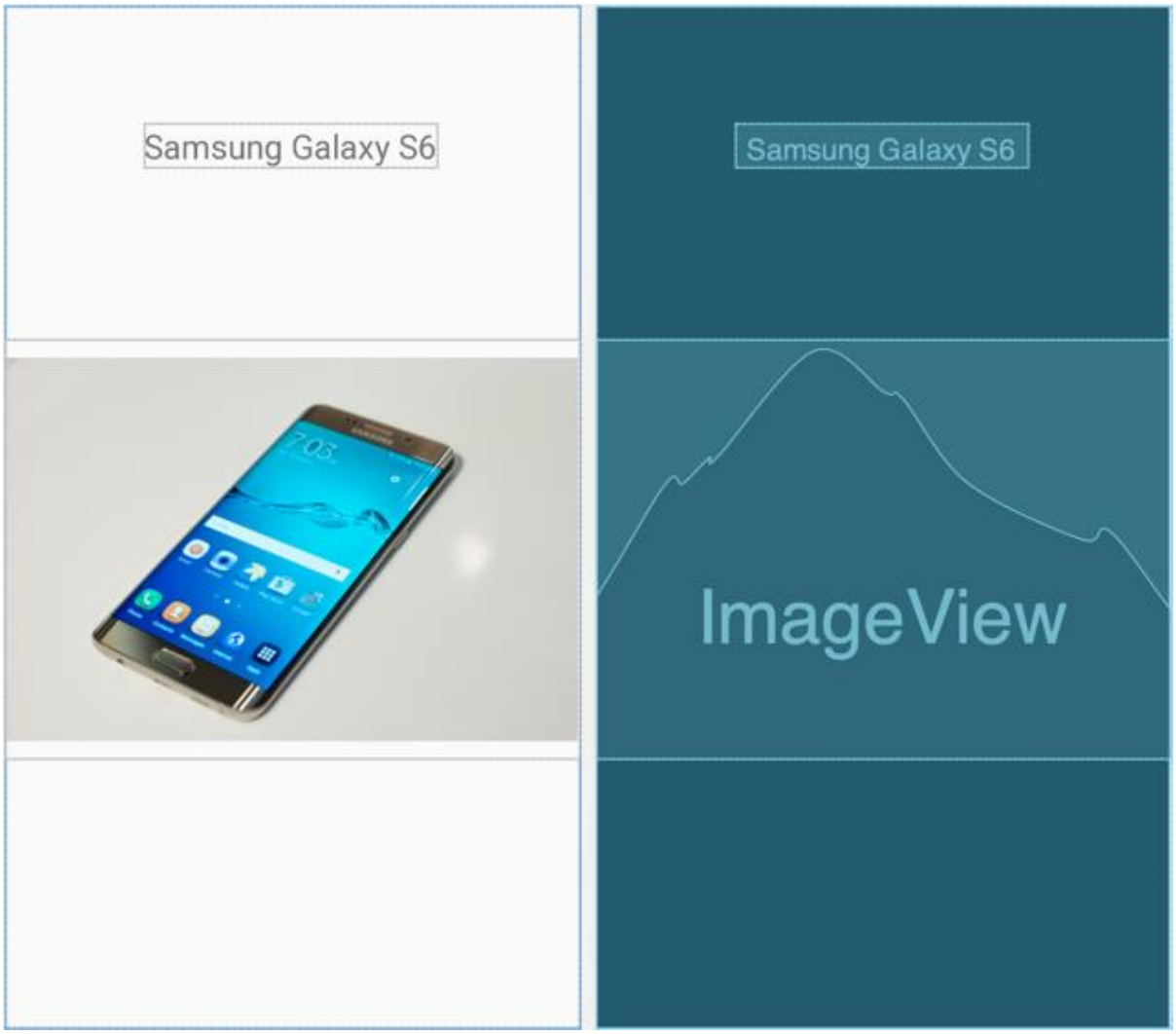


iii. Adjust the size of the ImageView using the resize handles and reposition it in the center of the layout as figure below:



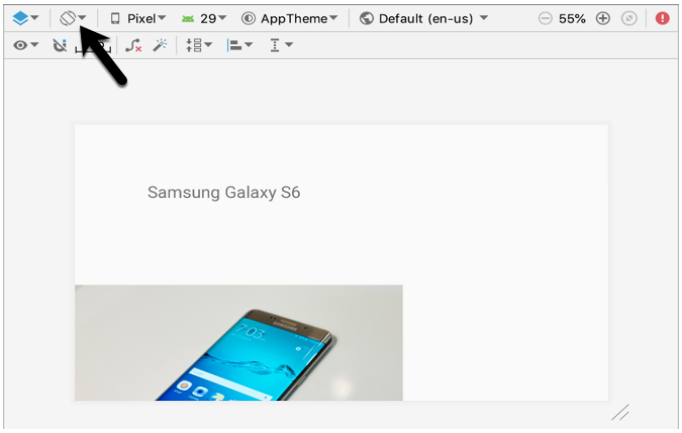
iv. Click and drag a TextView object from the *Common* section of the palette and position it above the ImageView.

v. Using the Attributes panel, unfold the *textAppearance* attribute entry in the Common Attributes section, change the *textSize* property to 24sp, the *textAlignment* setting to center and the text to “Samsung Galaxy S6” as illustrated in figure below:



vi. 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” as figure below:

vii. 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.
Use the device rotation button located in the Layout Editor toolbar to view the user interface in landscape orientation as figure below:



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.

d) Adding the Constraints

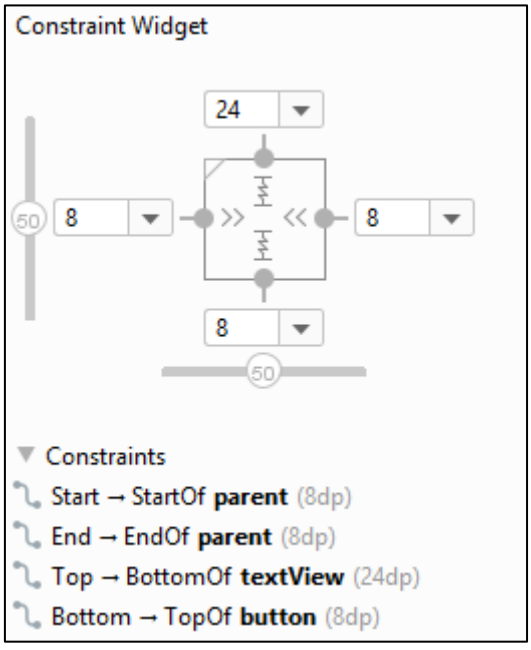
Constraints are the key to creating layouts that can adapt to device orientation changes and different screen sizes.

i. Rotate the layout back to portrait orientation and select the TextView widget located above the ImageView. With the widget selected, establish constraints from the left, right and top sides of the TextView.

ii. Select the ImageView instance and establish constraints on the left and right-hand sides with each connected to the corresponding sides of the parent layout.

iii. 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.

iv. Use the attribute panel to change the top and bottom margins on the ImageView to 24 and 8 respectively as below figure.



v. Then, change both the widget height and width dimension properties to *match_constraint* so that the widget will resize to match the constraints.

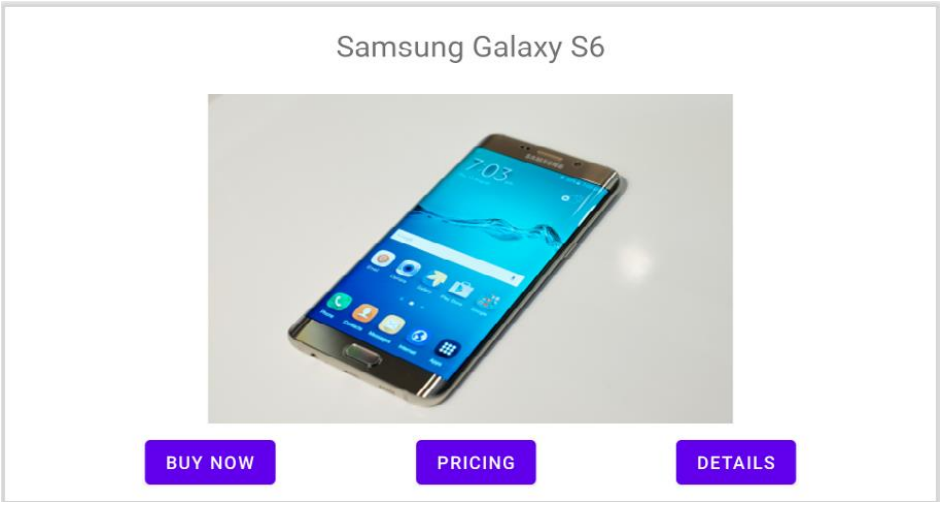
vi. The final task is to add constraints to the three Button widget. For this example, the buttons will be placed in a chain.

vii. First, turn on the Autoconnect.
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.

viii. 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. The completion of the buttons constrained as below figure:

e) Testing the Layout

i. Rotate the screen into landscape orientation and verify that the layout adapts to accommodate the new screen dimensions as figure below.

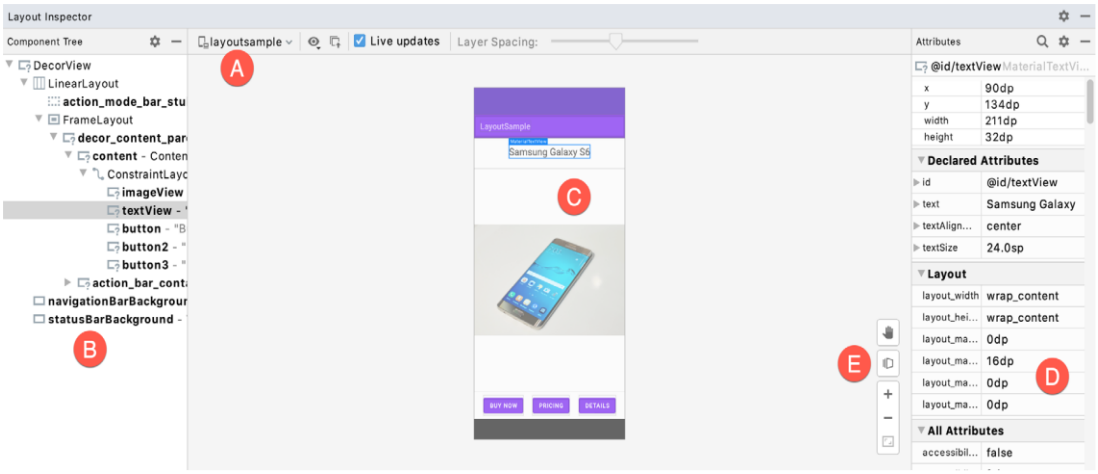


ii. Launch the app on a physical Android device or emulator to verify that the user interface reflects the layout created in the Layout Editor.

f) Using the Layout Inspector

i. 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.

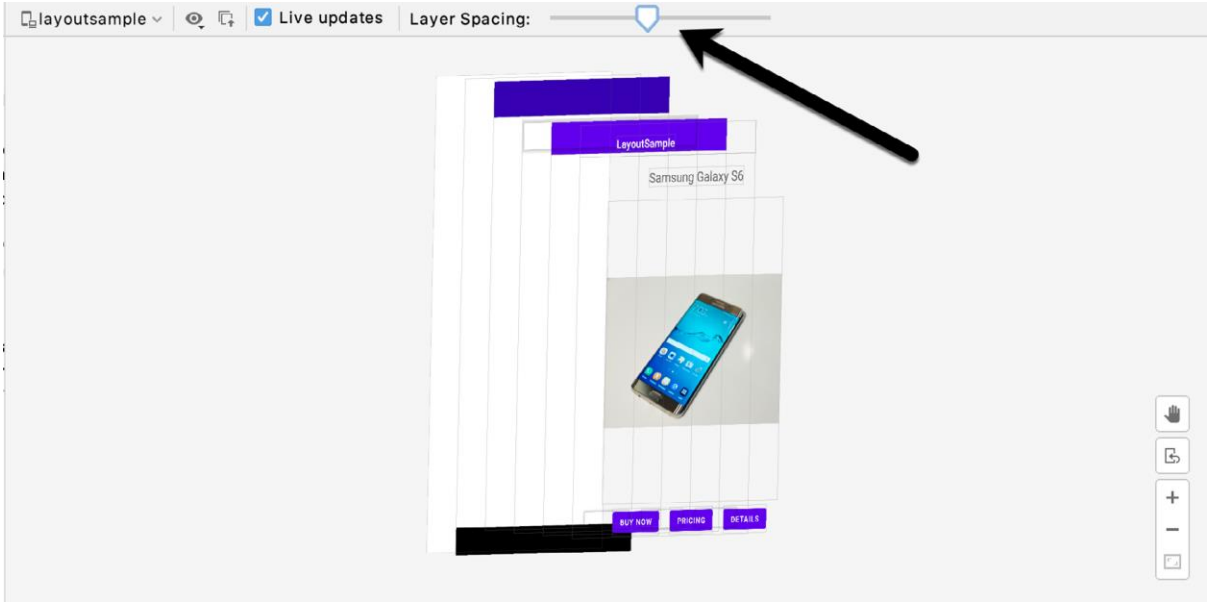
ii. Once the app is running, select the Tools -> Layout Inspector menu.



The details about each section in the Layout tools as in the table below:

Label	Description
A	The menu
B	Shows the hierarchy of components that make up the user interface layout
C	Shows a visual representation of the layout design
D	Contains all of the property settings for the currentlyselected component, allowing for in-depth analysis of the component's internal configuration
E	To view the layout in 3D, click and drag anywhere on the layout preview area. This displays an "exploded"representation of the hierarchy so that it can be rotated and inspected.

iii. 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.



Exercise:

Design another view by adding different image and different widget component such as textbox, radio button etc.