
IT332: Mobile Application Development

Lecture # 13 : Creating User Interfaces with Layouts and Widgets

Muhammad Imran



android



Outline

- Preparing Resources
- Using the Graphical Layout Tool
- Introducing ConstraintLayout
- Applying the Constraint Layout
- Screen pixel densities and dp and sp
- Styles, themes, and theme attributes






Final Objective Today

- Now we will be adding some style to list items in the RecyclerView
- We will be using ConstraintLayout for that purpose.



Preparing Resources

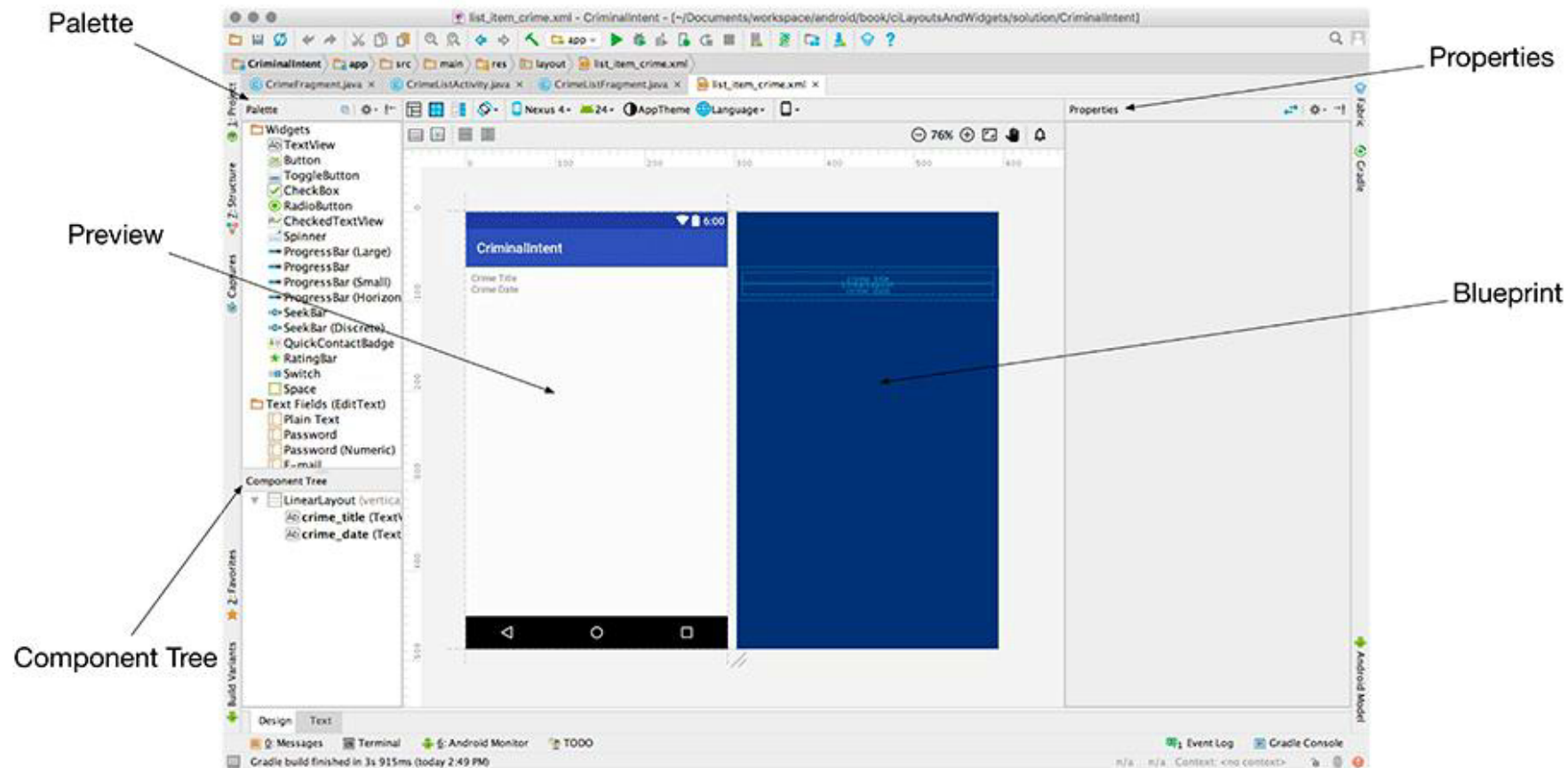
- You will need a copy of that fancy handcuff image
- Copy each density version of ic_solved.png into the appropriate drawable folder

 drawable-hdpi
 drawable-mdpi
 drawable-xhdpi
 drawable-xxhdpi
 drawable-xxxhdpi



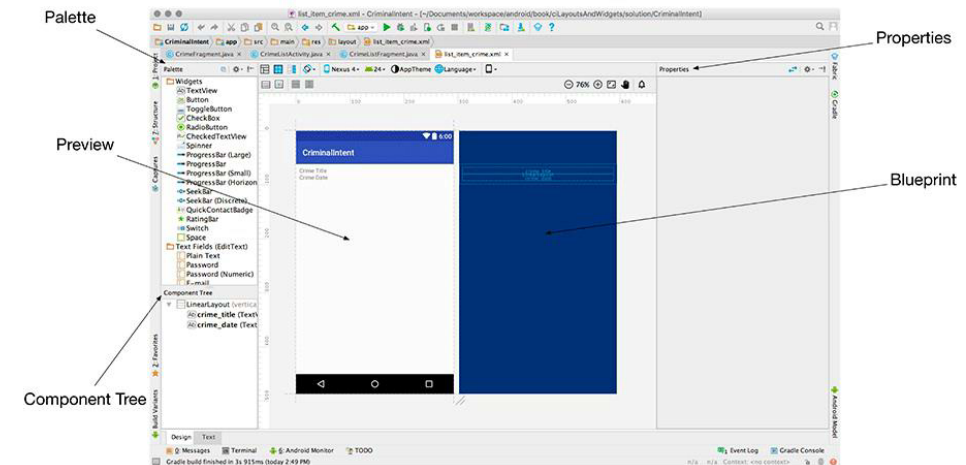
Using the Graphical Layout Tool

- So far, you have created layouts by typing XML.
- We can also use Android Studio's graphical layout tool.



Using the Graphical Layout Tool

- In the middle of the graphical layout tool is the preview
- The blueprint view shows an outline of each of your views to see how big each view is
- Palette contains all the widgets organized by category
- The component tree shows how the widgets are organized in the layout.
- Properties view is used to view and edit the attributes of the selected widget



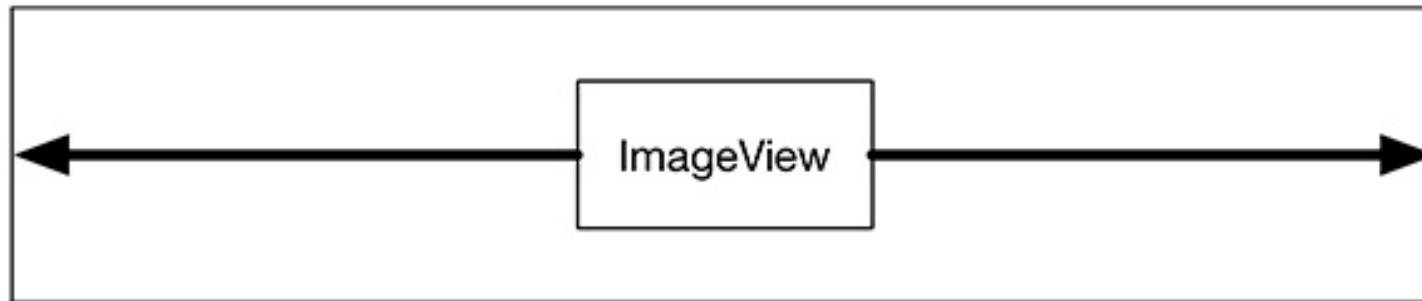
Introducing ConstraintLayout

- With ConstraintLayout, instead of using nested layouts you add a series of constraints to your layout.
- A constraint is like a rubber band which pulls two things toward each other.
- So, for example, you can attach a constraint from the right edge of an ImageView to the right edge of its parent (the ConstraintLayout itself)
- The constraint will hold the ImageView to the right.



Introducing ConstraintLayout

- You can create a constraint from all four edges of your ImageView (left, top, right, and bottom).
- If you have opposing constraints, they will equal out and your ImageView will be right in the center of the two constraints



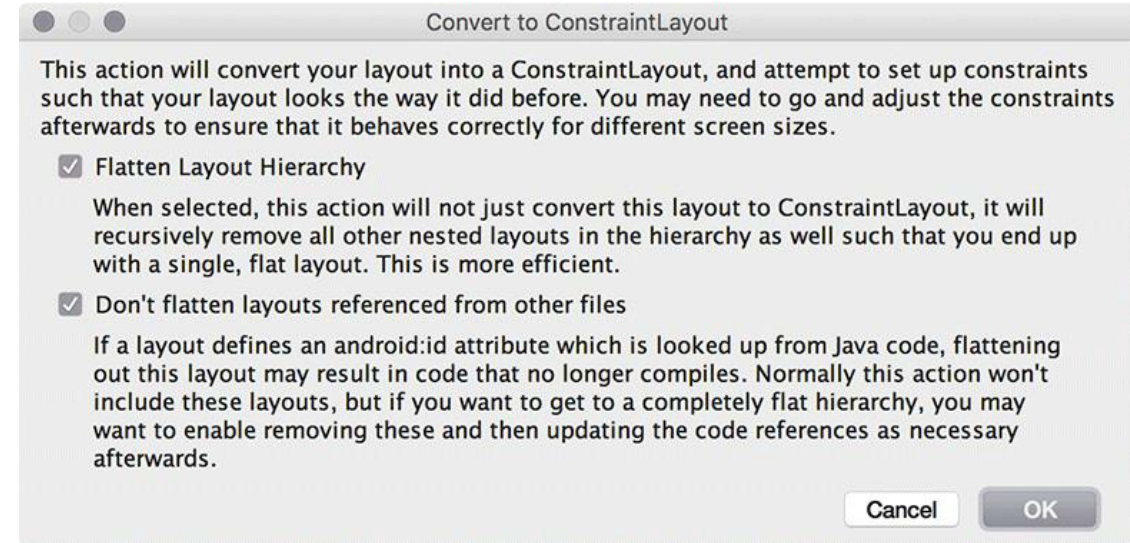
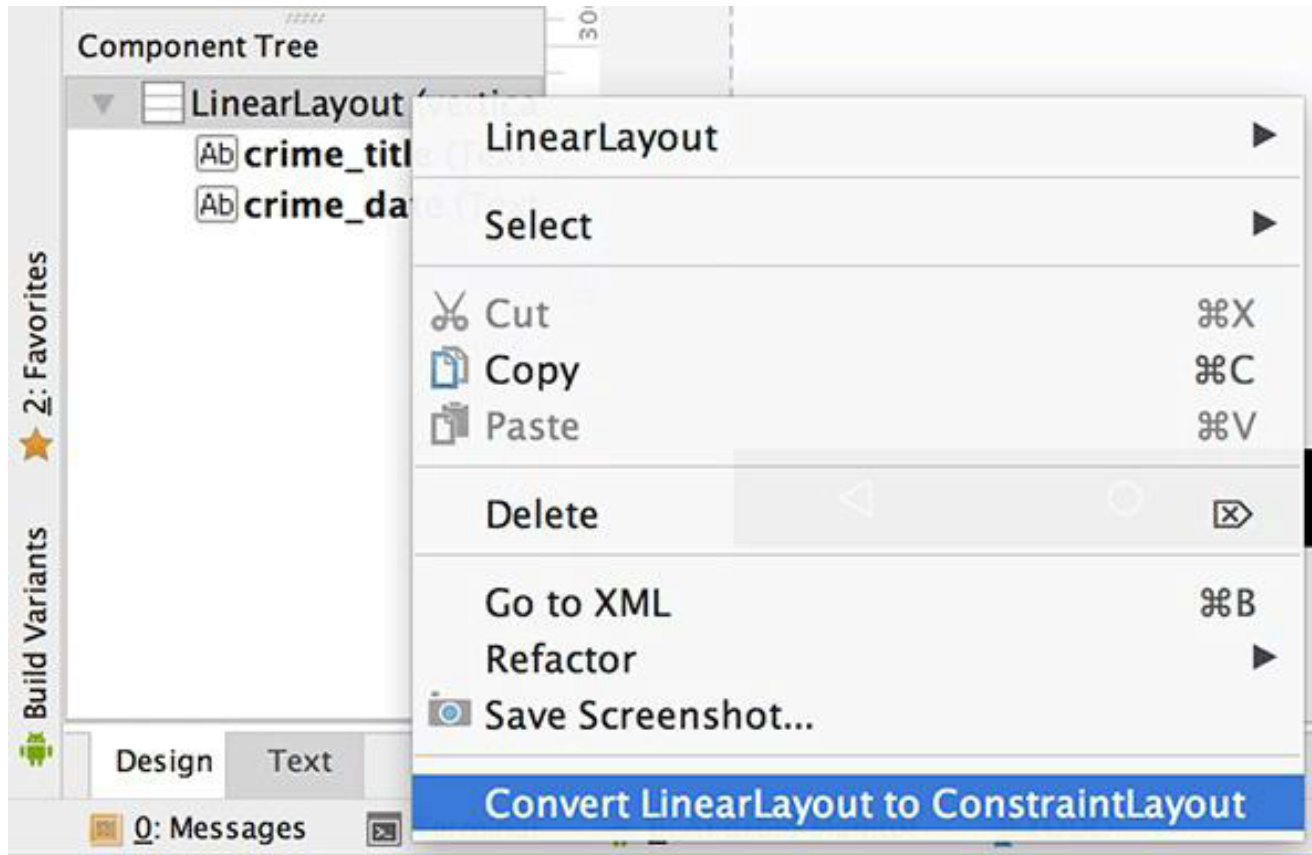
To place views where you want them to go in a ConstraintLayout, you give them constraints instead of dragging them around the screen.

Introducing ConstraintLayout

- For sizing widgets you have three options:
 - Let the widget decide (your old friend `wrap_content`),
 - decide for yourself, or
 - let your widget expand to fit your constraints.

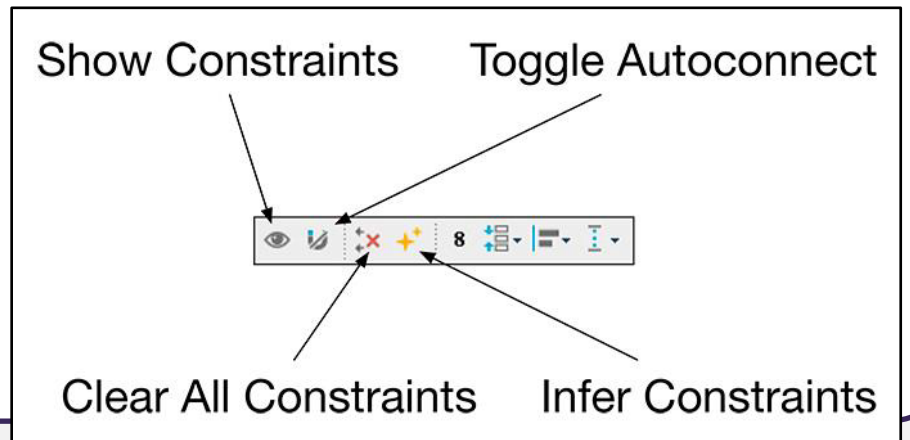
Converting to ConstraintLayout

- You can convert already existing layout to a ConstraintLayout.



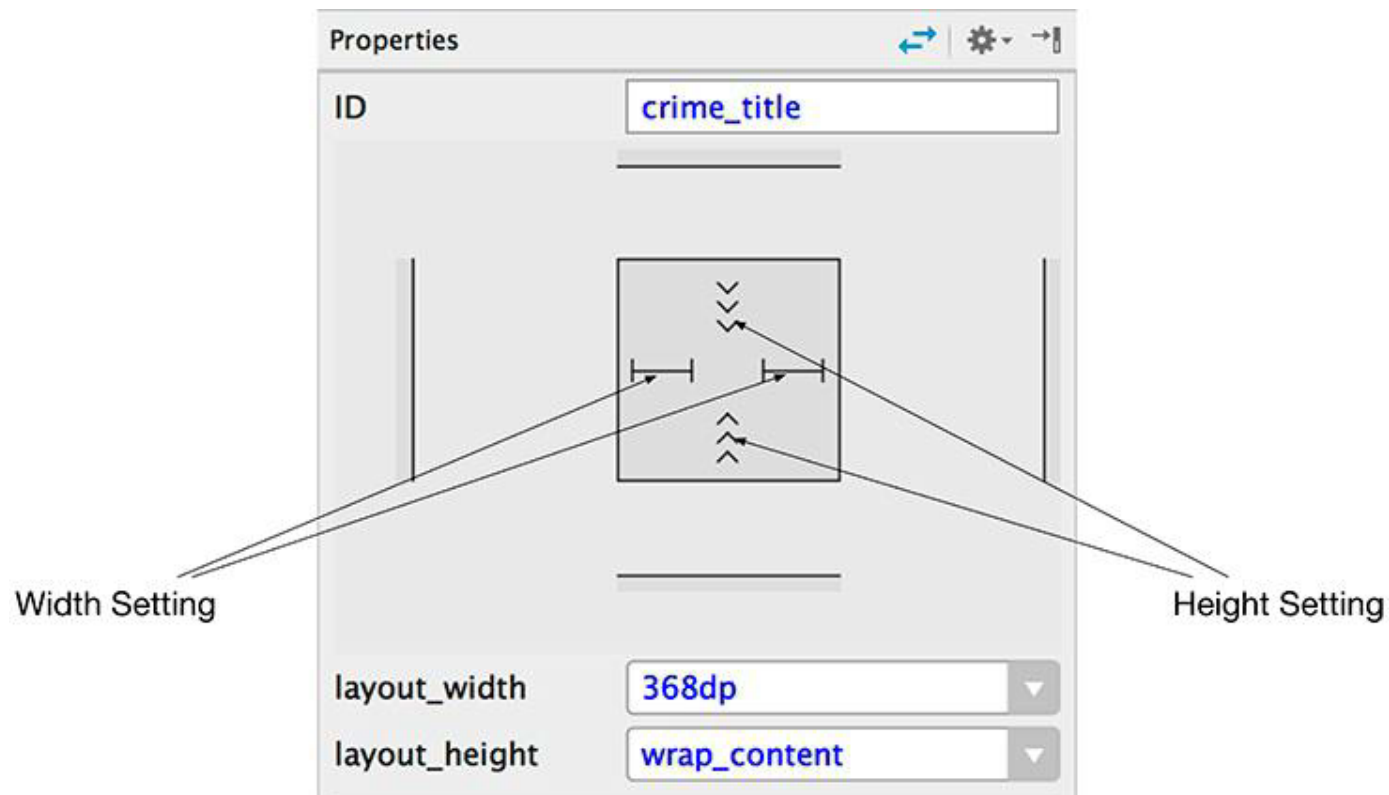
The Graphical Editor (Constraint controls)

Show Constraints	Show Constraints will reveal the constraints that are set up in the preview and blueprint views.
Toggle Autoconnect	When autoconnect is enabled, constraints will be automatically configured as you drag views into the preview. Android Studio will guess the constraints that you want a view to have and make those connections on demand.
Clear All Constraints	Clear All Constraints will remove all existing constraints in this layout file.
Infer Constraints	Android Studio will automatically create constraints for you, but it is only triggered when you select this option.



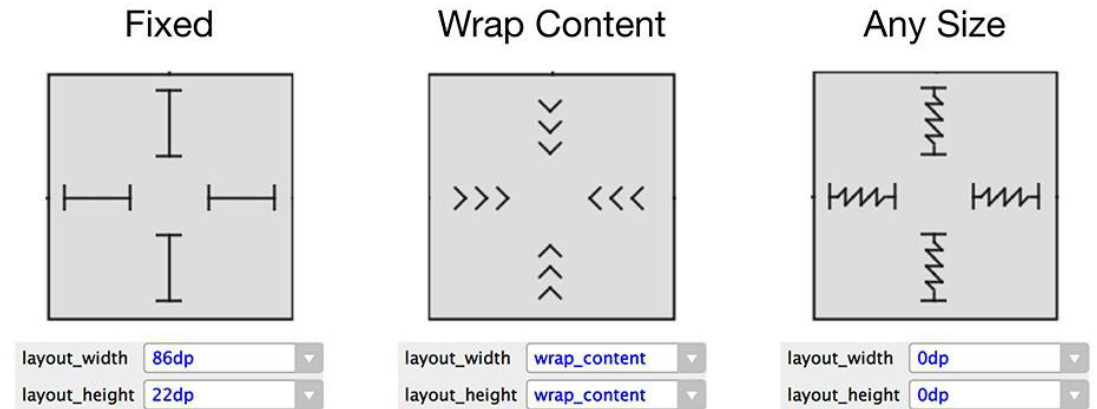
View Size Setting

- If you select a view in the component tree you can set the properties for width and height



View Size Setting

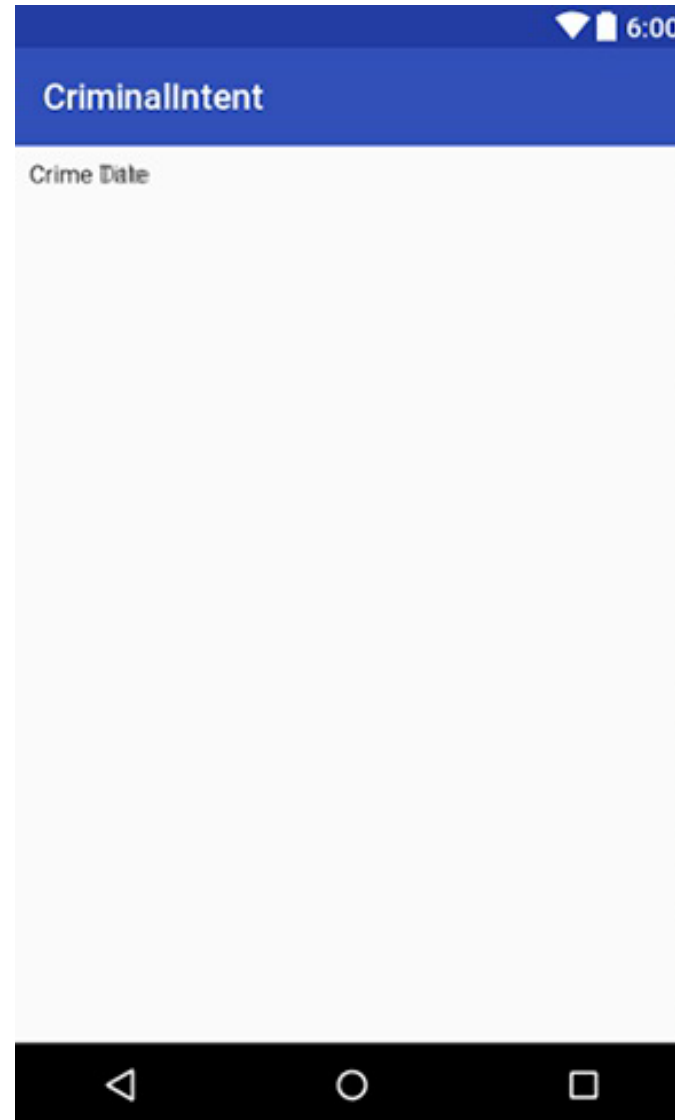
- These properties can be set to three view size settings each of which corresponds to a value for `layout_width` or `layout_height`.



Setting type	Setting value	Usage
fixed	Xdp	Specifies an explicit size (that will not change) for the view. The size is specified in dp units. (More on dp units later in this chapter.)
wrap content	wrap_content	Assigns the view its “desired” size. For a TextView , this means that the size will be just big enough to show its contents.
any size	0dp	Allows the view to stretch to meet the specified constraints.

Views without any Constraints

- If we do not add any constraints to the Views that are part of ConstraintLayout, they will move to Top Left of the screen and also overlap each other

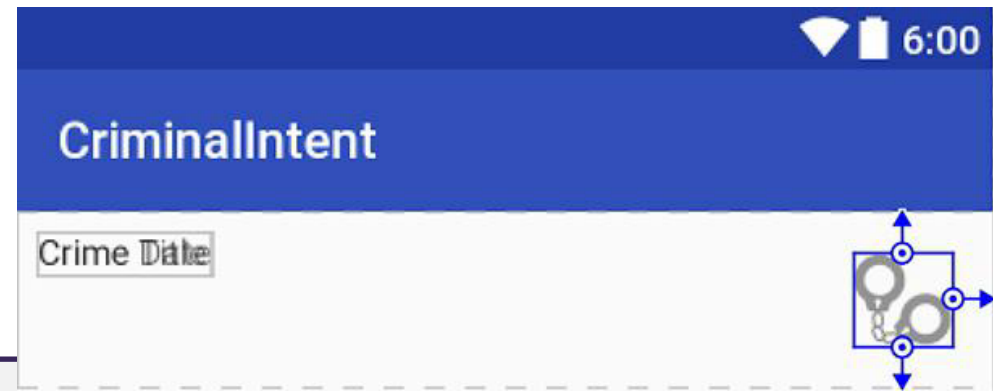
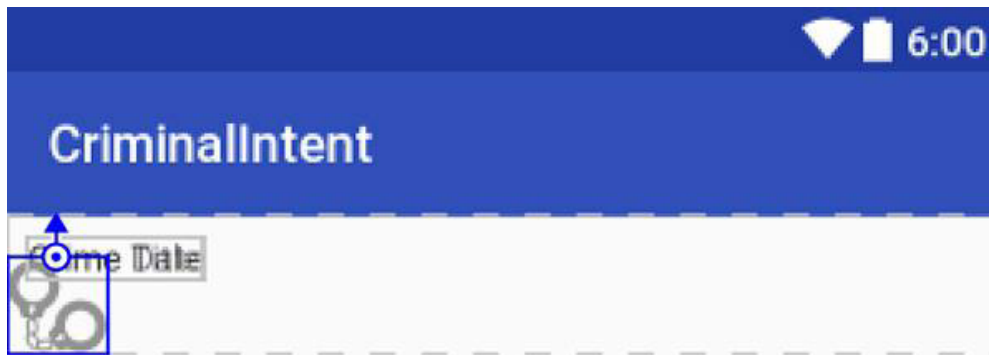


Applying Constraints to an Image View

- We can add an ImageView in our layout, but it will have no constraints so it will have no meaningful position



- To add a TOP Constraint, drag the top constraint handle from the ImageView to the top of the ConstraintLayout
- And similarly, other constraints can be added



Applying Constraints to other Views

- We can add constraints on a view with reference to other views



Screen pixel densities and dp and sp

- Sometimes you need to specify values for view attributes in terms of specific sizes (usually in pixels, but sometimes points, millimeters, or inches).
- You see this most commonly with attributes for text size, margins, and padding.
- Text size is the pixel height of the text on the device's screen.
- Margins specify the distances between views, and padding specifies the distance between a view's outside edges and its content.
- Android automatically scales images to different screen pixel densities using density-qualified drawable folders

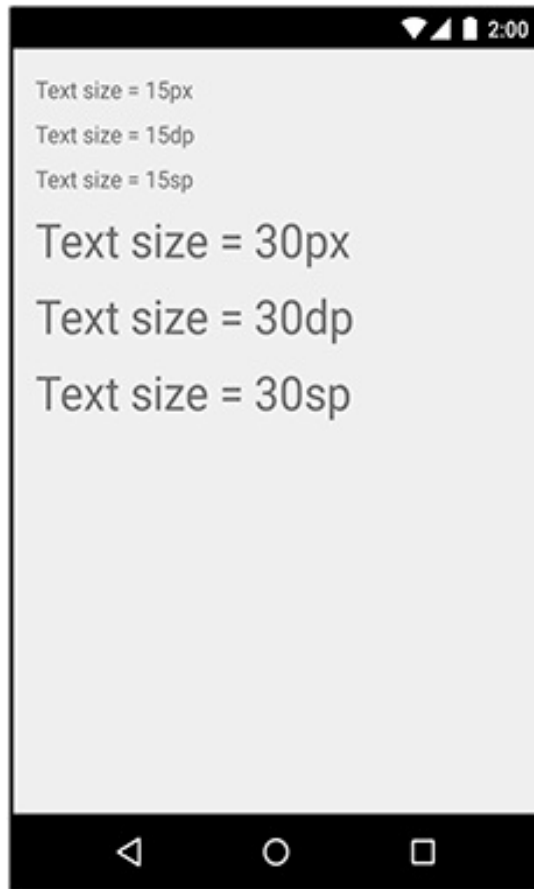
Screen pixel densities and dp and sp

- But problem occurs when the images scale, but margins do not? Or when the user configures a larger-than-default text size
- density-independent dimension units can be used to solve these problems
- It can be used to get the same size on different screen densities.
- Android translates these units into pixels at runtime, so there is no tricky math involved

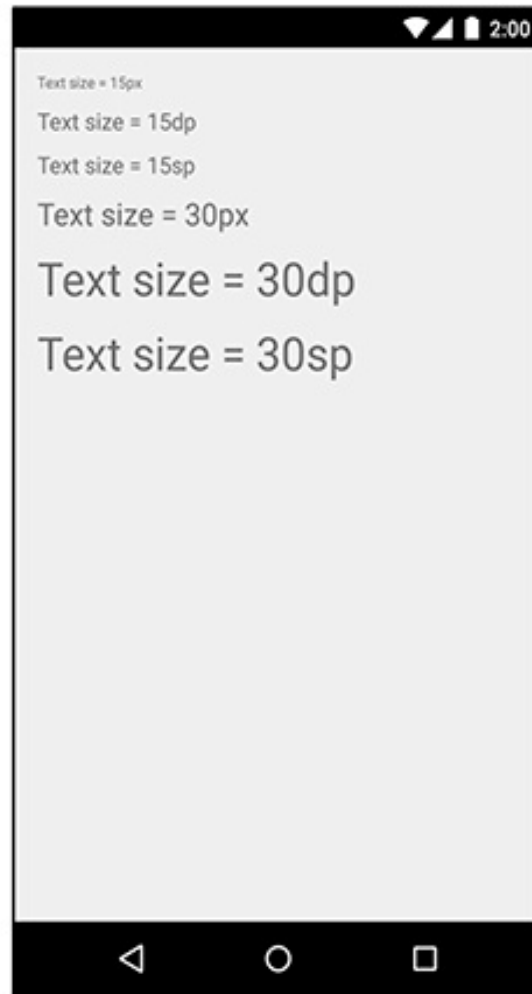
Screen pixel densities and dp and sp

px	Short for <i>pixel</i> . One pixel corresponds to one onscreen pixel, no matter what the display density is. Because pixels do not scale appropriately with device display density, their use is not recommended.
dp (or dip)	Short for <i>density-independent pixel</i> and usually pronounced “dip.” You typically use this for margins, padding, or anything else for which you would otherwise specify size with a pixel value. One dp is always 1/160th of an inch on a device’s screen. You get the same size regardless of screen density: When your display is a higher density, density-independent pixels will expand to fill a larger number of screen pixels.
sp	Short for <i>scale-independent pixel</i> . Scale-independent pixels are density-independent pixels that also take into account the user’s font size preference. You will almost always use sp to set display text size.
pt, mm, in	These are scaled units, like dp, that allow you to specify interface sizes in points (1/72 of an inch), millimeters, or inches. However, we do not recommend using them: Not all devices are correctly configured for these units to scale correctly.

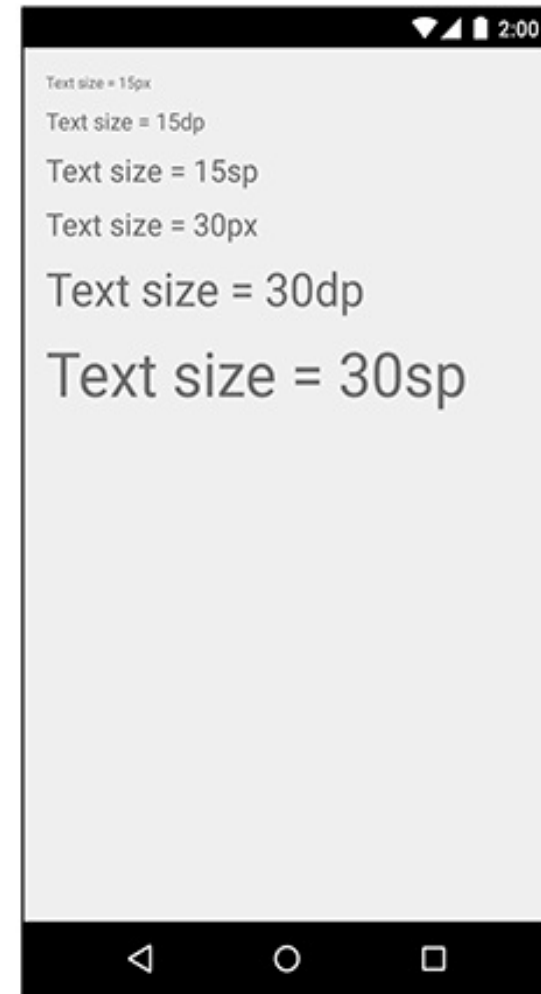
Screen pixel densities and dp and sp



MDPI



HDPI



HDPI, large text

Styles, themes, and theme attributes

- A style is an XML resource that contains attributes that describe how a widget should look and behave.
- For example, the following is a style resource that configures a widget with a larger-than-normal text size:

```
<style name="BigTextStyle">  
    <item name="android:textSize">20sp</item>  
    <item name="android:padding">3dp</item>  
</style>
```

- You can create your own styles and add them to a styles file in res/values/ and refer to them in layouts like this: @style/my_own_style.

Styles, themes, and theme attributes

- You can apply a style from the app's theme to a widget using a theme attribute reference. E.g. `?android:listSeparatorTextViewStyle`.

```
<TextView
    style="?android:listSeparatorTextViewStyle"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Title"/>
```

- In a theme attribute reference, you tell Android's runtime resource manager, "Go to the app's theme and find the attribute named `listSeparatorTextViewStyle`. This attribute points to another style resource. Put the value of that resource here."

Hands-on Videos

- You can watch the complete implementation example at the following link

<https://youtu.be/g9ZDdZd3IMA>

https://youtu.be/EkLT_JqkPAE

InClass Task 05

- The `Date` object is more of a timestamp than a conventional date. A timestamp is what you see when you call **`toString()`** on a `Date`, so that is what you have on in each of your `RecyclerView` rows. While timestamps make for good documentation, it might be nicer if the rows just displayed the date as humans think of it – like “Jul 22, 2016.” You can do this with an instance of the **`android.text.format.DateFormat`** class. The place to start is the reference page for this class in the Android documentation.
- You can use methods in the **`DateFormat`** class to get a common format. Or you can prepare your own format string. For a more advanced challenge, create a format string that will display the day of the week as well – for example, “Friday, Jul 22, 2016.”

Recommended Readings

- Page # 181 to 204, Chapter 09: Creating User Interfaces with Layouts and Widgets from Android Programming: The Big Nerd Ranch Guide, 3rd Edition by Bill Phillips, 2017