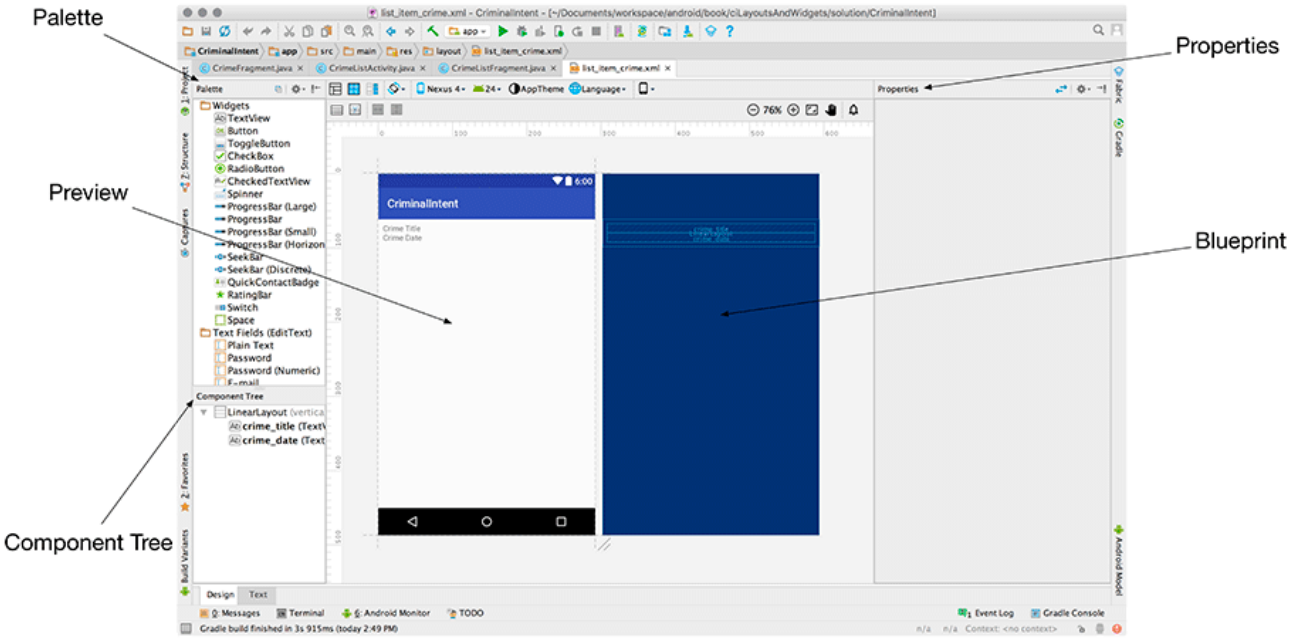
Layouts, Widgets and Fragment Arguments-11

In this assignment, you will learn more about layouts and widgets while adding some style to your list items in the RecyclerView. You will also learn about a new tool called ConstraintLayout and how to use fragment arguments to pass data around.

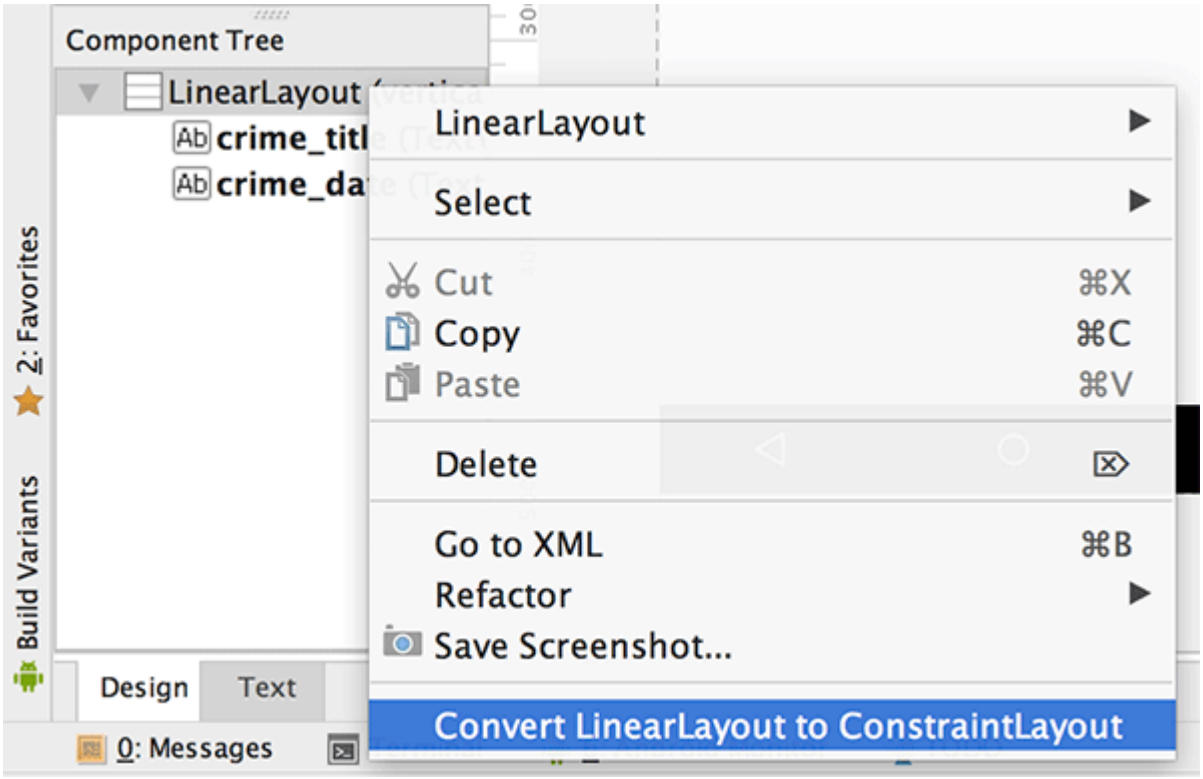
1. Open your CriminalIntent app in Android Studio. Ensure you have committed your previous lab and submitted the commit ID before moving forward.

2. So far, you have created layouts by typing XML. In this assignment, you will use Android Studio's graphical layout tool. Open list_item_crime.xml and select the Design tab at the bottom of the file.

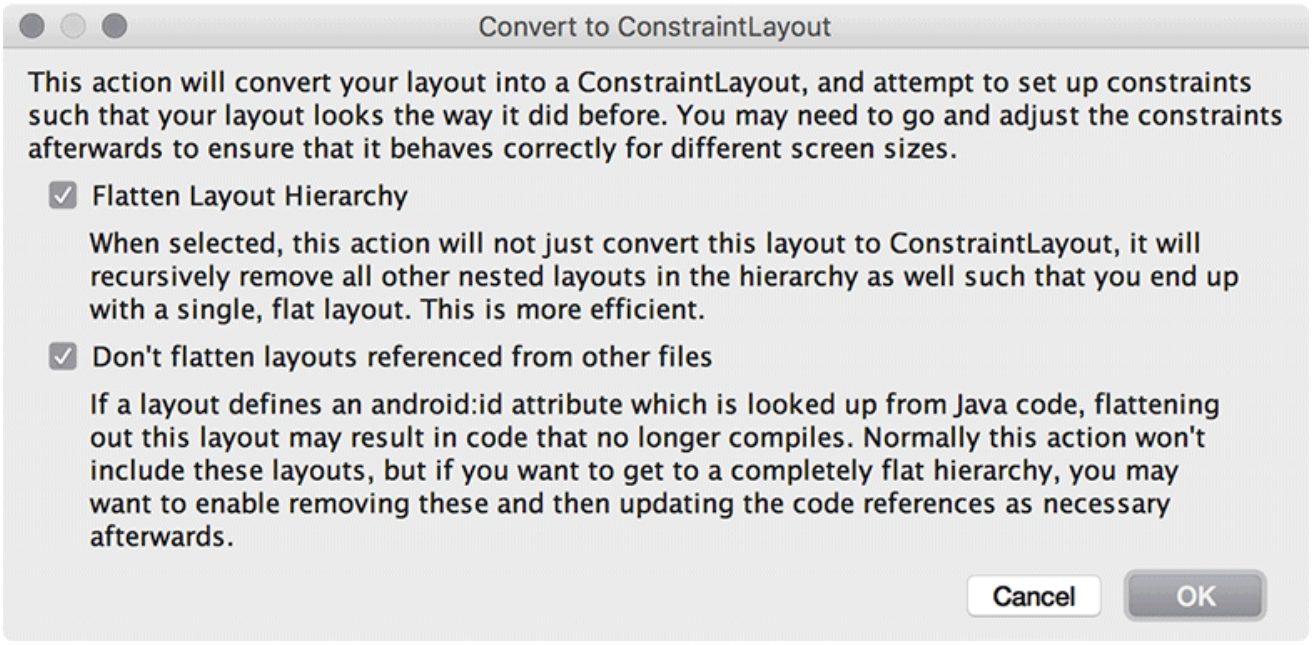
3. In the middle of the graphical layout tool is the preview you have already seen. Just to the right of the preview is the blueprint. The blueprint view is like the preview but shows an outline of each of your views. This can be useful when you need to see how big each view is, not just what it is displaying. On the left-hand side of the screen is the palette. This view contains all the widgets you could wish for, organized by category. The component tree is in the bottom left. The tree shows how the widgets are organized in the layout. 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.



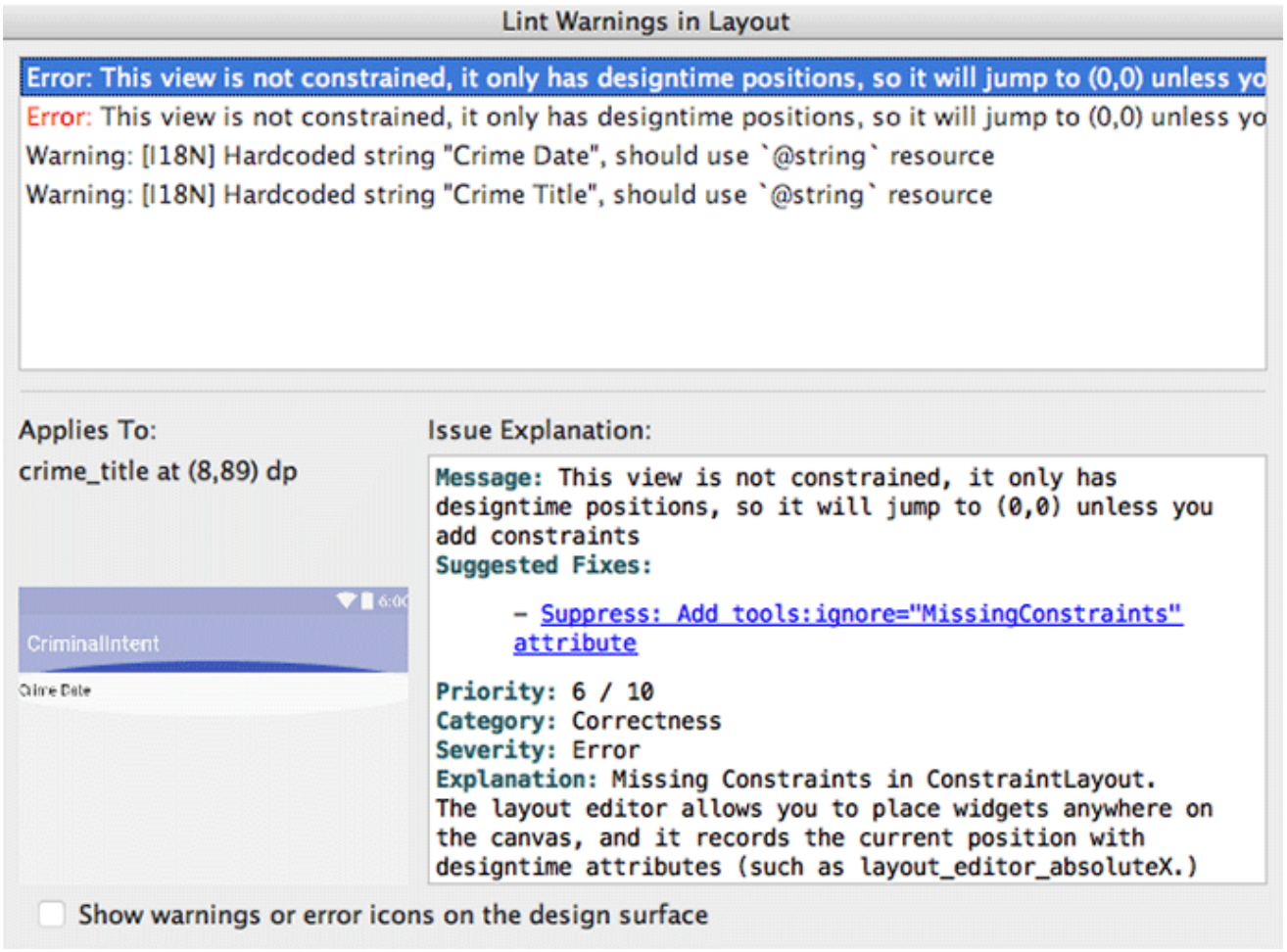
4. Convert list_item_crime.xml to use a ConstraintLayout. Right-click on your root LinearLayout in the component tree and select Convert LinearLayout to ConstraintLayout.



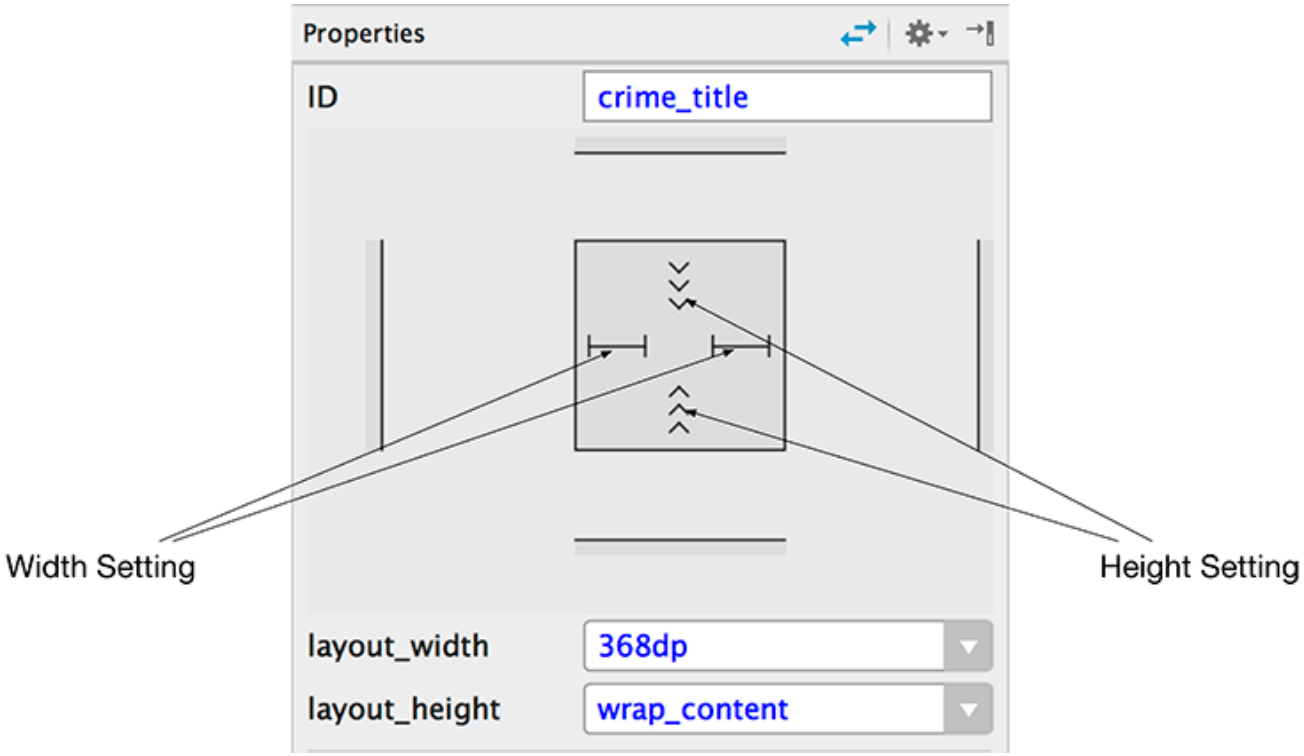
5. Android Studio will ask you in a pop-up how aggressive you would like this conversion process to be. Since list_item_crime is a simple layout file, there is not much that Android Studio can optimize. Leave the default values checked and select OK.



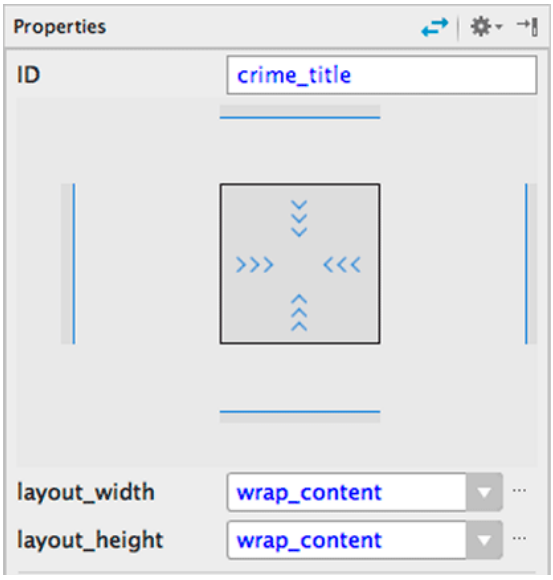
6. Select the ConstraintLayout view in the component tree, then choose the Clear All Constraints option. You will immediately see a red warning flag with the number 4 at the top right of the screen. Click on it to see what that is all about.



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

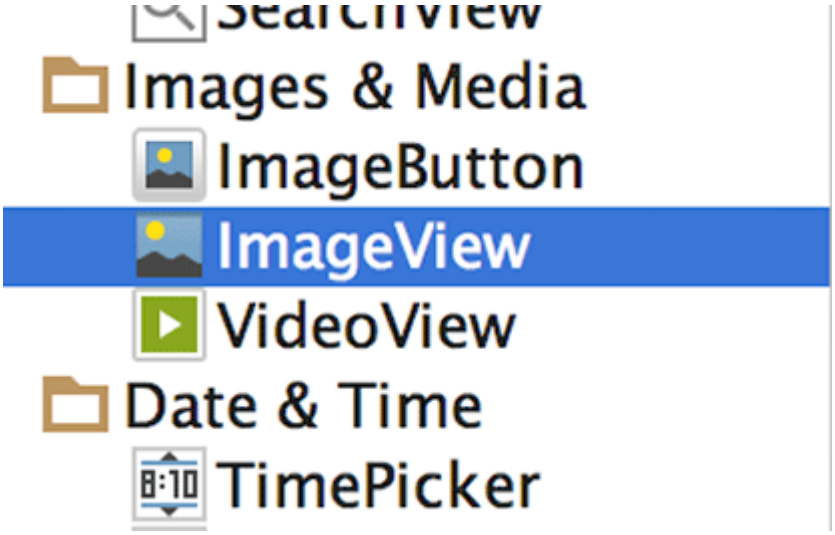


8. Both crime_title and crime_date are set to a large fixed width, which is why they are taking up the whole screen. Adjust the width and height of both of these widgets. With crime_title still selected in the component tree, click the width setting until it cycles around to the wrap content setting.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

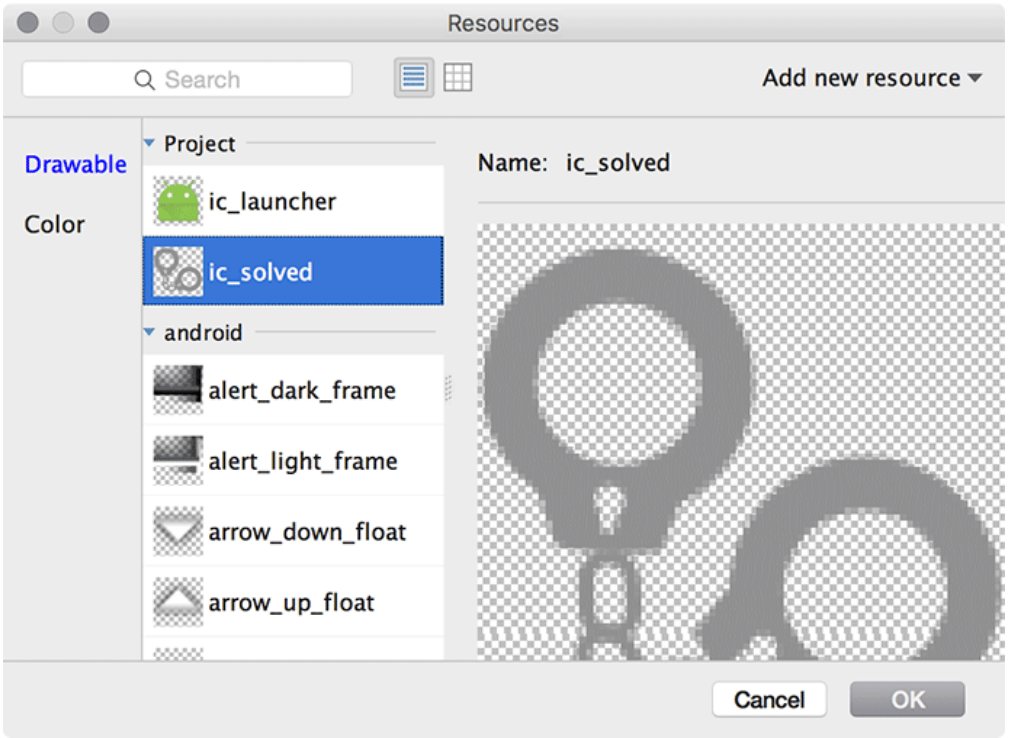


9. Repeat the process with the crime_date widget to set its width and height. Now, the two widgets overlap but are much smaller.

10. With your other widgets out of the way, you can now add a handcuff image (download the images from the LMS and import them into your project). Add an ImageView to your layout file. In the palette, find ImageView. Drag it into your component tree as a child of ConstraintLayout, just underneath crime_date.

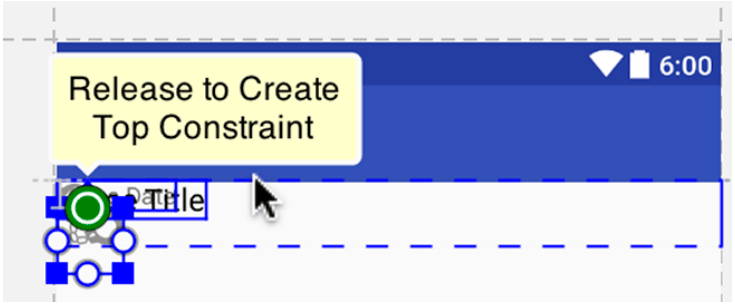


11. In the pop-up, choose ic_solved (if you do not see it, ensure you added the image from the LMS to your project) as the resource for the ImageView. This image will be used to indicate which crimes have been solved.

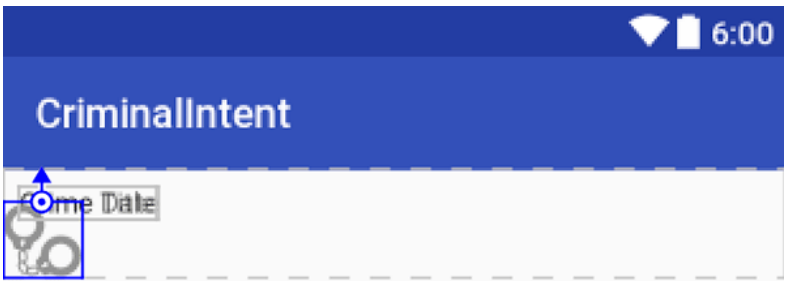


12. The ImageView is now a part of your layout, but it has no constraints. So while the graphical editor gives it a position, that position does not really mean anything. Time to add some constraints. Click on your ImageView in the preview or in the component tree. You will see dots on each side of the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

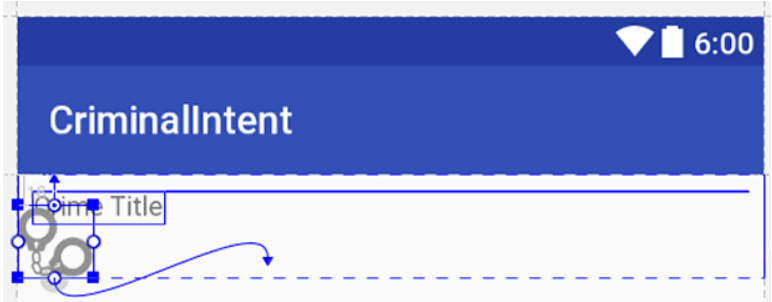
13. You want the ImageView to be anchored in the right side of the view. To accomplish this, you need to create constraints from the top, right, and bottom edges of the ImageView. First, you are going to set a constraint between the top of the ImageView and the top of the ConstraintLayout. The top of the ConstraintLayout is a little difficult to see, but it is just under the blue CriminalIntent toolbar. In the preview, drag the top constraint handle from the ImageView to the top of the ConstraintLayout – you will need to drag to the right somewhat, because the image is at the top of the constraint layout. Watch for the constraint handle to turn green and a pop-up reading Release to Create Top Constraint to appear, and release the mouse.



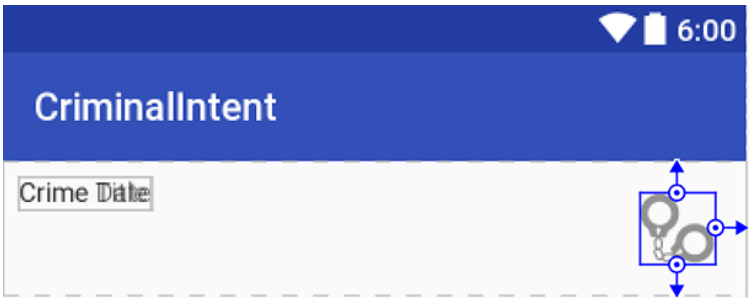
14. Be careful to avoid clicking when the mouse cursor is a corner shape – this will resize your ImageView instead. Also, make sure you do not inadvertently attach the constraint to one of your TextViews. If you do, click on the constraint handle to delete the bad constraint, then try again. When you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



15. 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 TextViews. Again, you will need to drag the connection toward the center of the root view and then slightly down.

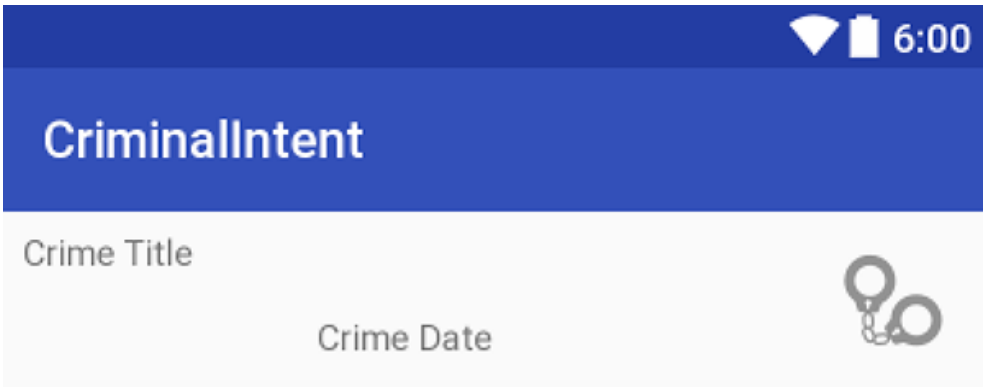


16. Drag the right constraint handle from the ImageView to the right side of the root view. That should set all of your constraints. Hovering over the ImageView will show all of them. Your constraints should look like:



17. 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 ViewGroups. Switch to the text view to see what happened to the XML when you created the three constraints on your ImageView.

18. Select crime_date in the component tree and drag it out of the way. Remember that any changes you make to the position in the preview will not be represented when the app is running. At runtime, only constraints remain.

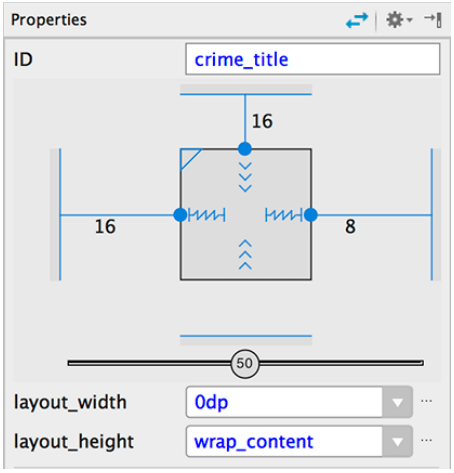


19. Now, select crime_title in the component tree. This will also highlight crime_title in the preview. You want crime_title to be at the top left of your layout, positioned to the left of your new ImageView. That requires three constraints:
    a) from the left side of your view to the left side of the parent, with a 16dp margin
    b) from the top of your view to the top of the parent, with a 16dp margin
    c) from the right of your view to the left side of the new ImageView, with an 8dp margin

20. Modify your layout so that all of these constraints are in place. (As of this writing, finding the right place to click can be tricky. Try to click inside of the TextView, and remember that you can always key Ctrl+Z to undo and try again. Verify that your constraints look like this:

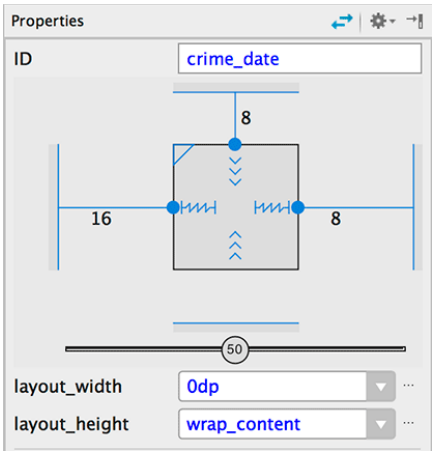
    (The selected widget will show squiggly lines for any of its constraints that are stretching.)

21. When you click on the TextView, you can see that it has an oval area that the ImageView did not have. TextViews have this additional constraint anchor that can be used to align text. You will not be using it in this assignment, but now you know what it is. Now that the constraints are set up, you can restore the title TextView to its full glory. Adjust its horizontal view setting to any size (0dp) to allow the title TextView to fill all of the space available within its constraints. 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. Verify that your settings match these:

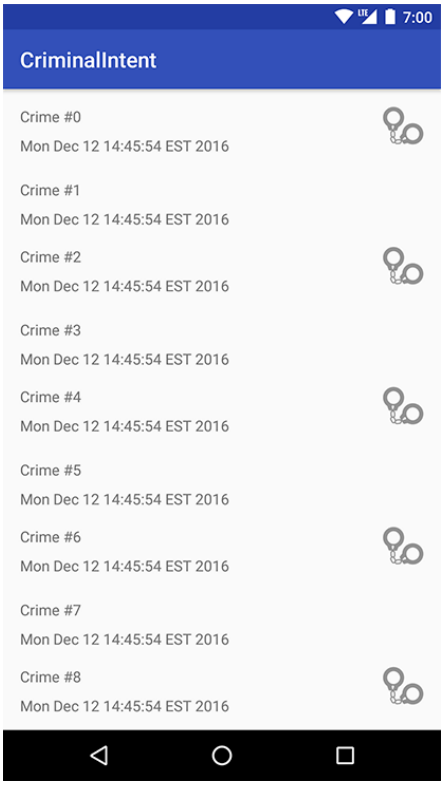


22. Now, add constraints to the date TextView. Select crime_date in the component tree. You are going to add three constraints:
    a) from the left side of your view to the left side of the parent, with a 16dp margin
    b) from the top of your view to the bottom of the crime title, with an 8dp margin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

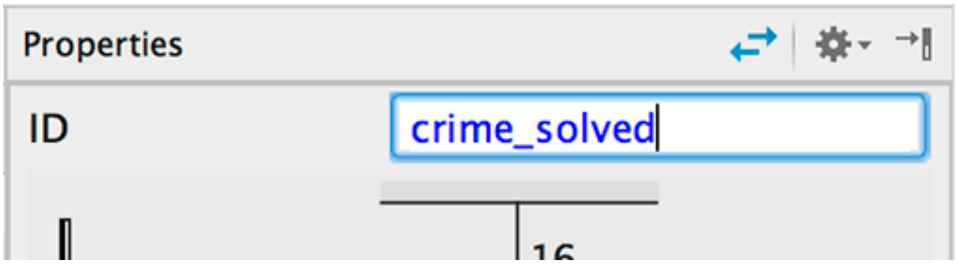
23. After adding the constraints, adjust the properties of the TextView. You want the width of your date TextView to be Any Size and the height to be Wrap Content, just like the title TextView. Verify that your settings match these:



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



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



26. With a proper ID in place, now you will update your code. Open CrimeListFragment.java. In CrimeHolder, add an ImageView instance variable and toggle its visibility based on the solved status of your crime.

```
private class CrimeHolder extends RecyclerView.ViewHolder
        implements View.OnClickListener {
    ...
    private TextView mDateTextView;
    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(crime.isSolved() ? View.VISIBLE : View.GONE);
    }
    ...
}
```

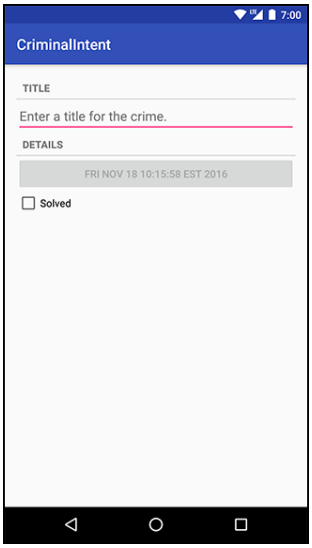
27. Let's add a few more tweaks to the design of list_item_crime.xml and, in the process, answer some lingering questions you might have about widgets and attributes. Navigate back to the Design view of list_item_crime.xml. Select crime_title and adjust some of the attributes in the properties view. Click the disclosure arrow next to textAppearance to reveal a set of text and font attributes. Update the textColor attribute to @android:color/black (Figure 9.27).

28. Next, set the textSize attribute to 18sp. Run CriminalIntent and be amazed at how much better everything looks with a fresh coat of paint.

29. In CrimeListFragment's CrimeHolder, begin by replacing the toast with code that starts an instance of CrimeActivity.

```
private class CrimeHolder extends RecyclerView.ViewHolder
        implements View.OnClickListener {
    ...
    @Override
    public void onClick(View view) {
        Toast.makeText(getActivity(),
                mCrime.getTitle() + " clicked!", Toast.LENGTH_SHORT)
                .show();
        Intent intent = new Intent(getActivity(), CrimeActivity.class);
        startActivity(intent);
    }
}
```

30. Run CriminalIntent. Press any list item, and you will see a new CrimeActivity hosting a CrimeFragment.



31. You can tell CrimeFragment which Crime to display by passing the crime ID as an Intent extra when CrimeActivity is started. Start by creating a newIntent method in CrimeActivity.

```
public class CrimeActivity extends SingleFragmentActivity {
    public static final String EXTRA_CRIME_ID =
            "com.bignerdranch.android.criminalintent.crime_id";

    public static Intent newIntent(Context packageContext, UUID crimeId) {
        Intent intent = new Intent(packageContext, CrimeActivity.class);
        intent.putExtra(EXTRA_CRIME_ID, crimeId);
        return intent;
    }
    ...
}
```

32. After creating an explicit intent, you call putExtra(…) and pass a string key and the value the key maps to (the crimeId). You are calling putExtra(String, Serializable) because UUID is a Serializable object. Update the CrimeHolder to use the newIntent method while passing in the crime ID.

```
private class CrimeHolder extends RecyclerView.ViewHolder implements View.OnClickListener {
    ...
    @Override
    public void onClick(View view) {
        ~~Intent intent = new Intent(getActivity(), CrimeActivity.class);~~
        Intent intent = CrimeActivity.newIntent(getActivity(), mCrime.getId());
        startActivity(intent);
    }
}
```

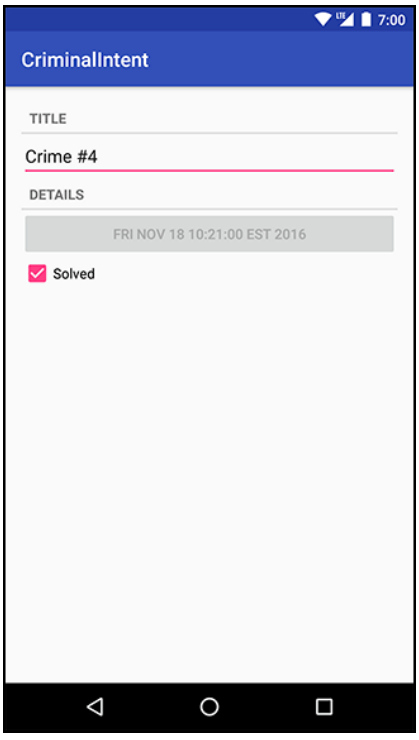
33. There are two ways a fragment can access data in its activity's intent: an easy, direct shortcut and a complex, flexible implementation. First, you are going to try out the shortcut. Then you will implement the complex and flexible solution. In the shortcut, CrimeFragment will simply use the getActivity() method to access the CrimeActivity's intent directly. In CrimeFragment.java, retrieve the extra from CrimeActivity's intent and use it to fetch the Crime.

```
public class CrimeFragment extends Fragment {
    ...
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ~~mCrime = new Crime();~~
        UUID crimeId = (UUID) getActivity().getIntent()
                .getSerializableExtra(CrimeActivity.EXTRA_CRIME_ID);
        mCrime = CrimeLab.get(getActivity()).getCrime(crimeId);
    }
    ...
}
```

34. Now that CrimeFragment fetches a Crime, its view can display that Crime's data. Update onCreateView(…) to display the Crime's title and solved status. (The code for displaying the date is already in place.)

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
    ...
    mTitleField = (EditText)v.findViewById(R.id.crime_title);
    mTitleField.setText(mCrime.getTitle());
    mTitleField.addTextChangedListener(new TextWatcher() {
        ...
    });
    ...
    mSolvedCheckBox = (CheckBox)v.findViewById(R.id.crime_solved);
    mSolvedCheckBox.setChecked(mCrime.isSolved());
    mSolvedCheckBox.setOnCheckedChangeListener(new OnCheckedChangeListener() {
        ...
    });
    ...
    return v;
}
```

35. Run CriminalIntent. Select Crime #4 and watch a CrimeFragment instance with the correct crime data appear.



36. In CrimeFragment, write a newInstance(UUID) method that accepts a UUID, creates an arguments bundle, creates a fragment instance, and then attaches the arguments to the fragment.

```
public class CrimeFragment extends Fragment {

   private static final String ARG_CRIME_ID = "crime_id";

   private Crime mCrime;
   private EditText mTitleField;
   private Button mDateButton;
   private CheckBox mSolvedCheckbox;

   public static CrimeFragment newInstance(UUID crimeId) {
      Bundle args = new Bundle();
      args.putSerializable(ARG_CRIME_ID, crimeId);

      CrimeFragment fragment = new CrimeFragment();
      fragment.setArguments(args);
      return fragment;
   }
   ...
}
```

37. Now, CrimeActivity should call CrimeFragment.newInstance(UUID) when it needs to create a CrimeFragment. It will pass in the UUID it retrieved from its extra. Return to CrimeActivity and, in createFragment(), retrieve the extra from CrimeActivity's intent and pass it into CrimeFragment.newInstance(UUID). You can now also make EXTRA_CRIME_ID private, because no other class will access that extra.

```
public class CrimeActivity extends SingleFragmentActivity {

   ~~public~~ private static final String EXTRA_CRIME_ID =
         "com.bignerdranch.android.criminalintent.crime_id";
   ...
   @Override
   protected Fragment createFragment() {
      ~~return new CrimeFragment();~~
      UUID crimeId = (UUID) getIntent().getSerializableExtra(EXTRA_CRIME_ID);
      return CrimeFragment.newInstance(crimeId);
   }
}
```

38. When a fragment needs to access its arguments, it calls the Fragment method getArguments() and then one of the type-specific "get" methods of Bundle. Back in CrimeFragment.onCreate(…), replace your shortcut code with retrieving the UUID from the fragment arguments:

```
@Override
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   ~~UUID crimeId = (UUID)~~
~~getActivity().getIntent().getSerializableExtra(CrimeActivity.EXTRA_CRIME_ID);~~
   UUID crimeId = (UUID) getArguments().getSerializable(ARG_CRIME_ID);
   mCrime = CrimeLab.get(getActivity()).getCrime(crimeId);
}
```

39. In CrimeListFragment, override onResume() and trigger a call to updateUI() to reload the list. Modify the updateUI() method to call notifyDataSetChanged() if the CrimeAdapter is already set up:

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                  Bundle savedInstanceState) {
   ...
}

@Override
public void onResume() {
   super.onResume();
   updateUI();
}

private void updateUI() {
   CrimeLab crimeLab = CrimeLab.get(getActivity());
   List<Crime> crimes = crimeLab.getCrimes();

   if (mAdapter == null) {
      mAdapter = new CrimeAdapter(crimes);
      mCrimeRecyclerView.setAdapter(mAdapter);
   } else {
      mAdapter.notifyDataSetChanged();
   }
}
```

40. Run CriminalIntent. Select a crime and change its details. When you return to the list, you will immediately see your changes.

Final App Example