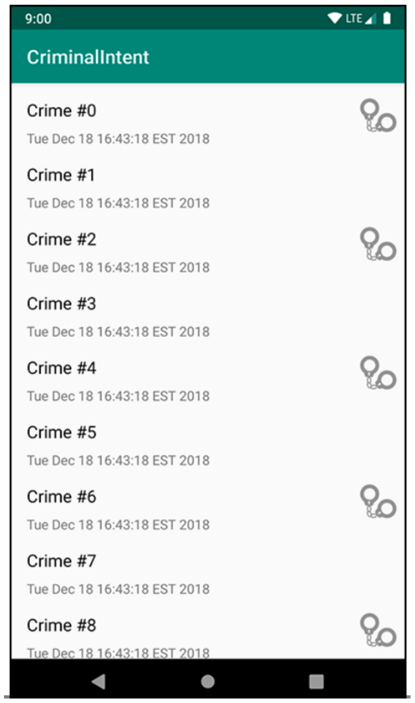
Level/Subject	:	**D3/Programming 6**
Topic	:	UI in Android apps
Week	:	4
Activity	:	Creating UI with Layout & Widgets in Android apps using Android Studio
Alocated time	:	60 mins labs
Deliverables	:	Project folder
Due date	:	end of week

Competency :

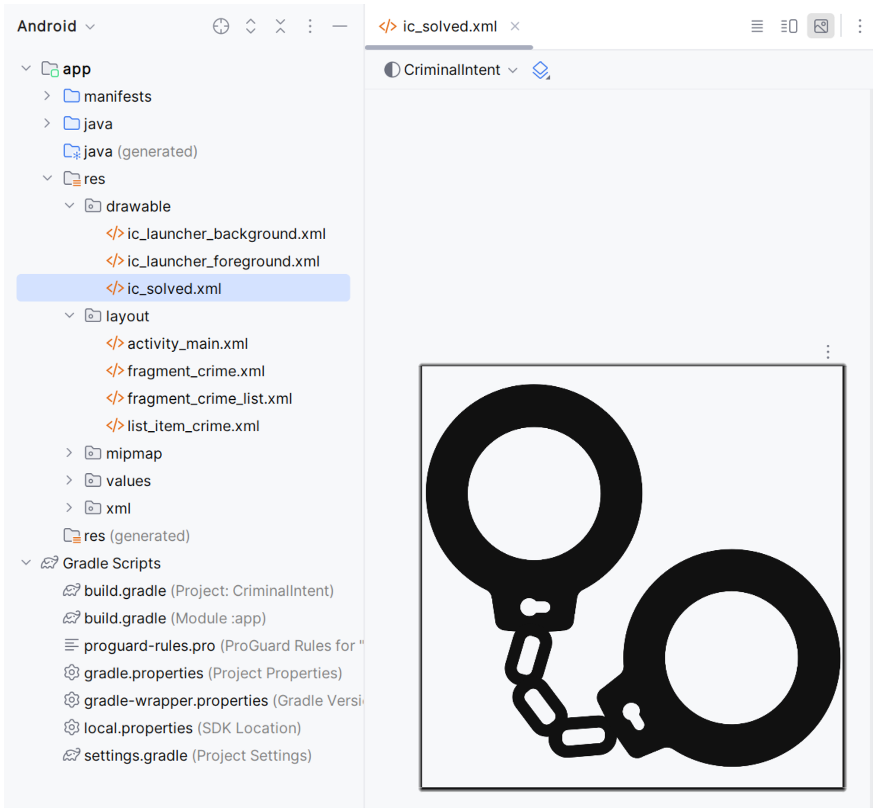
Student expected to be able to create UI with Layout & Widgets in Android apps using Android Studio IDE.

Example Practice Task:

Create UI with Layout & Widgets in Android apps using Android Studio.

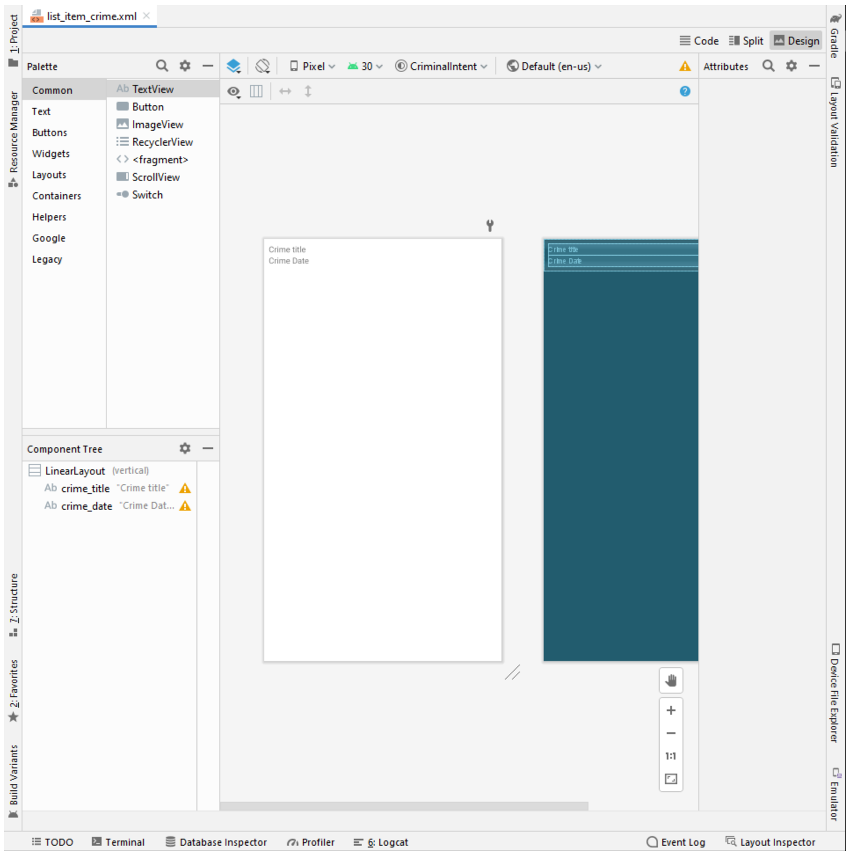

9:00
LTE
CriminalIntent
Crime #0
Tue Dec 18 16:43:18 EST 2018
Crime #1
Tue Dec 18 16:43:18 EST 2018
Crime #2
Tue Dec 18 16:43:18 EST 2018
Crime #3
Tue Dec 18 16:43:18 EST 2018
Crime #4
Tue Dec 18 16:43:18 EST 2018
Crime #5
Tue Dec 18 16:43:18 EST 2018
Crime #6
Tue Dec 18 16:43:18 EST 2018
Crime #7
Tue Dec 18 16:43:18 EST 2018
Crime #8
Tue Dec 18 16:43:18 EST 2018

1. learn more about layouts and widgets while adding some style to list items in the **RecyclerView**.
2. learn about a new tool called **ConstraintLayout**.
3. Copy each density version of **ic_solved.xml** into the appropriate drawable folder in your project.

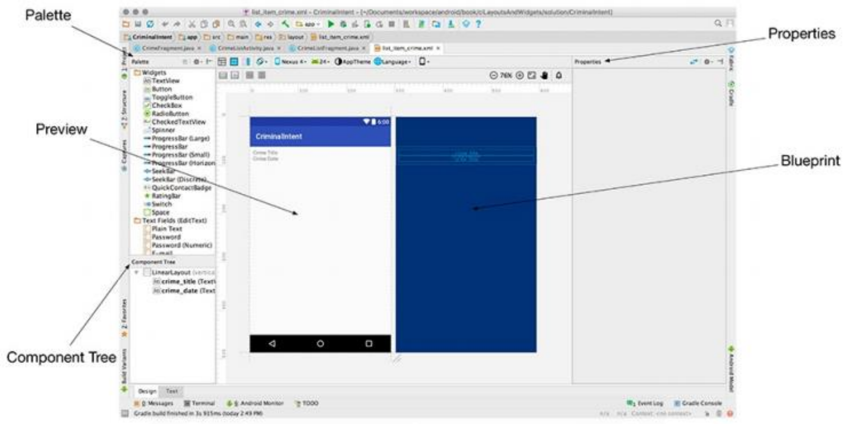

Android
ic_solved.xml
CriminalIntent
app
manifests
java
java (generated)
res
drawable
ic_launcher_background.xml
ic_launcher_foreground.xml
ic_solved.xml
layout
activity_main.xml
fragment_crime.xml
fragment_crime_list.xml
list_item_crime.xml
mipmap
values
xml
res (generated)
Gradle Scripts
build.gradle (Project: CriminalIntent)
build.gradle (Module :app)
proguard-rules.pro (ProGuard Rules for "
gradle.properties (Project Properties)
gradle-wrapper.properties (Gradle Versi
local.properties (SDK Location)
settings.gradle (Project Settings)

Using the Graphical Layout Tool

4. So far, you have created layouts by typing XML. In this section, you will use Android Studio's graphical layout tool.
5. Open list_item_crime.xml and select the Design tab at the upper right of the window.



6. In the middle of the graphical layout tool is the preview you have already seen. Just to the right of the preview is the *blueprint*.
7. The blueprint view is like the preview but shows an outline of each of your views.
8. This can be useful when you need to see how big each view is, not just what it is displaying.
9. On the lefthand side of the screen is the *palette*. This view contains all the widgets you could wish for, organized by category.

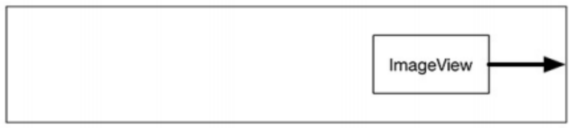


10. The *component tree* is in the bottom left. The tree shows how the widgets are organized in the layout.

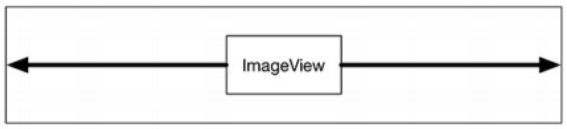
11. On the right side of the screen is the *properties view*. In this view, you can view and edit the attributes of the widget selected in the component tree.

Introducing ConstraintLayout

12. With **ConstraintLayout**, instead of using nested layouts you add a series of *constraints* to your layout.
13. A constraint is like a rubber band. It pulls two things toward each other.
14. 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), as shown.
15. The constraint will hold the **ImageView** to the right.



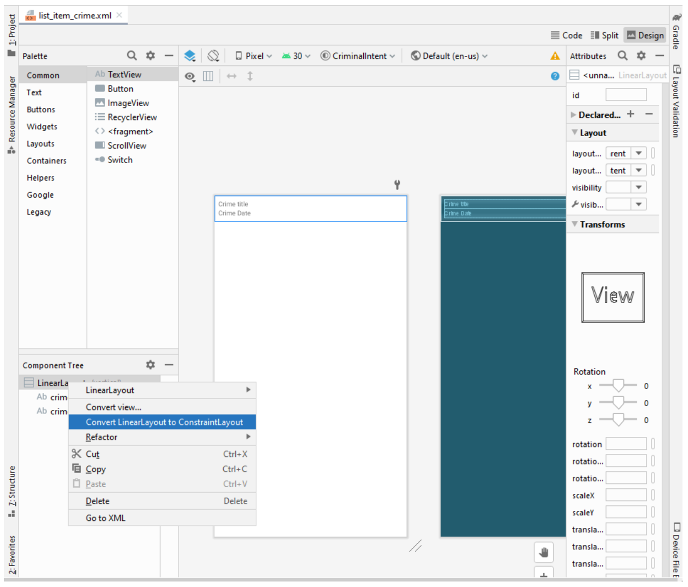
16. You can create a constraint from all four edges of your **ImageView** (left, top, right, and bottom). If you have opposing constraints, they will equal out and your **ImageView** will be right in the center of the two constraints



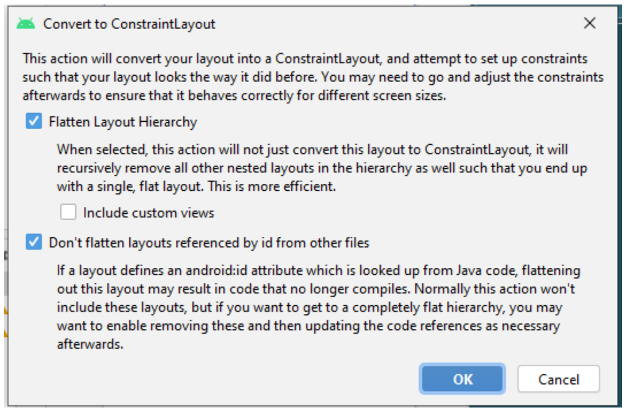
17. To place views where you want them to go in a **ConstraintLayout**, you give them constraints instead of dragging them around the screen.
18. What about sizing widgets? three options: Let the widget decide (wrap_content), decide for yourself, or let your widget expand to fit your constraints.

Using ConstraintLayout

19. convert list_item_crime.xml to use a **ConstraintLayout**. Right-click on your root **LinearLayout** in the component tree and select Convert LinearLayout to ConstraintLayout



20. Android Studio will ask you in a pop-up how aggressive you would like this conversion process to be.
21. Since list_item_crime is a simple layout file, there is not much that Android Studio can optimize. Leave the default values checked and select OK.



Convert to ConstraintLayout

This action will convert your layout into a ConstraintLayout, and attempt to set up constraints such that your layout looks the way it did before. You may need to go and adjust the constraints afterwards to ensure that it behaves correctly for different screen sizes.

[✓] Flatten Layout Hierarchy

When selected, this action will not just convert this layout to ConstraintLayout, it will recursively remove all other nested layouts in the hierarchy as well such that you end up with a single, flat layout. This is more efficient.

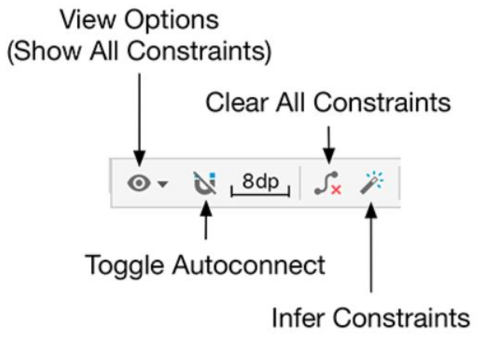
[ ] Include custom views

[✓] Don't flatten layouts referenced by id from other files

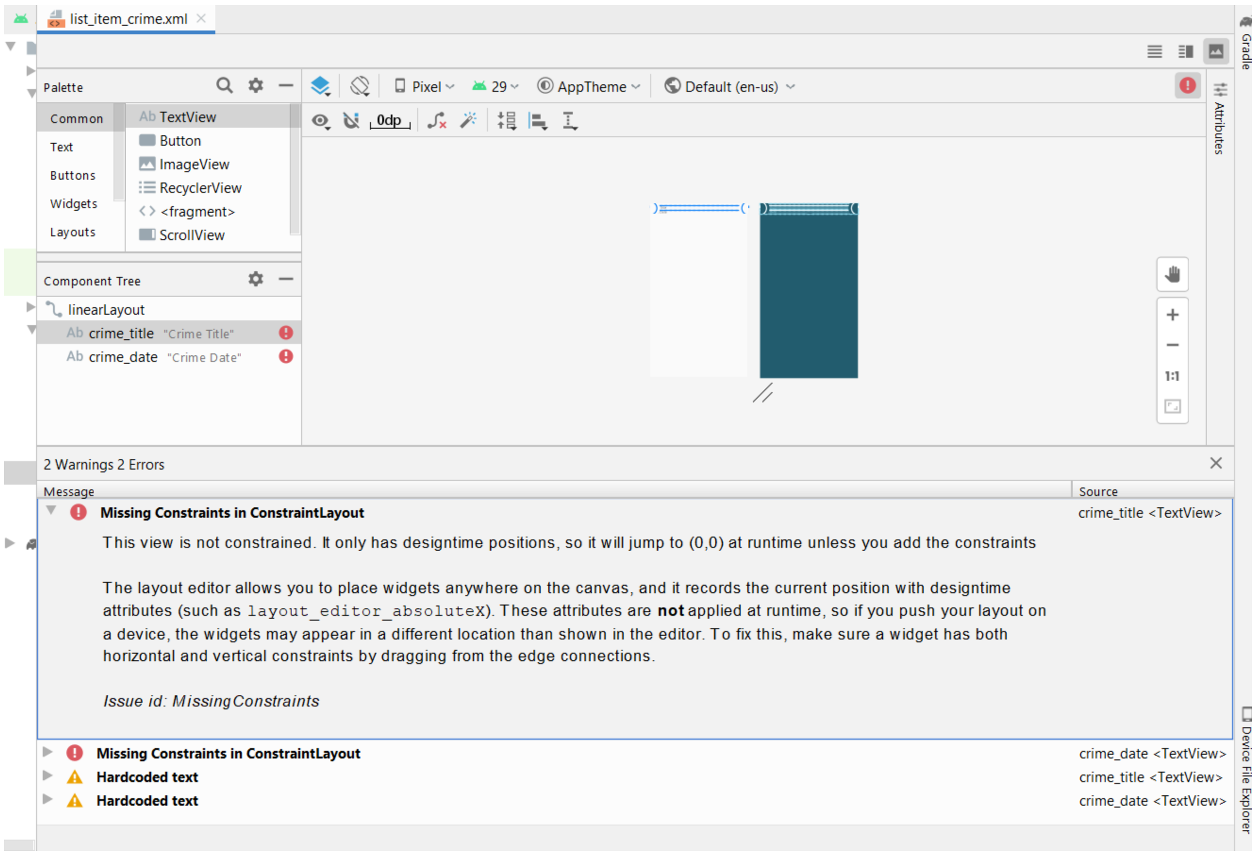
If a layout defines an android:id attribute which is looked up from Java code, flattening out this layout may result in code that no longer compiles. Normally this action won't include these layouts, but if you want to get to a completely flat hierarchy, you may want to enable removing these and then updating the code references as necessary afterwards.

The graphical editor

22. In **Component Tree** window, click **crime_title**. Look to the toolbar near the top of the preview and you will find a few editing controls



23. When you converted list_item_crime to use **ConstraintLayout**, Android Studio automatically added the constraints it thinks will replicate the behavior of your old layout.
24. However, to learn how constraints work you are going to start from scratch.
25. Select the ConstraintLayout view in the component tree, then choose the Clear All Constraints option.
26. You will immediately see red warning flags, including one at the top right of the screen. Click on it to see what that is all about.



2 Warnings 2 Errors

Message	Source
Missing Constraints in ConstraintLayout	crime_title <TextView>

This view is not constrained. It only has designtime positions, so it will jump to (0,0) at runtime unless you add the constraints

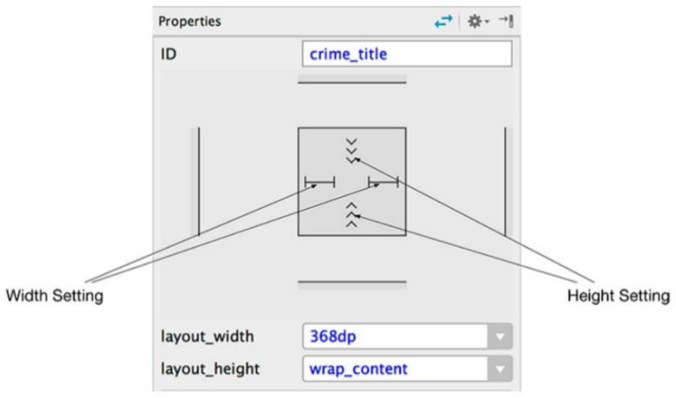
The layout editor allows you to place widgets anywhere on the canvas, and it records the current position with designtime attributes (such as `layout_editor_absoluteX`). These attributes are **not** applied at runtime, so if you push your layout on a device, the widgets may appear in a different location than shown in the editor. To fix this, make sure a widget has both horizontal and vertical constraints by dragging from the edge connections.

Issue id: MissingConstraints

Message	Source
Missing Constraints in ConstraintLayout	crime_date <TextView>
Hardcoded text	crime_title <TextView>
Hardcoded text	crime_date <TextView>

Making room

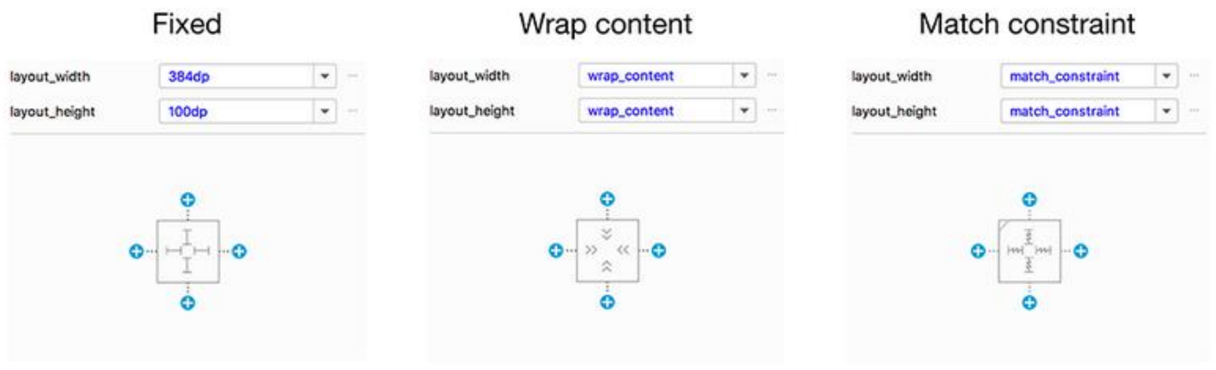
27. You need to make some room.
28. Your two **TextViews** are taking up the entire area, which will make it hard to wire up anything else.
29. Time to shrink those two widgets.
30. Select crime_title in the component tree and look at the properties view on the right



ID: crime_title
layout_width: 368dp
layout_height: wrap_content

31. The vertical and horizontal sizes of your **TextView** are governed by the height setting and width setting, respectively.

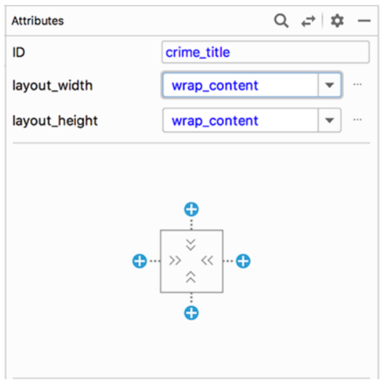
32. These can be set to one of three view size settings, each of which corresponds to a value for layout_width or layout_height.


Fixed
layout_width 384dp
layout_height 100dp

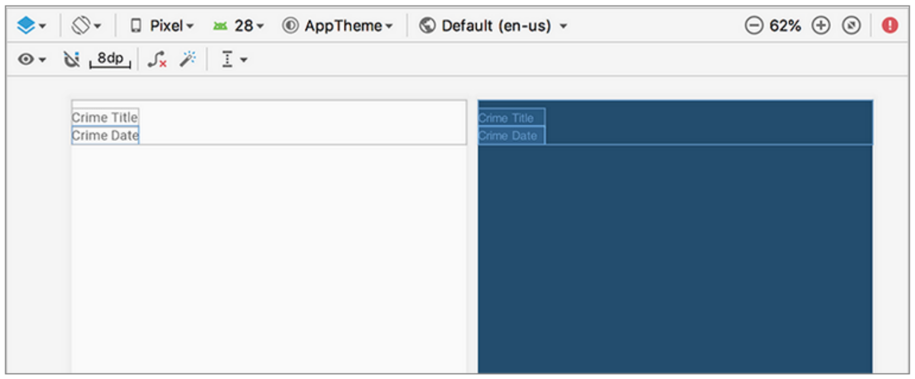
Wrap content
layout_width wrap_content
layout_height wrap_content

Match constraint
layout_width match_constraint
layout_height match_constraint

33. Both crime_title and crime_date are set to a large fixed width, which is why they are taking up the whole screen.
34. Adjust the width and height of both of these widgets. With crime_title still selected in the component tree, click the width setting until it cycles around to the wrap content setting. If necessary, adjust the height setting until the height is also set to wrap content

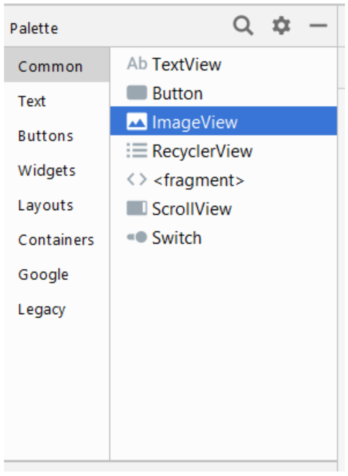

Attributes
ID crime_title
layout_width wrap_content
layout_height wrap_content

35. Repeat the process with the crime_date widget to set its width and height. Now, the two widgets overlap but are much smaller

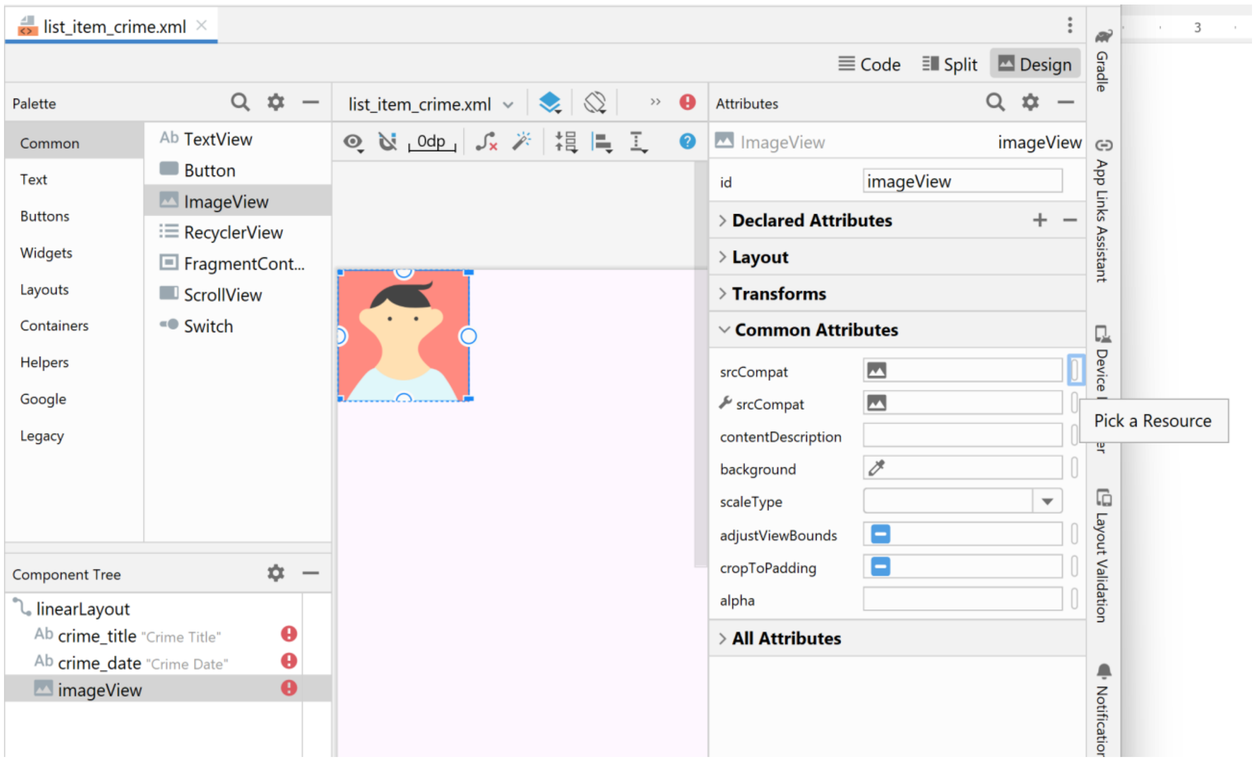


Adding widgets

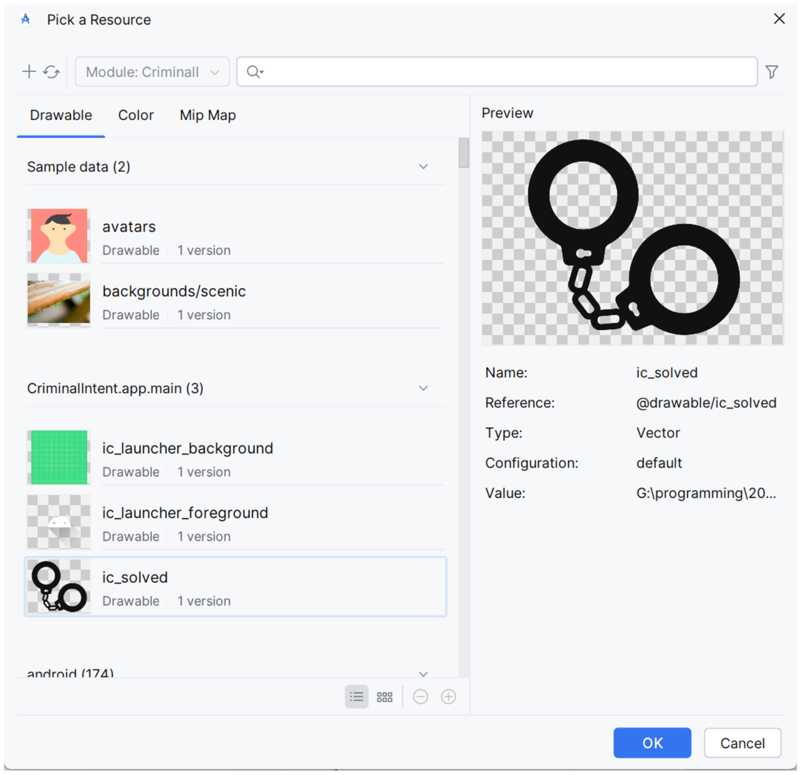
36. With your other widgets out of the way, you can add the handcuffs image to your layout. Add an **ImageView** to your layout file. In the palette, find **ImageView**.
37. Drag it into your component tree as a child of **ConstraintLayout**, just underneath crime_date.



38. on the Attributes window, click the right side of srcCompat and pick a resource



39. In the pop-up, choose ic_solved as the resource for the **ImageView**. This image will be used to indicate which crimes have been solved.



Pick a Resource

Module: Criminall

Drawable | Color | Mip Map

Sample data (2)

- avatars — Drawable | 1 version
- backgrounds/scenic — Drawable | 1 version

CriminalIntent.app.main (3)

- ic_launcher_background — Drawable | 1 version
- ic_launcher_foreground — Drawable | 1 version
- ic_solved — Drawable | 1 version

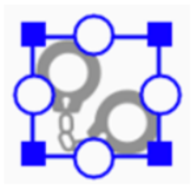
android (174)

Preview

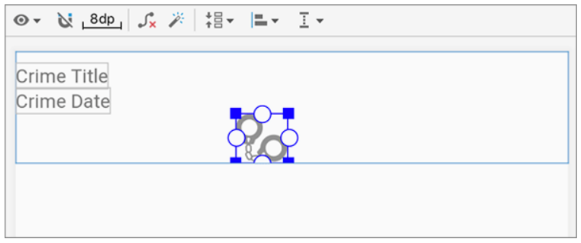
Name:	ic_solved
Reference:	@drawable/ic_solved
Type:	Vector
Configuration:	default
Value:	G:\programming\20...

OK | Cancel

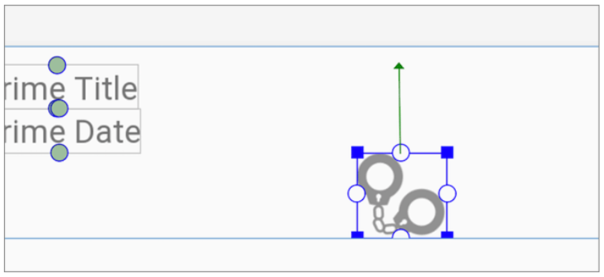
40. The **ImageView** is now a part of your layout, but it has no constraints. So while the graphical editor gives it a position, that position does not really mean anything.
41. Time to add some constraints. Click on your **ImageView** in the preview or in the component tree. You will see dots on each side of the **ImageView**. Each of these dots represents a *constraint handle*.



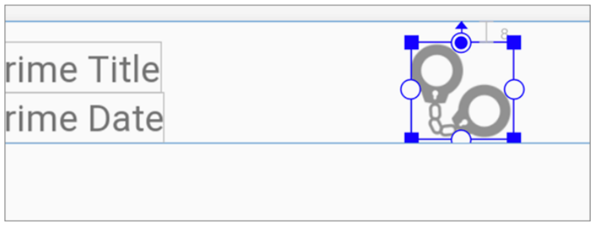
42. You want the **ImageView** to be anchored in the right side of the view. To accomplish this, you need to create constraints from the top, right, and bottom edges of the **ImageView**.
43. Before adding constraints, drag the `ImageView` to the right and down to move it away from the `TextViews`. Do not worry about where you place the `ImageView`. This placement will be ignored once you get your constraints in place.



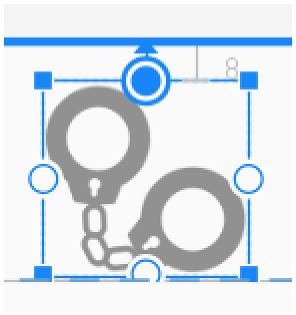
44. First, you are going to set a constraint between the top of the **ImageView** and the top of the **ConstraintLayout**.
45. In the preview, drag the top constraint handle from the `ImageView` to the top of the `ConstraintLayout`. The handle will display an arrow.



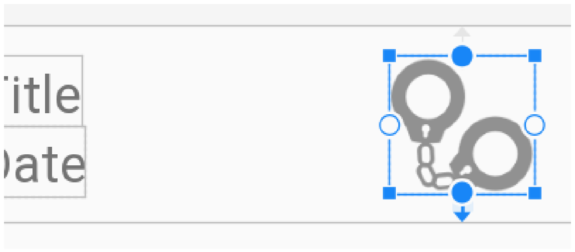
46. Keep dragging. Then release the mouse to create the constraint.



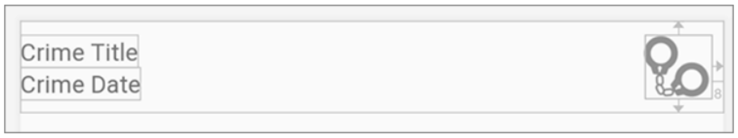
47. Be careful to avoid clicking when the mouse cursor is a corner shape – this will resize your **ImageView** instead. Also, make sure you do not inadvertently attach the constraint to one of your **TextViews**.
48. If you do, click on the constraint handle to delete the bad constraint, then try again.
49. When you let go and set the constraint, the view will snap into position to account for the presence of the new constraint.
50. This is how you move views around in a **ConstraintLayout** – by setting and removing constraints.
51. Verify that your **ImageView** has a top constraint connected to the top of the **ConstraintLayout** by hovering over the **ImageView** with your mouse.



52. Do the same for the bottom constraint handle, dragging it from the **ImageView** to the bottom of the root view, also taking care to avoid attaching to the **TextView**s. Again, you will need to drag the connection toward the center of the root view and then slightly down, as shown.

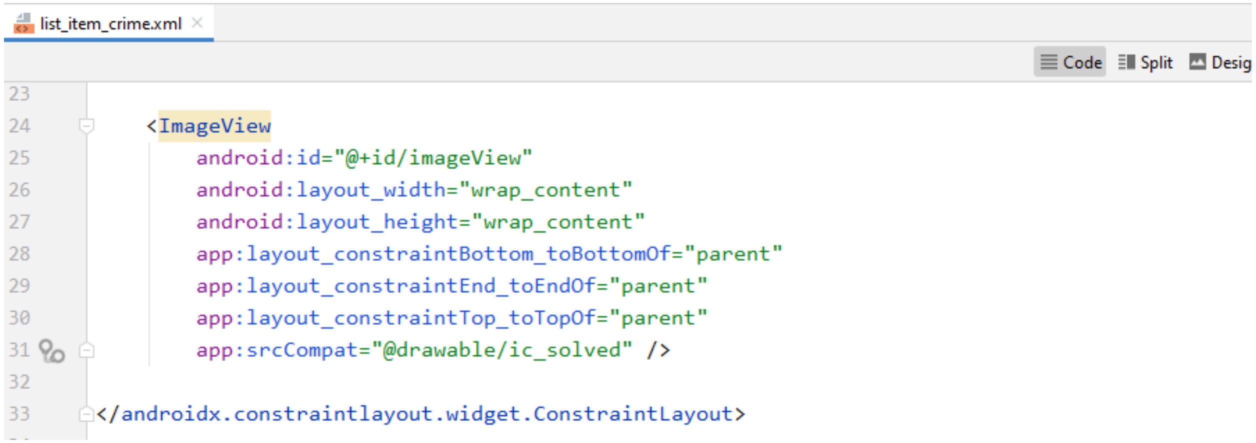


53. Finally, drag the right constraint handle from the **ImageView** to the right side of the root view. That should set all of your constraints. Hovering over the **ImageView** will show all of them. Your constraints should look like this.



ConstraintLayout's inner workings

54. Any edits that you make with the graphical editor are reflected in the XML behind the scenes. You can still edit the raw **ConstraintLayout** XML, but the graphical editor will often be easier, because **ConstraintLayout** is much more verbose than other **ViewGroup**s.
55. Switch to the text view to see what happened to the XML when you created the three constraints on your **ImageView**.



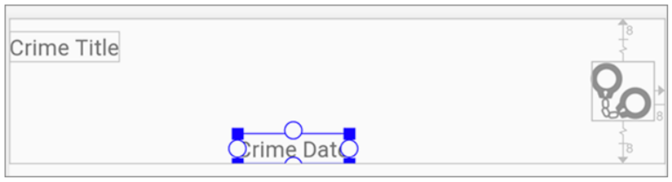
```xml

    <ImageView
        android:id="@+id/imageView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:srcCompat="@drawable/ic_solved" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

56. Take a closer look at the top constraint: app:layout_constraintTop_toTopOf="parent“
57. This attribute begins with layout_. All attributes that begin with layout_ are known as *layout parameters*.
58. Unlike other attributes, layout parameters are directions to that widget's *parent*, not the widget itself.
59. They tell the parent layout how to arrange the child element within itself.

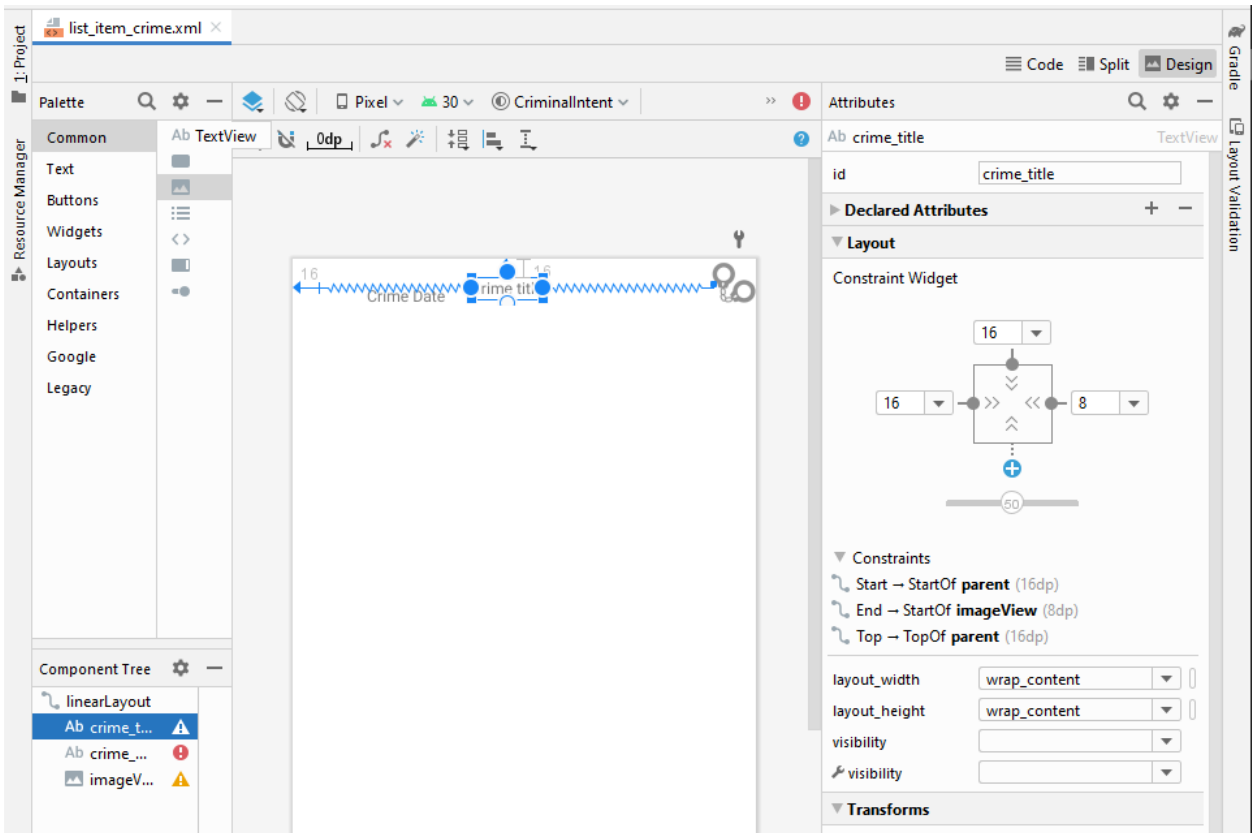
60. You have seen a few layout parameters so far, like layout_width and layout_height.
61. The name of the constraint is constraintTop. This means that this is the top constraint on your **ImageView**.
62. Finally, the attribute ends with toTopOf="parent". This means that this constraint is connected to the top edge of the parent. The parent here is the **ConstraintLayout**.

Editing properties

63. Your **ImageView** is now positioned correctly.
64. Next up: Position and size the title **TextView**.
65. First, select crime_date in the component tree and drag it out of the way.
66. Remember that any changes you make to the position in the preview will not be represented when the app is running.
67. At runtime, only constraints remain.



68. Now, select crime_title in the component tree. This will also highlight crime_title in the preview.
69. You want crime_title to be at the top left of your layout, positioned to the left of your new **ImageView**.
70. That requires three constraints:
    - from the left side of your view to the left side of the parent, with a 16dp margin
    - from the top of your view to the top of the parent, with a 16dp margin
    - from the right of your view to the left side of the new **ImageView**, with an 8dp margin
71. Modify your layout so that all of these constraints are in place. (finding the right place to click can be tricky. Try to click inside of the **TextView**, and remember that you can always press (Ctrl+Z) to undo and try again.)
72. Verify that your constraints look like here. (The selected widget will show squiggly lines for any of its constraints that are stretching.)



crime_title TextView

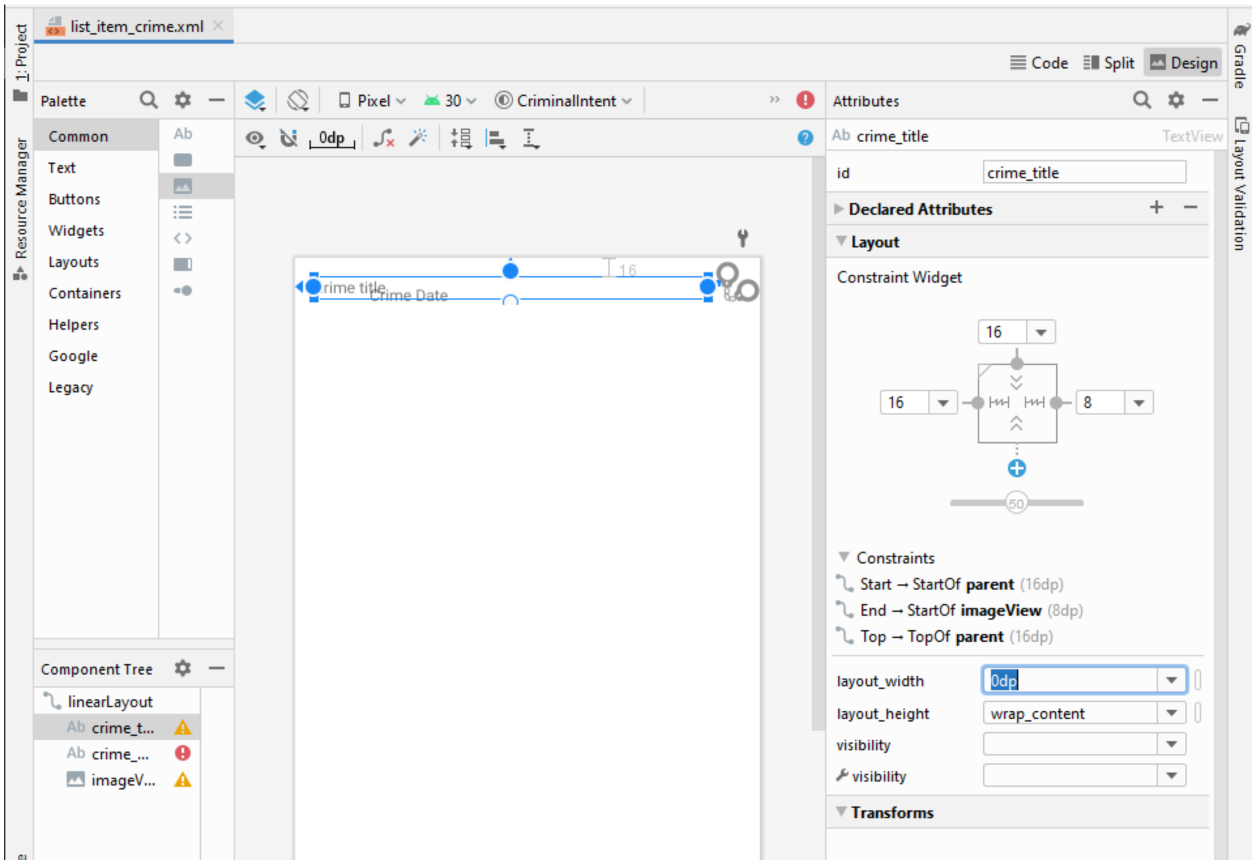
id: crime_title

Constraint Widget: 16, 16, 8

Constraints
- Start → StartOf **parent** (16dp)
- End → StartOf **imageView** (8dp)
- Top → TopOf **parent** (16dp)

layout_width	wrap_content
layout_height	wrap_content
visibility	
visibility	

73. Now that the constraints are set up, you can restore the title **TextView** to its full glory.
74. Adjust its horizontal view setting to any size (0dp) to allow the title **TextView** to fill all of the space available within its constraints.
75. Adjust the vertical view size to wrap_content, if it is not already, so that the **TextView** will be just tall enough to show the title of the crime.
76. Since you added constraints to the top, left, and right of the `TextView`, dropdown menus appear to allow you to select the margin for each constraint. Select `16dp` for the left and top margins, and select `8dp` for the right margin.



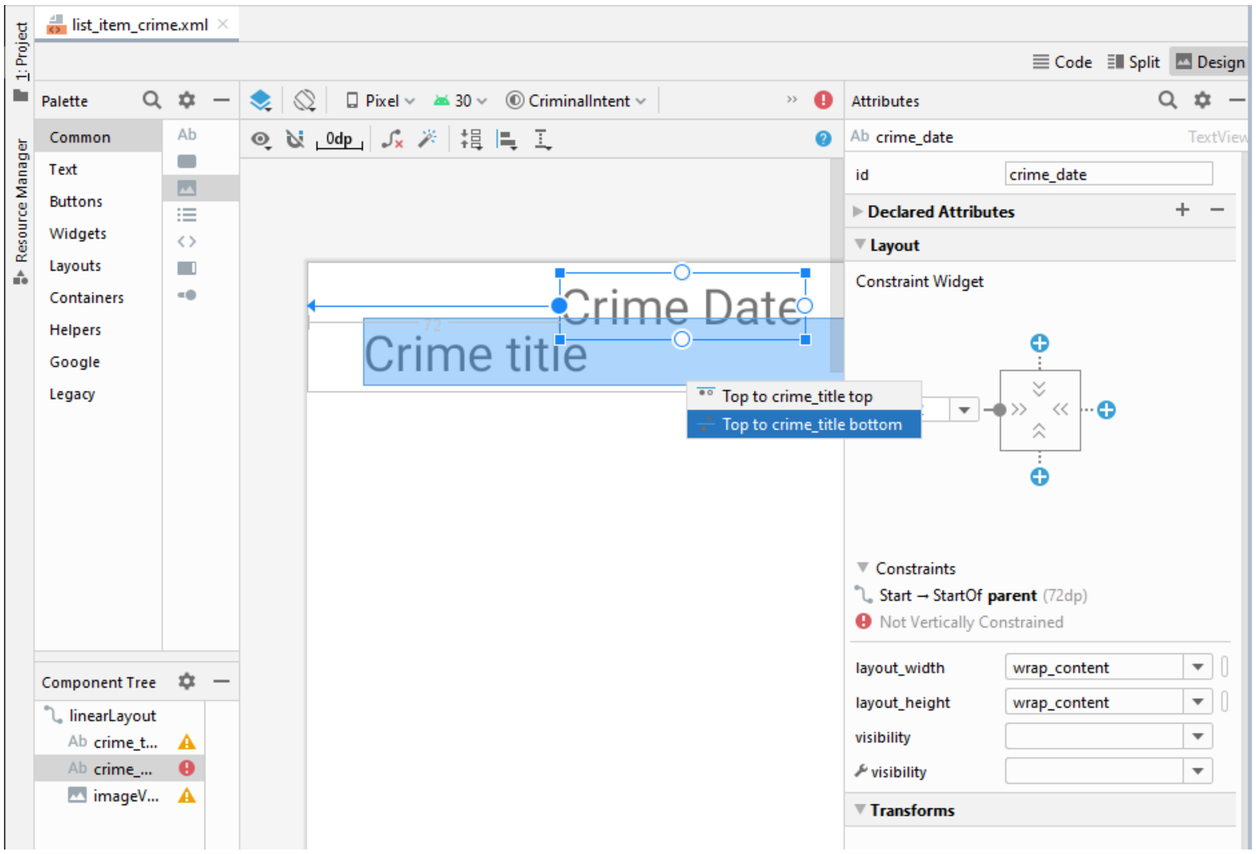
crime_title TextView

id crime_title

Constraints
Start → StartOf **parent** (16dp)
End → StartOf **imageView** (8dp)
Top → TopOf **parent** (16dp)

layout_width 0dp
layout_height wrap_content
visibility
visibility

77. Now, add constraints to the date **TextView**.
78. Select crime_date in the component tree.
79. You are going to add three constraints:
    - from the left side of your view to the left side of the parent, with a 16dp margin
    - from the top of your view to the bottom of the crime title, with an 8dp margin
    - from the right of your view to the left side of the new **ImageView**, with an 8dp margin

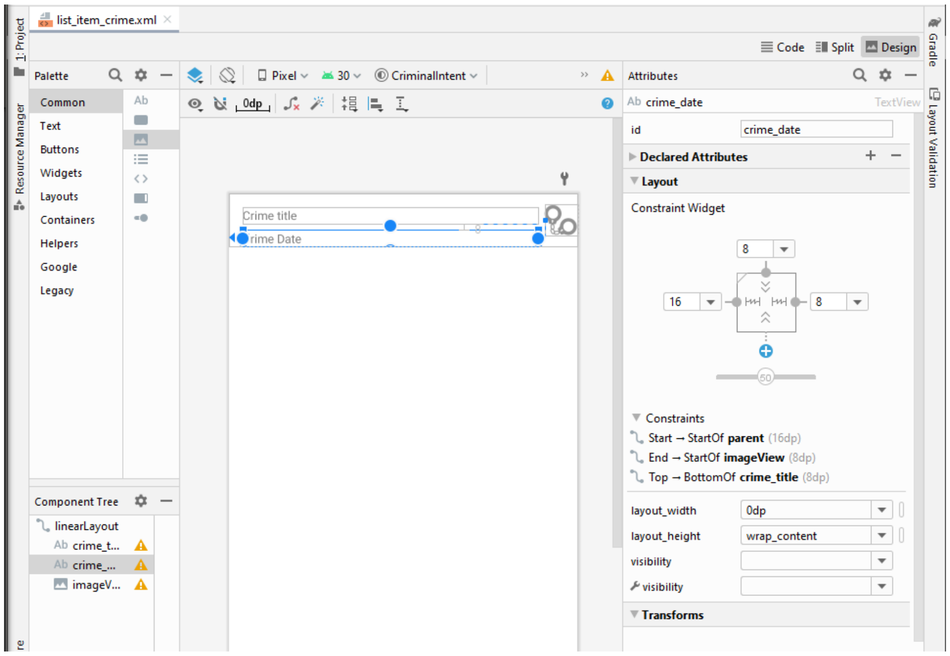


id: crime_date

Constraints
Start → StartOf **parent** (72dp)
Not Vertically Constrained

layout_width	wrap_content
layout_height	wrap_content
visibility	
visibility	

80. After adding the constraints, adjust the properties of the **TextView**. You want the width of your date **TextView** to be 0dp(match_constraint) and the height to be Wrap Content, just like the title **TextView**.
81. Verify that your settings match those shown

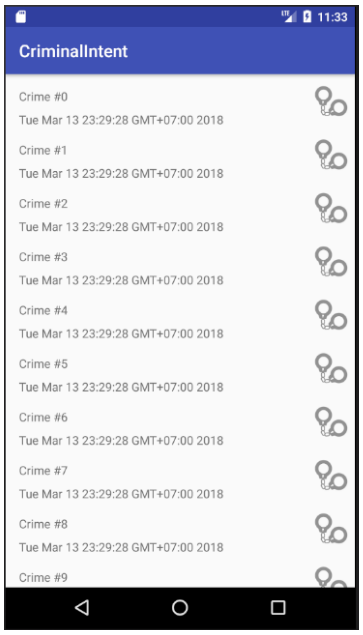


id: crime_date

Constraints
Start → StartOf **parent** (16dp)
End → StartOf **imageView** (8dp)
Top → BottomOf **crime_title** (8dp)

layout_width	0dp
layout_height	wrap_content
visibility	
visibility	

82. Run CriminalIntent and verify that you see all three components lined up nicely in each row of your **RecyclerView**


CriminalIntent

Crime #0
Tue Mar 13 23:29:28 GMT+07:00 2018

Crime #1
Tue Mar 13 23:29:28 GMT+07:00 2018

Crime #2
Tue Mar 13 23:29:28 GMT+07:00 2018

Crime #3
Tue Mar 13 23:29:28 GMT+07:00 2018

Crime #4
Tue Mar 13 23:29:28 GMT+07:00 2018

Crime #5
Tue Mar 13 23:29:28 GMT+07:00 2018

Crime #6
Tue Mar 13 23:29:28 GMT+07:00 2018

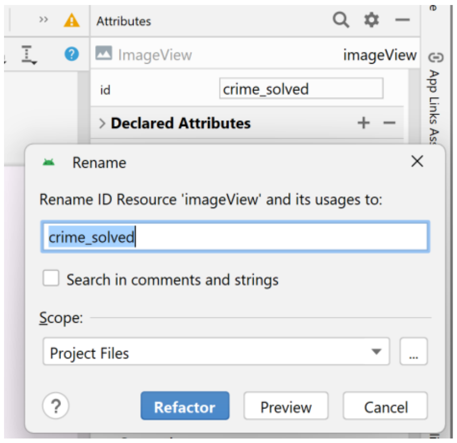
Crime #7
Tue Mar 13 23:29:28 GMT+07:00 2018

Crime #8
Tue Mar 13 23:29:28 GMT+07:00 2018

Crime #9

Making list items dynamic

83. Now that the layout includes the right constraints, update the **ImageView** so that the handcuffs are only shown on crimes that have been solved.
84. First, update the ID of your **ImageView**.
85. When you added the **ImageView** to your **ConstraintLayout**, it was given a default name. That name is not too descriptive.
86. Select your **ImageView** in list_item_crime.xml and, in the properties view, update the ID attribute to crime_solved.
87. You will be asked whether Android Studio should update all usages of the ID; select refactor.


Attributes

ImageView imageView

id crime_solved

Declared Attributes

Rename

Rename ID Resource 'imageView' and its usages to:

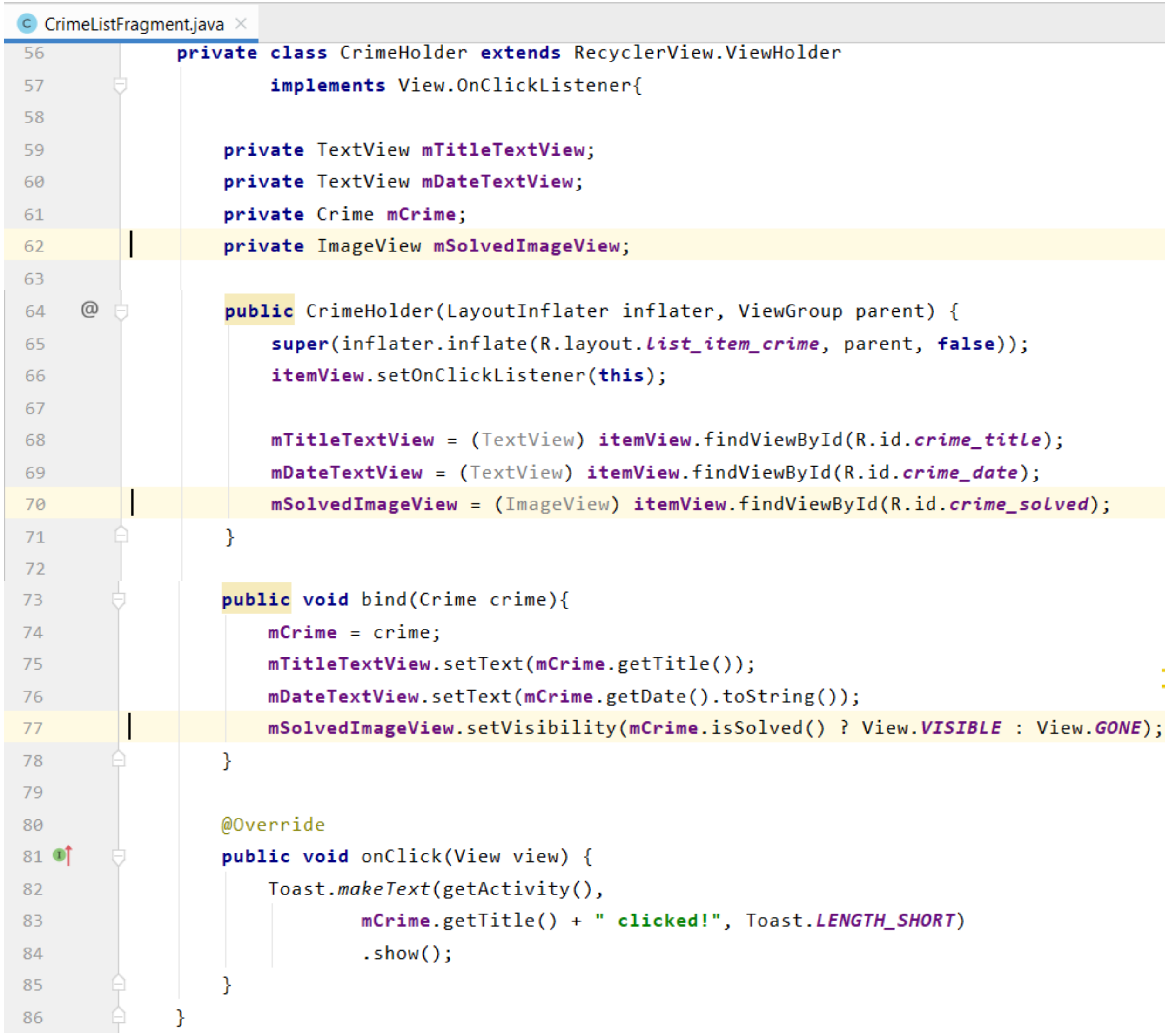
crime_solved

Search in comments and strings

Scope:

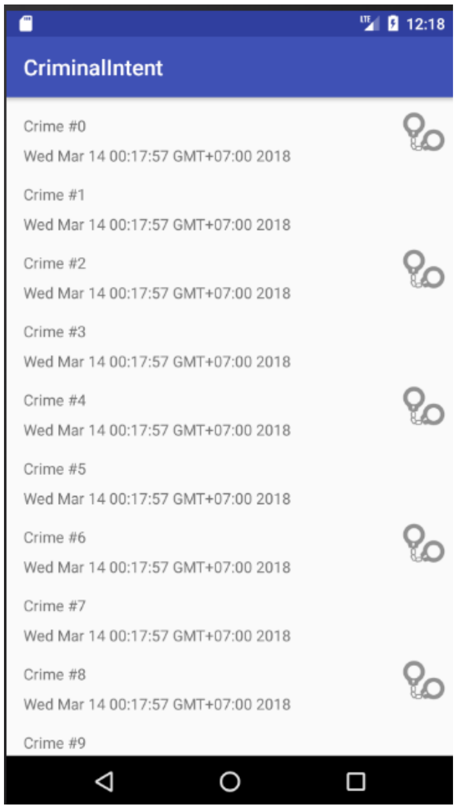
Project Files

88. You may notice that you are using the same view IDs in different layouts. The crime_solved ID is used in both the list_item_crime.xml and fragment_crime.xml layouts. You may think reusing IDs would be an issue, but in this case it is not a problem. Layout IDs only need to be unique in the same layout. Since your IDs are defined in different layout files, there is no problem using the same ID in both.
89. With a proper ID in place, now you will update your code.
90. Open CrimeListFragment.java.
91. In **CrimeHolder**, add an **ImageView** instance variable and toggle its visibility based on the solved status of your crime.


CrimeListFragment.java

```java
    private class CrimeHolder extends RecyclerView.ViewHolder
            implements View.OnClickListener{

        private TextView mTitleTextView;
        private TextView mDateTextView;
        private Crime mCrime;
        private ImageView mSolvedImageView;

        public CrimeHolder(LayoutInflater inflater, ViewGroup parent) {
            super(inflater.inflate(R.layout.list_item_crime, parent, false));
            itemView.setOnClickListener(this);

            mTitleTextView = (TextView) itemView.findViewById(R.id.crime_title);
            mDateTextView = (TextView) itemView.findViewById(R.id.crime_date);
            mSolvedImageView = (ImageView) itemView.findViewById(R.id.crime_solved);
        }

        public void bind(Crime crime){
            mCrime = crime;
            mTitleTextView.setText(mCrime.getTitle());
            mDateTextView.setText(mCrime.getDate().toString());
            mSolvedImageView.setVisibility(mCrime.isSolved() ? View.VISIBLE : View.GONE);
        }

        @Override
        public void onClick(View view) {
            Toast.makeText(getActivity(),
                    mCrime.getTitle() + " clicked!", Toast.LENGTH_SHORT)
                    .show();
        }
    }
```

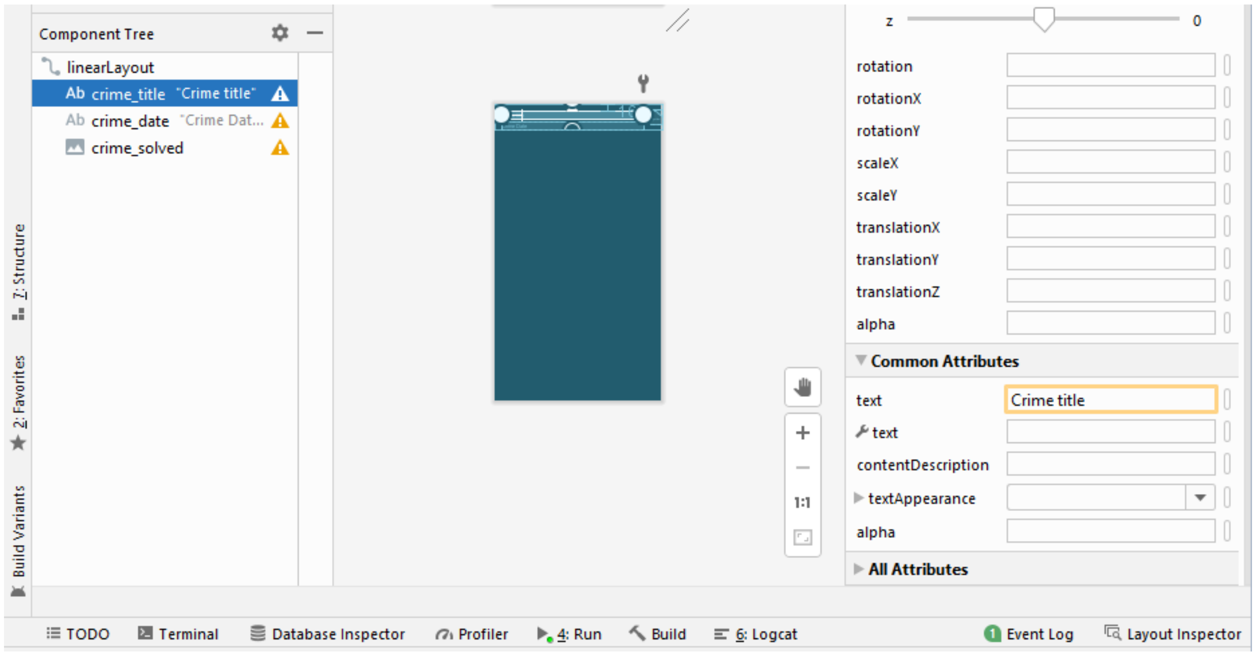
92. Run CriminalIntent and verify that the handcuffs now appear on every other row.



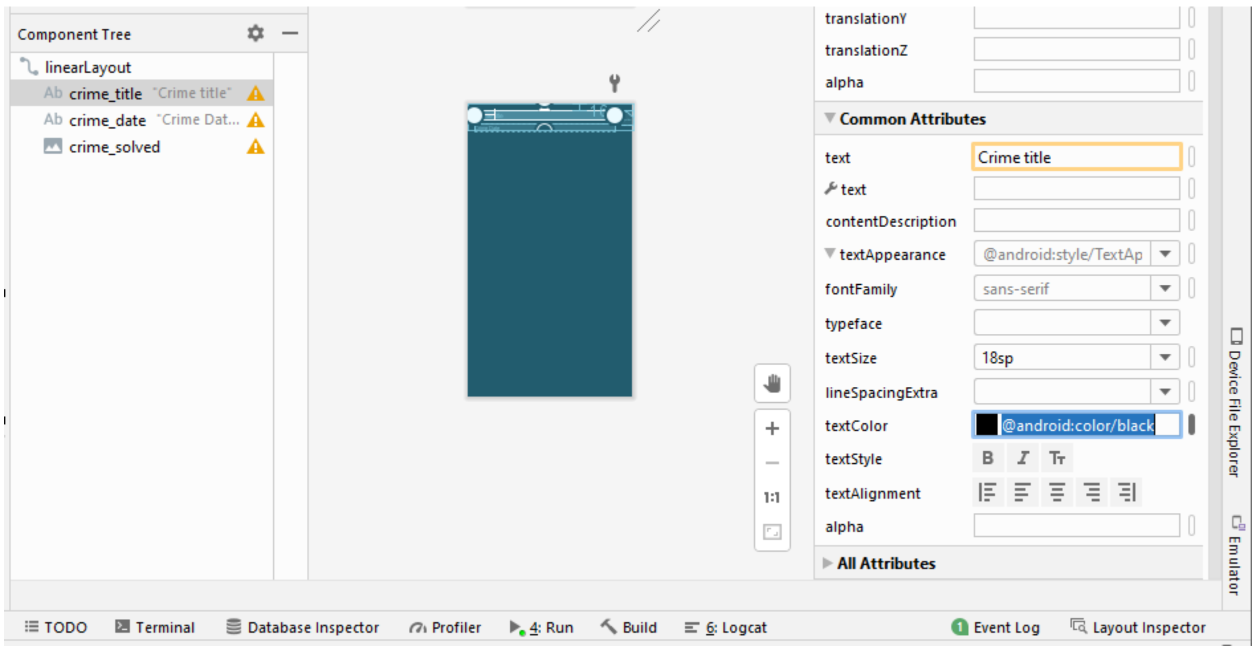
```
CriminalIntent
Crime #0
Wed Mar 14 00:17:57 GMT+07:00 2018
Crime #1
Wed Mar 14 00:17:57 GMT+07:00 2018
Crime #2
Wed Mar 14 00:17:57 GMT+07:00 2018
Crime #3
Wed Mar 14 00:17:57 GMT+07:00 2018
Crime #4
Wed Mar 14 00:17:57 GMT+07:00 2018
Crime #5
Wed Mar 14 00:17:57 GMT+07:00 2018
Crime #6
Wed Mar 14 00:17:57 GMT+07:00 2018
Crime #7
Wed Mar 14 00:17:57 GMT+07:00 2018
Crime #8
Wed Mar 14 00:17:57 GMT+07:00 2018
Crime #9
```

More on Layout Attributes

93. Navigate back to the Design view of list_item_crime.xml.
94. Select crime_title and adjust some of the attributes in the properties view.
95. Click the disclosure arrow next to textAppearance to reveal a set of text and font attributes.

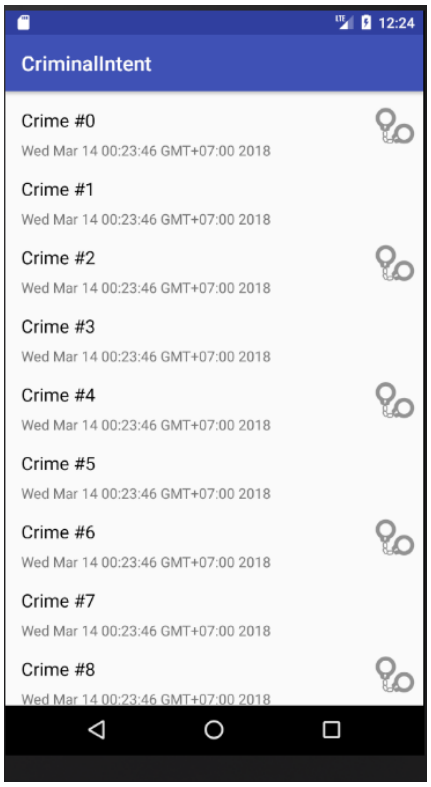


96. Update the textColor attribute to @android:color/black
97. Next, set the textSize attribute to 18sp.



translationY
translationZ
alpha
text: Crime title
text
contentDescription
textAppearance: @android:style/TextAp
fontFamily: sans-serif
typeface
textSize: 18sp
lineSpacingExtra
textColor: @android:color/black
textStyle
textAlignment
alpha

98. Run CriminalIntent and see the new look.



Crime #0
Wed Mar 14 00:23:46 GMT+07:00 2018
Crime #1
Wed Mar 14 00:23:46 GMT+07:00 2018
Crime #2
Wed Mar 14 00:23:46 GMT+07:00 2018
Crime #3
Wed Mar 14 00:23:46 GMT+07:00 2018
Crime #4
Wed Mar 14 00:23:46 GMT+07:00 2018
Crime #5
Wed Mar 14 00:23:46 GMT+07:00 2018
Crime #6
Wed Mar 14 00:23:46 GMT+07:00 2018
Crime #7
Wed Mar 14 00:23:46 GMT+07:00 2018
Crime #8
Wed Mar 14 00:23:46 GMT+07:00 2018

Task : Formatting the Date

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 your button. While timestamps make for good documentation, it might be nicer if the button just displayed the date as humans think of it – like "Mar 22, 2021." 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.

Crreate a format string that will display the day of the week as well – for example, "Monday, Mar 22, 2021."

Notes:

1. Create folder PRG6_M4_P3_XXXXXXXXXX.
2. Zip the folder and submit it to the server.

Bibliography:

- Marsicano, et. al., "Android Programming – The Big Nerd Ranch", 5th Ed, 2022, Pearson Technology.