

Android UI Fragment Basics

Javed Hasan
BJIT Limited

Fragment Lesson Content

Building Dynamic UI

- Fragment Basic

Sample Project Criminal Intent Overview

- Fragment and Activity Components
- Criminal Intent Object Diagram with MVC
- Criminal Intent Project Setup, Model Part

Fragment

- Fragment Lifecycle, Hosting using Activity Layout
- Create Fragment, Fragment Layout, Fragment Class
- Implementing Lifecycle methods, Wiring widgets in fragment

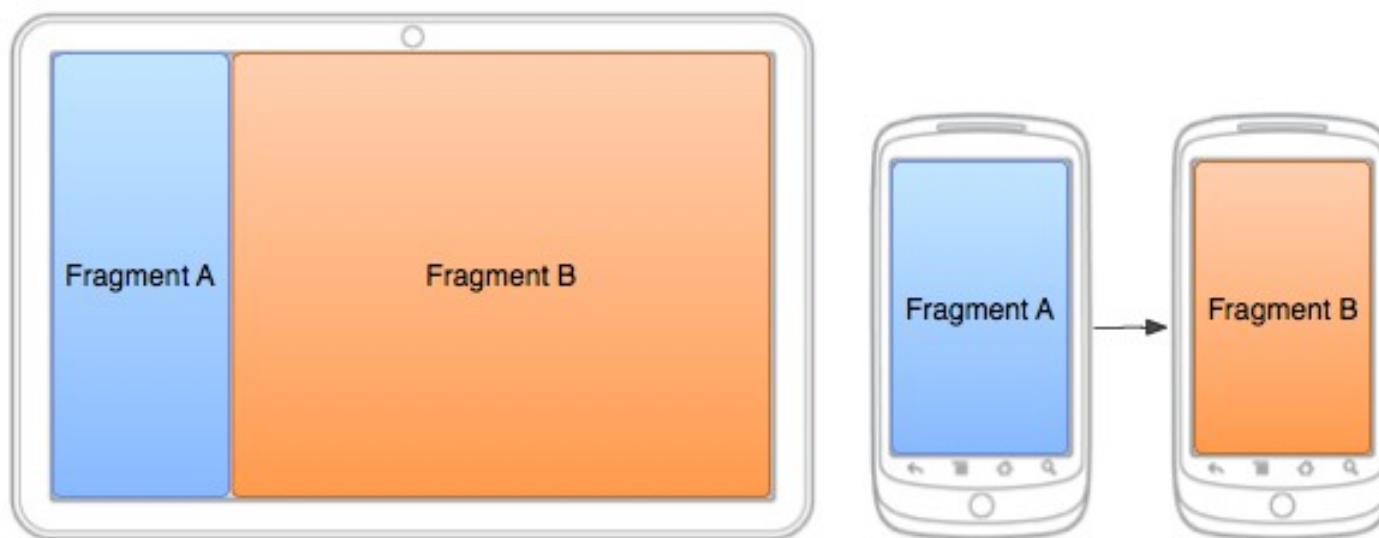
Fragment Manager

- Adding fragment to Fragment Manager, Fragment Transaction

Fragment Basic

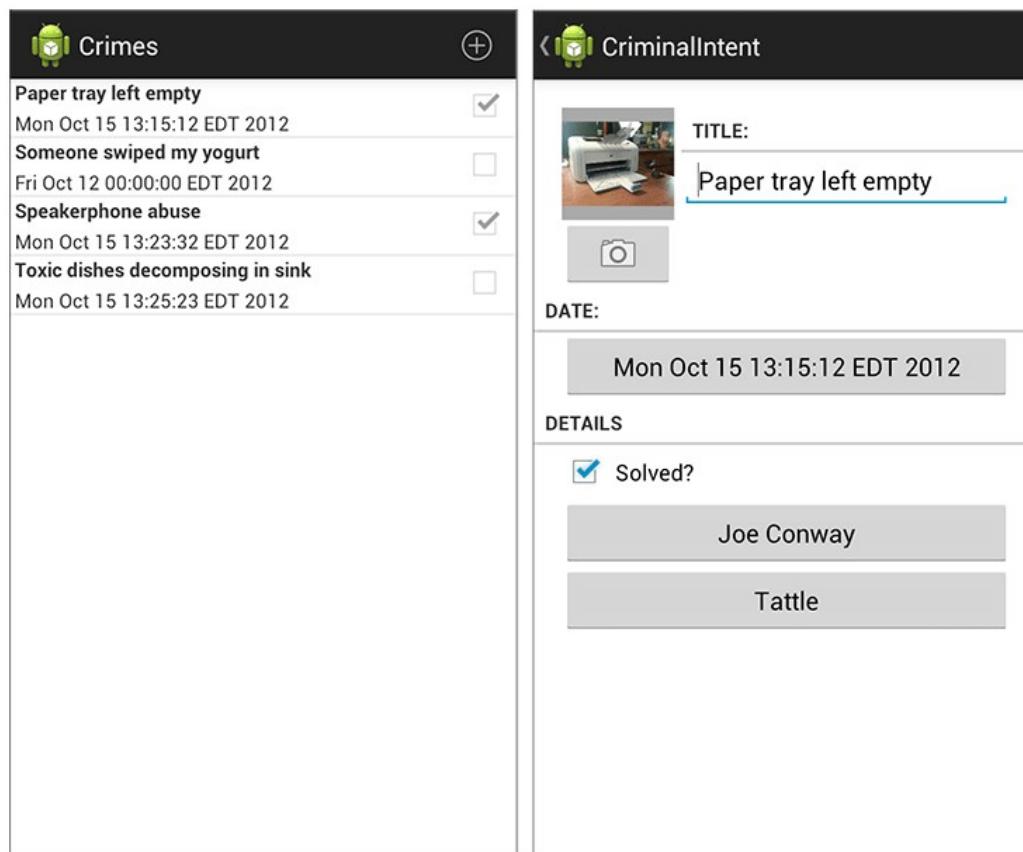
Fragment is a modular section of an activity, which has its own lifecycle, receives its own input events, and which you can add or remove while the activity is running.

When designing your application to support a wide range of screen sizes, you can reuse your fragments in different layout configurations to optimize the user experience based on the available screen space.

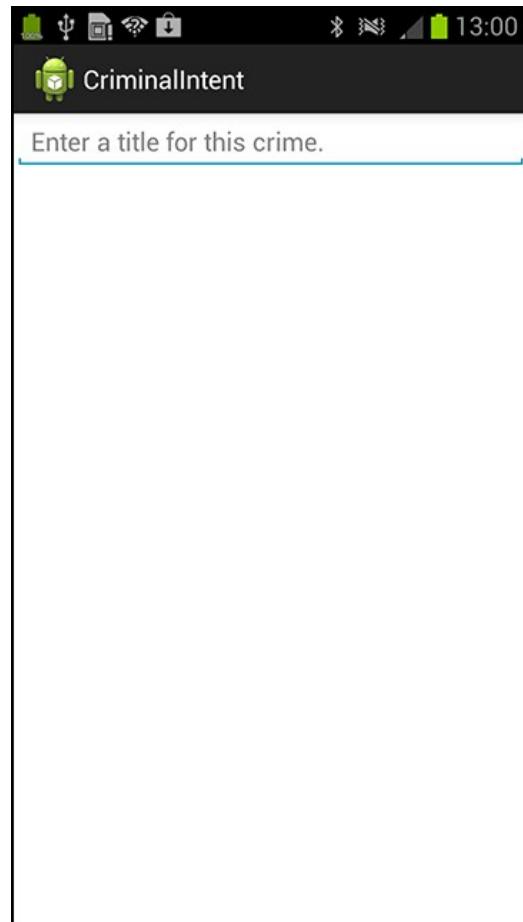


Main Screens of Sample Project CriminalIntent

CriminalIntent records the details of “office crimes” – things like leaving dirty dishes in the breakroom sink or walking away from an empty shared printer after your documents have printed.

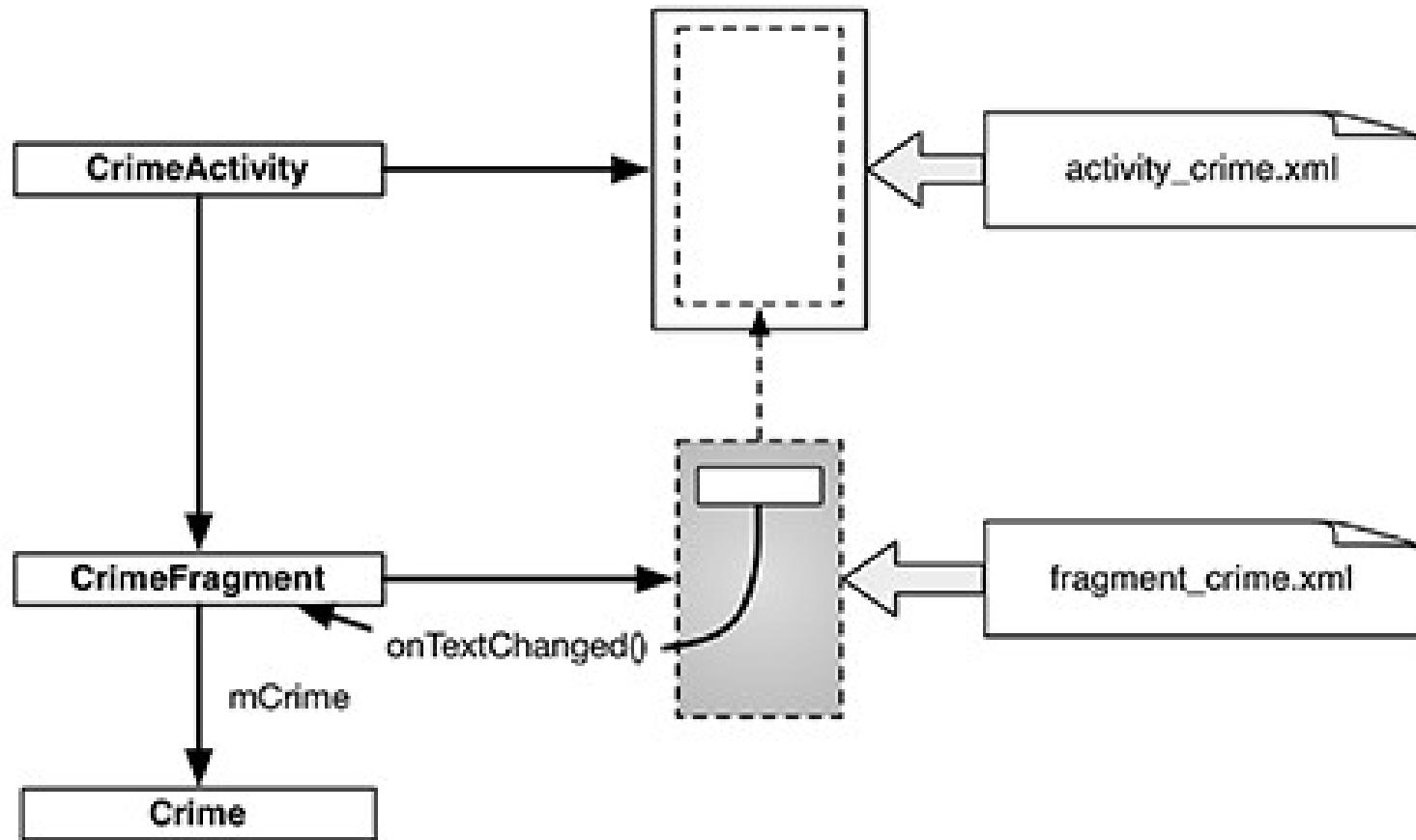


Main Screen for This Lesson



CriminalIntent Activity and Fragment

CrimeActivity hosts CrimeFragment



CriminalIntent Project Setup

New Android Application

New Android Application
Creates a new Android Application

Application Name: 

Project Name:

Package Name: **(This field is highlighted with a blue border)**

Minimum Required SDK:

Target SDK:

Compile With:

Theme:

 The package name must be a unique identifier for your application. It is typically not shown to users, but it **must** stay the same for the lifetime of the app. Multiple versions of the same application are considered the "same app". This is typically the reverse domain name of your organization plus one or more components, and it must be a valid Java package name.

< Back **Next >**

New Blank Activity

New Blank Activity
Creates a new blank activity, with optional inner navigation.

Activity Name: 

Layout Name:

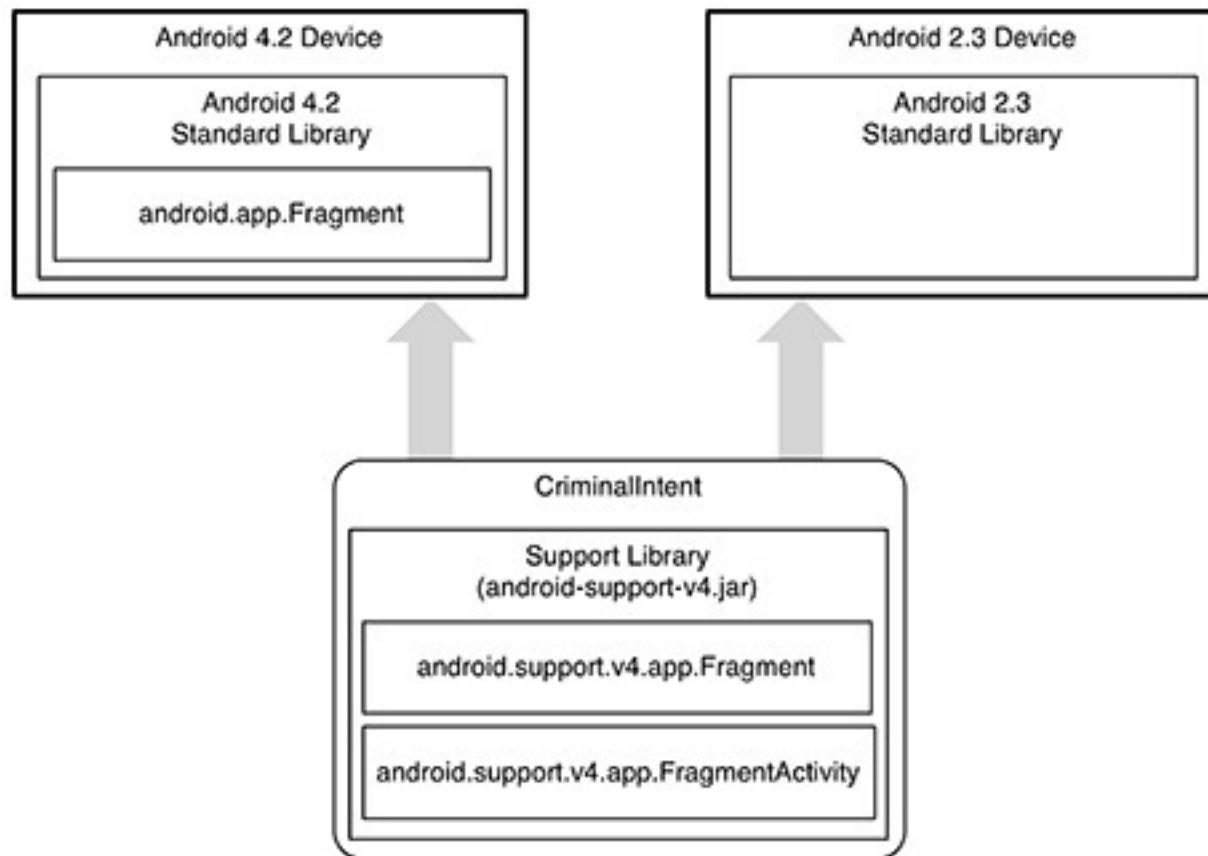
Navigation Type:

 The name of the activity class to create

< Back **Next >** Cancel **Finish**

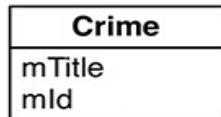
Fragments and Support Library

Fragments were introduced at API level 11. Android provides a support library to make fragments backward compatible as low as API level 4.

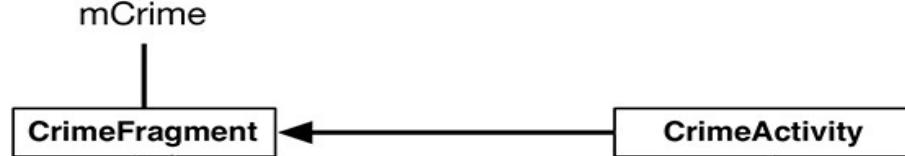


CriminalIntent Object Diagram

Model



Controller

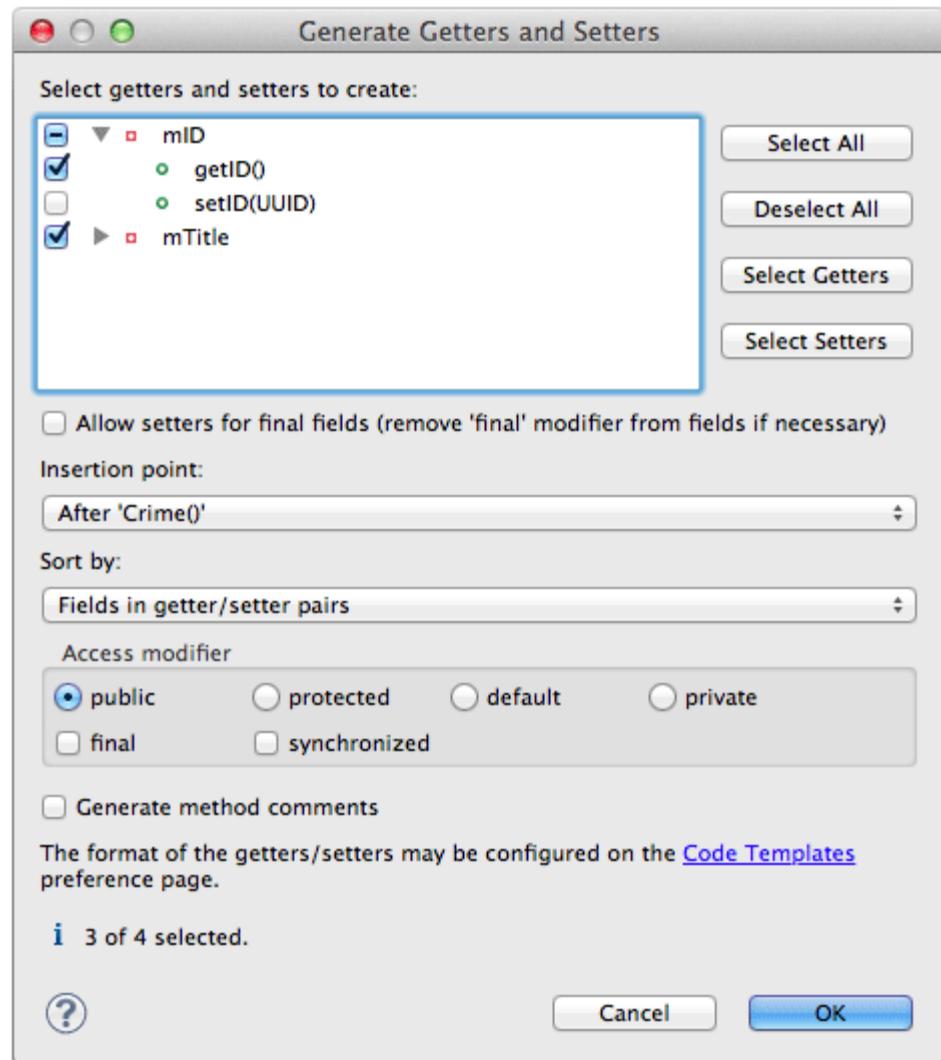


View (layout)



CriminalIntent Model – Crime Class

```
public class Crime {  
    private UUID mId;  
  
    private String mTitle;  
  
    public Crime() {  
        mId = UUID.randomUUID();  
    }  
  
    public UUID getId() {  
        return mId;  
    }  
  
    public String getTitle() {  
        return mTitle;  
    }  
  
    public void setTitle(String title) {  
        mTitle = title;  
    }  
}
```



Fragment Lifecycle

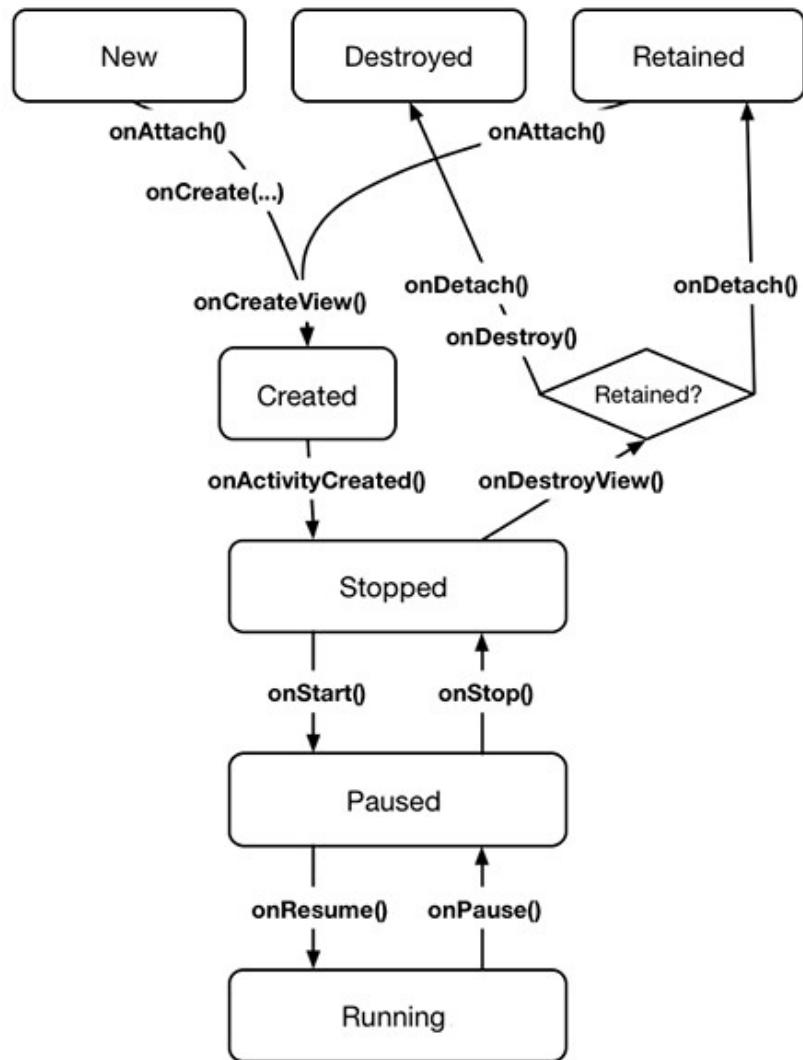
Fragment lifecycle is similar to the activity lifecycle.

It has stopped, paused, and running states.

It has methods you can override to get things done at critical points.

One critical difference between the fragment lifecycle and the activity lifecycle is that fragment lifecycle methods are called by the hosting activity, not the OS.

The OS knows nothing about the fragments that an activity is using to manage things. Fragments are the activity's internal business.



Two Ways of Hosting UI Fragment 1/2

1. Add the fragment to activity's layout

- Inflexible. Hard-wire the fragment and its view to the activity's view
- Can not swap out the fragment during runtime

2. Add the fragment in the activity's code

- Complex but flexible
- Add, remove and replace of fragments in an activity is possible during runtime

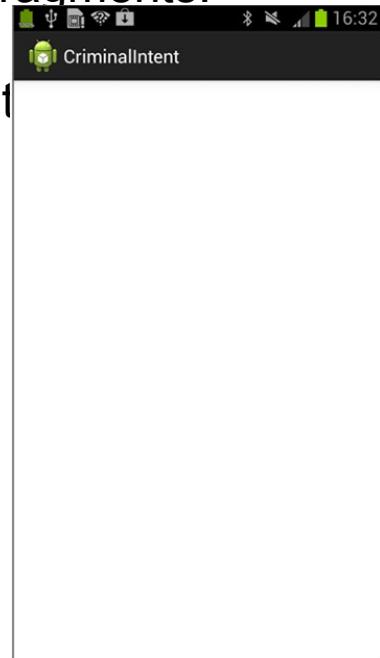
In this lesson, we will use the second approach to host fragment in activity

Make a Fragment Container in Activity for Hosting

This `FrameLayout` will be the container view for a `CrimeFragment`.

Notice that the container view is completely generic; it does not name the `CrimeFragment` class. You can and will use this same layout to host other fragments.

Create fragment container
(`activity_crime.xml`)



Fragment-hosting layout for `CrimeActivity`

FrameLayout

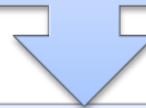
```
xmlns:android="http://schemas.android.com/apk/res/android"  
android:id="@+id/fragmentContainer"  
android:layout_width="match_parent"  
android:layout_height="match_parent"/>
```

`activity_crime.xml`

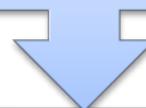
```
<?xml version="1.0" encoding="utf-8"?>  
<FrameLayout  
    xmlns:android="http://schemas.android.com/apk/res/android"  
        android:id="@+id/fragmentContainer"  
        android:layout_width="match_parent"  
        android:layout_height="match_parent"  
    />
```

Creating an UI Fragment

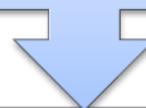
Step1: Compose an interface by defining widgets in a layout file



Step2: Create the class and set its view to be the layout that you defined



Step3: Implement fragment lifecycle methods



Step4: wire up the widgets inflated from the layout in code

Step1: Defining CrimeFragment Layout 1/2

```
LinearLayout  
xmlns:android="http://schemas.android.com/apk/res/android"  
android:layout_width="match_parent"  
android:layout_height="match_parent"  
android:orientation="vertical"
```

```
↓  
EditText  
android:id="@+id/crime_title"  
android:layout_width="match_parent"  
android:layout_height="wrap_content"  
android:hint="@string/crime_title_hint"
```

Fragment_crime.xml layout file

```
<?xml version="1.0" encoding="utf-8"?>  
<LinearLayout  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:orientation="vertical"  
>  
<EditText android:id="@+id/crime_title"  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content"  
        android:hint="@string/crime_title_hint"  
    />  
</LinearLayout>
```

Step1: Defining CrimeFragment Layout 2/2

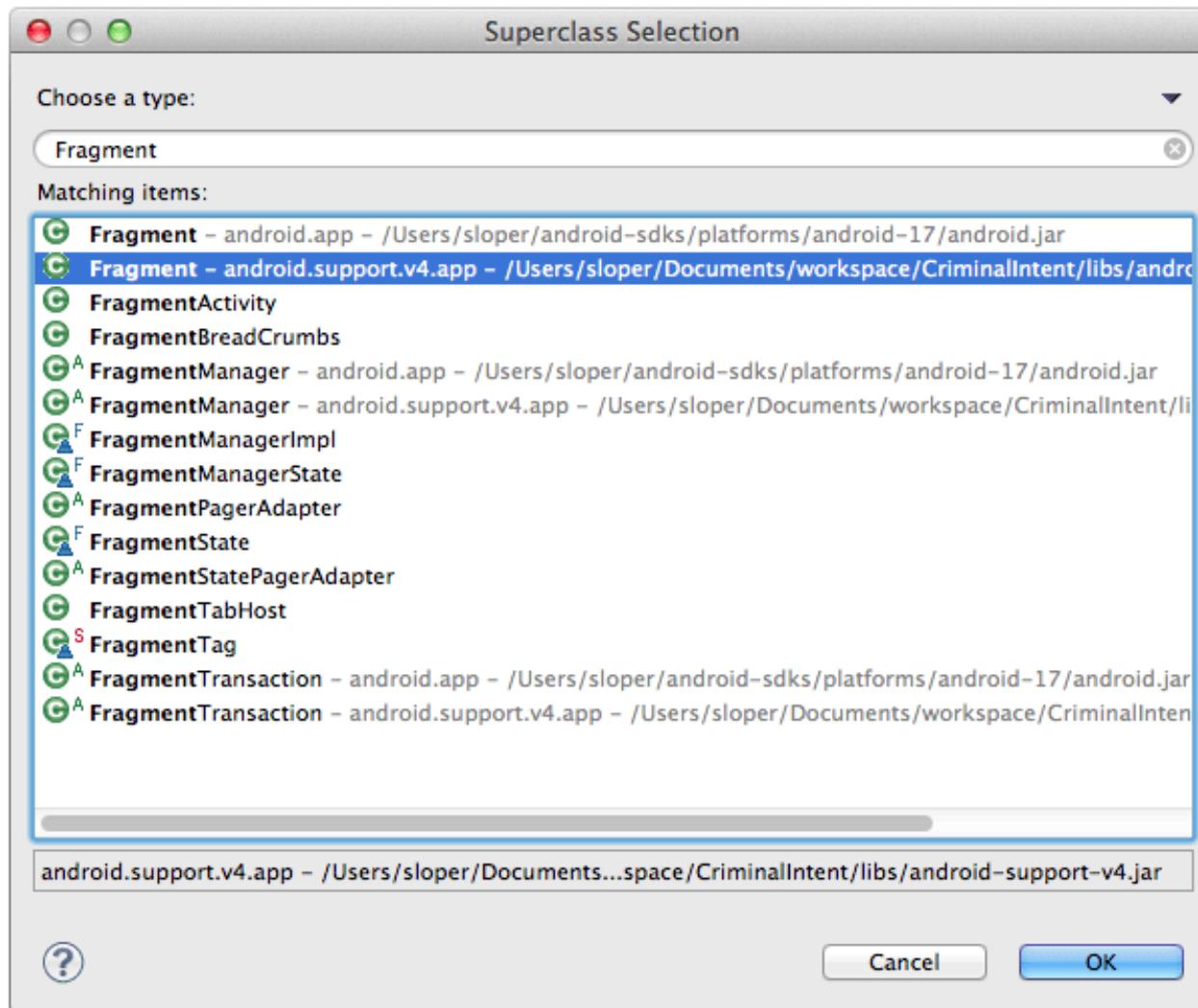
Adding and deleting strings
res/values/strings.xml

```
<?xml version="1.0" encoding="utf-8"?>  
<resources>  
    <string name="app_name">CriminalIntent</string>  
    <string name="hello_world">Hello world!</string>  
    <string name="menu_settings">Settings</string>  
    <string name="title_activity_crime">CrimeActivity</string>  
    <string name="crime_title_hint">Enter a title for the  
    crime.</string>  
</resources>
```

Tweaking template code
(CrimeActivity.java)

```
public class CrimeActivity extends Activity  
    FragmentActivity {  
        @Override  
        public void onCreate(Bundle savedInstanceState) {  
            super.onCreate(savedInstanceState);  
            setContentView(R.layout.activity_crime);  
        }  
        @Override  
        public boolean onCreateOptionsMenu(Menu menu) {  
            getMenuInflater().inflate(R.menu.activity_crime_,  
                menu);  
            return true;  
        }  
    }
```

Step2: Creating the CrimeFragment class



Step3: Implementing Fragment Lifecycle Methods

```
public class CrimeFragment extends Fragment {  
    private Crime mCrime;  
  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        mCrime = new Crime();  
    }  
  
    @Override  
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,  
        Bundle savedInstanceState) {  
        View v = inflater.inflate(R.layout.fragment_crime, parent, false);  
        return v;  
    }  
}
```

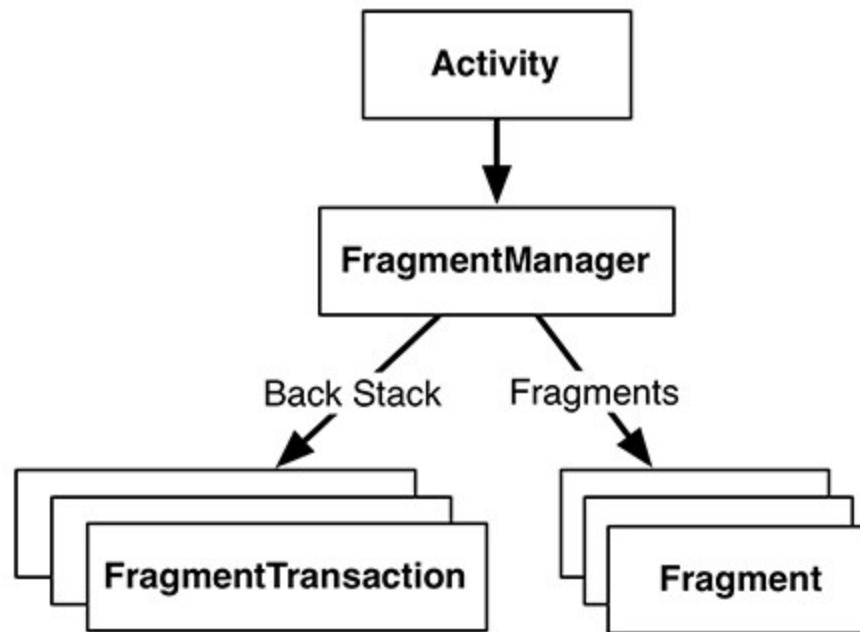
Step4: Wiring Widgets in a Fragment

```
public class CrimeFragment extends Fragment {  
    private Crime mCrime;  
    private EditText mTitleField;  
    ...  
    public View onCreateView(LayoutInflater inflater, ViewGroup parent, Bundle savedInstanceState) {  
        View v = inflater.inflate(R.layout.fragment_crime, parent, false);  
        mTitleField = (EditText)v.findViewById(R.id.crime_title);  
        mTitleField.addTextChangedListener(new TextWatcher() {  
            public void onTextChanged(  
                CharSequence c, int start, int before, int count) {  
                mCrime.setTitle(c.toString());  
            }  
            public void beforeTextChanged(  
                CharSequence c, int start, int count, int after) {  
                // This space intentionally left blank  
            }  
            public void afterTextChanged(Editable c) {  
                // This one too  
            } });  
        return v;  
    } }
```

Fragment Manager

The FragmentManager is responsible for managing your fragments and adding their views to the activity's view hierarchy.

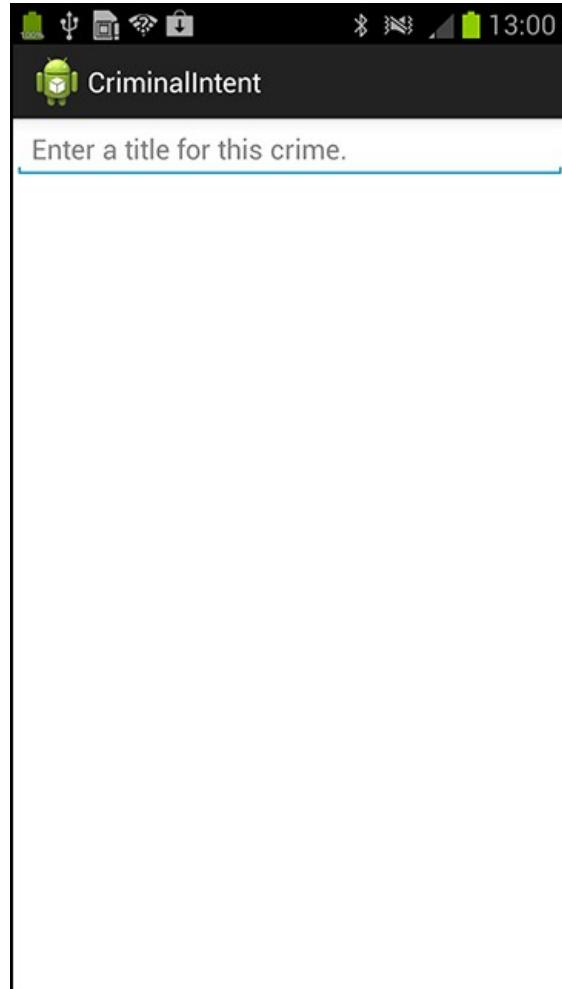
The FragmentManager handles two things: a list of fragments and a back stack of fragment transactions



Adding a UI Fragment to FragmentManager

```
public class CrimeActivity extends FragmentActivity {  
    /** Called when the activity is first created. */  
  
    @Override  
  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
  
        setContentView(R.layout.activity_crime);  
  
        FragmentManager fm = getSupportFragmentManager();  
  
        Fragment fragment = fm.findFragmentById(R.id.fragmentContainer);  
  
        if (fragment == null) {  
            fragment = new CrimeFragment();  
  
            fm.beginTransaction()  
                .add(R.id.fragmentContainer, fragment)  
                .commit();  
        }  
    }  
}
```

CrimeFragment's view hosted by CrimeActivity



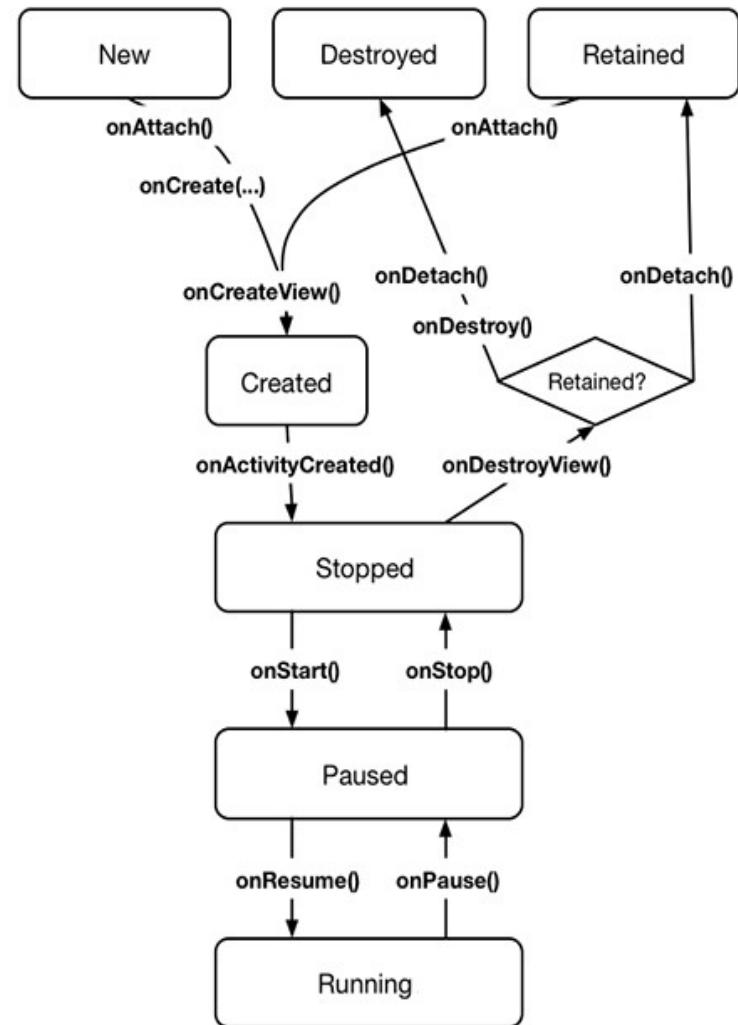
The FragmentManager and Fragment Lifecycle

The FragmentManager of an activity is responsible for calling the lifecycle methods of the fragments in its list.

The `onAttach(Activity)`, `onCreate(Bundle)`, and `onCreateView(...)` are called when you add the fragment to the FragmentManager.

The `onActivityCreated(...)` is called after the hosting activity's `onCreate(...)` method has executed.

What happens if you add a fragment while the activity is already stopped, paused, or running? In that case, the FragmentManager immediately walks the fragment through whatever steps are necessary to get it caught up to the activity's state.



Android UI

Create UI with Layout & Widgets

Javed Hasan
BJIT Limited

Lesson Content

CriminalIntent New Features Overview

- Main Screen, Updated Portrait Layout, New Landscape Layout

Discussion on XML Layout Attributes

- Style, Theme and Theme Attributes
- Screen Pixel Densities dp and sp
- Layout Parameters, Margin vs Padding, Layout Weight

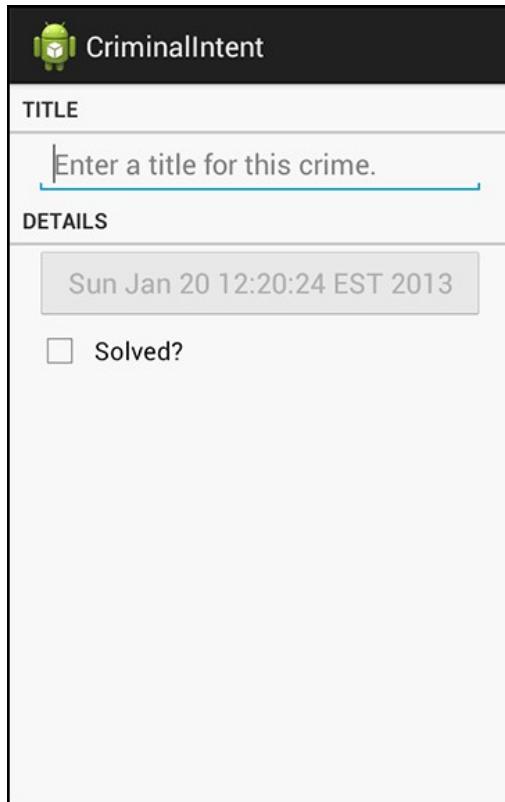
Design UI with Graphical Layout Tool

- Add new widgets, edit properties, reorganize widgets

Important notes on Managing Multiple Layout

- Widget IDs and multiple layouts

Updated Portrait and Landscape Layout



Layout Style and Themes

A **style** is a collection of properties that specify the look and format for a view or window.

A style can specify properties such as height, padding, font color, font size, background color, and much more.

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:layout_margin">3dp</item>  
</style>
```

A **theme** is a collection of styles.

Android provides platform themes that your apps can use. CriminalIntent uses **Holo Light with Dark Action Bar** as the app's theme by default.

You can apply a style from the app's theme to a widget using a theme attribute reference.

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

sp and dp

A **dp** is a density-independent pixel that corresponds to the physical size of a pixel at 160 dpi.

An **sp** is the same base unit, but is scaled by the user's preferred text size (it's a scale-independent pixel).

Defining layout dimensions with pixels is a problem because different screens have different pixel densities, so the same number of pixels may correspond to different physical sizes on different devices. Therefore, when specifying dimensions, always use either **dp** or **sp** units.

For example, when you specify spacing between two views, use dp rather than physical pixel size px:

```
<Button android:layout_width="wrap_content"  
       android:layout_height="wrap_content"  
       android:text="@string/clickme"  
       android:layout_marginTop="20dp" />
```

When specifying text size, always use sp:

```
<TextView android:layout_width="match_parent"  
         android:layout_height="wrap_content"  
         android:textSize="20sp" />
```

Example: Dimension Units in Action on TextView

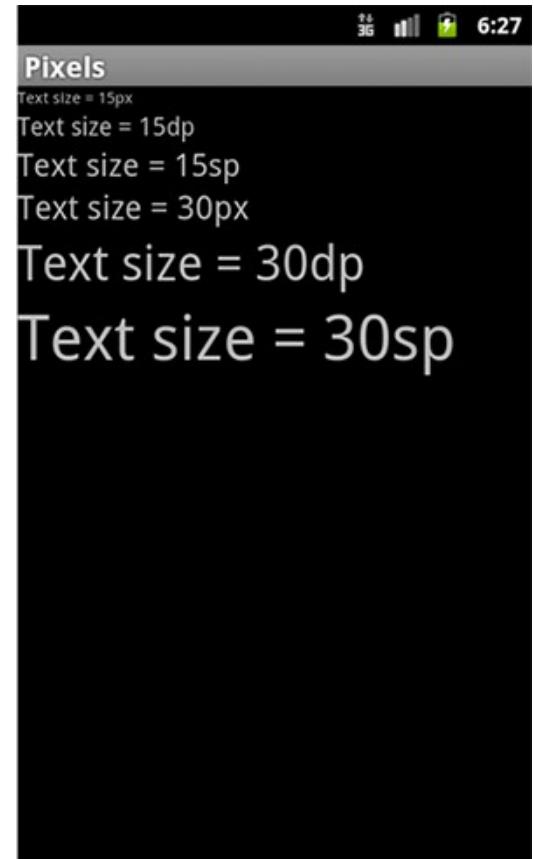
Left: MDPI



Middle: HDPI



Right: HDPI with large text



Margin vs Padding

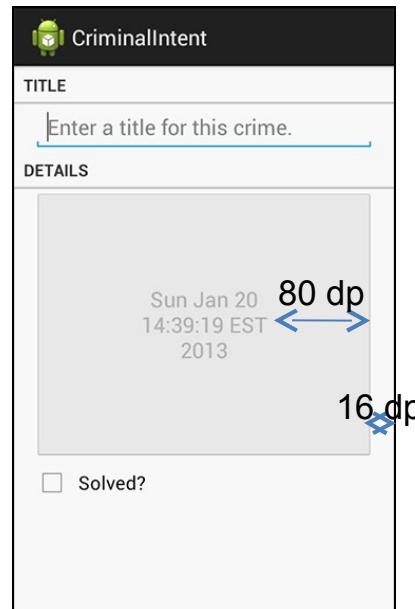
Margins specify the distances between views, and **padding** specifies the distance between a view's outside edges and its content.

Attributes whose names do not begin with **layout_** are directions to the widget. **Padding** is the attribute of the widget.

When an attribute's name begins with **layout_**, these attributes are known as layout parameters, and they tell the parent layout how to arrange the child element within the parent. **Margin** attributes are layout parameters.

Padding in Action

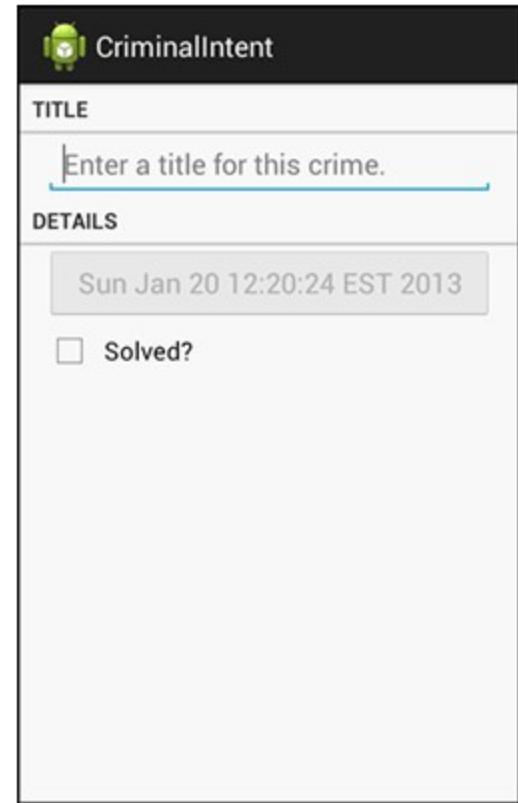
```
<Button android:id="@+id/crime_date"  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content"  
        android:layout_marginLeft="16dp"  
        android:layout_marginRight="16dp"  
        android:padding="80dp" />
```



Updating the Layout

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/crime_title_label"
        style="?android:listSeparatorTextViewStyle"
    />
    <EditText android:id="@+id/crime_title"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginLeft="16dp"
        android:layout_marginRight="16dp"
        android:hint="@string/crime_title_hint"
    />
```

```
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/crime_details_label"
    style="?android:listSeparatorTextViewStyle"
/>
<Button android:id="@+id/crime_date"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginLeft="16dp"
    android:layout_marginRight="16dp"
/>
<CheckBox android:id="@+id/crime_solved"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginLeft="16dp"
    android:layout_marginRight="16dp"
    android:text="@string/crime_solved_label" />
</LinearLayout>
```



Adding String Resources

Adding string resources (strings.xml)

```
<resources>
    <string name="app_name">CriminalIntent</string>
    <string name="title_activity_crime">CrimeActivity</string>
    <string name="crime_title_hint">Enter a title for this crime.</string>
    <string name="crime_title_label">Title</string>
    <string name="crime_details_label">Details</string>
    <string name="crime_solved_label">Solved?</string>
</resources>
```

Upgrading Model Crime

Adding more fields to Crime (Crime.java)

```
public class Crime {  
    private UUID mId;  
    private String mTitle;  
    private Date mDate;  
    private boolean mSolved;  
  
    public Crime() {  
        mId = UUID.randomUUID();  
        mDate = new Date();  
    }  
    ...  
}
```

Generated getters and setters (Crime.java)

```
public class Crime {  
    ...  
    public void setTitle(String title) {  
        mTitle = title;  
    }  
    public Date getDate() {  
        return mDate;  
    }  
    public void setDate(Date date) {  
        mDate = date;  
    }  
    public boolean isSolved() {  
        return mSolved;  
    }  
    public void setSolved(boolean solved) {  
        mSolved = solved;  
    }  
}
```

Wiring Widgets 1/2

Adding widget instance variables (CrimeFragment.java)

```
public class CrimeFragment extends Fragment {  
  
    private Crime mCrime;  
  
    private EditText mTitleField;  
  
    private Button mDateButton;  
  
    private CheckBox mSolvedCheckBox;  
  
  
    public void onCreate(Bundle savedInstanceState) {  
  
        ...  
    }
```

Setting Button text (CrimeFragment.java)

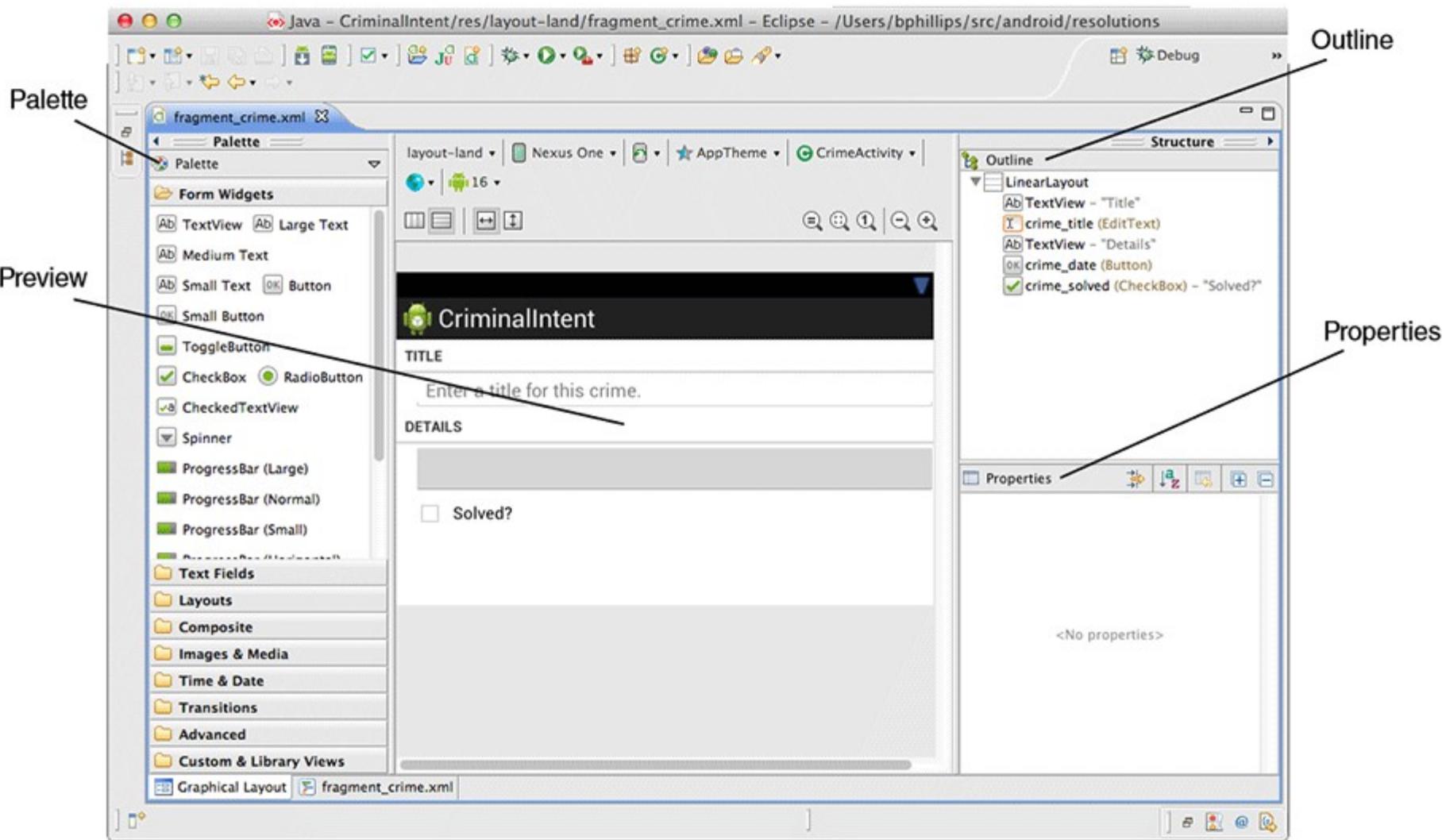
```
@Override  
  
public View onCreateView(LayoutInflater inflater, ViewGroup  
parent, Bundle savedInstanceState) {  
  
    View v = inflater.inflate(R.layout.fragment_crime, parent, false);  
  
    ...  
  
    mTitleField.addTextChangedListener(new TextWatcher() {  
  
        ...  
    });  
  
    mDateButton = (Button)v.findViewById(R.id.crime_date);  
  
    mDateButton.setText(mCrime.getDate().toString());  
  
    mDateButton.setEnabled(false);  
  
    return v;  
}
```

Wiring Widgets 2/2

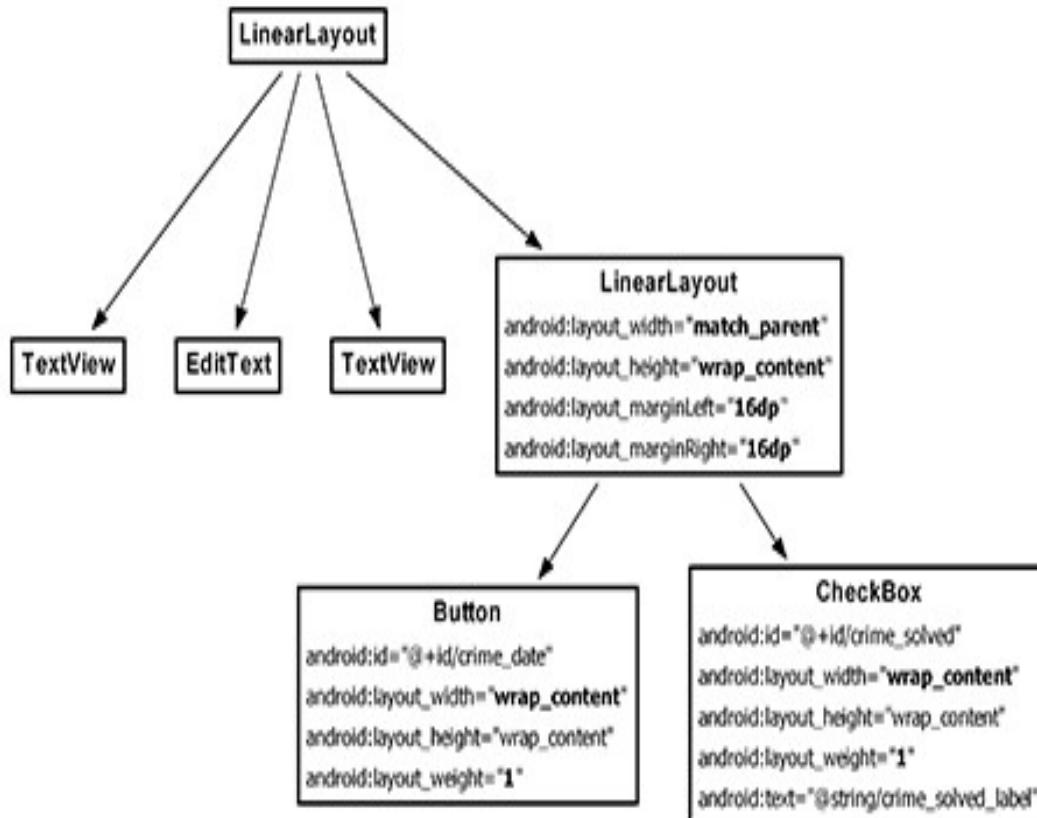
Listening for CheckBox changes (CrimeFragment.java)

```
...  
  
mDateButton = (Button)v.findViewById(R.id.crime_date);  
mDateButton.setText(mCrime.getDate().toString());  
mDateButton.setEnabled(false);  
  
mSolvedCheckBox = (CheckBox)v.findViewById(R.id.crime_solved);  
mSolvedCheckBox.setOnCheckedChangeListener(new OnCheckedChangeListener() {  
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {  
        // Set the crime's solved property  
        mCrime.setSolved(isChecked);  
    }  
});  
  
return v;  
}
```

Design Layout with Graphical Layout Tool



Landscape Layout for CrimeFragment

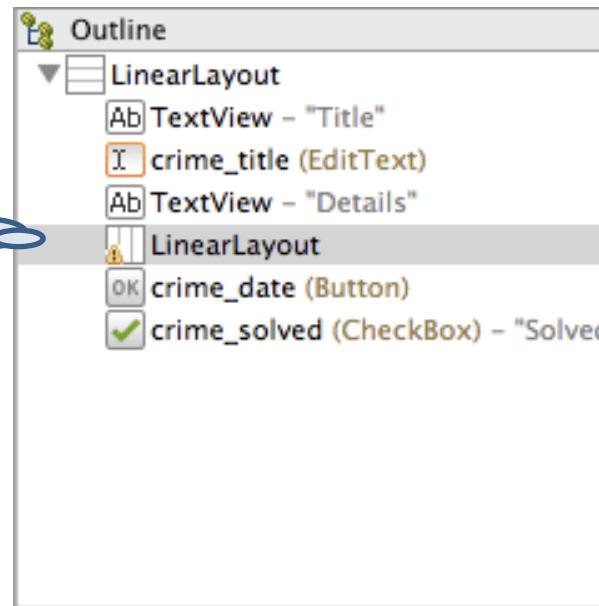


Adding a New Widget

You can add a widget by selecting it in the palette and then dragging to the outline view.

Click the Layouts category in the palette. Select LinearLayout (Horizontal) and drag to the outline view. Drop this LinearLayout on top of the root LinearLayout to add it as a direct child of the root LinearLayout.

Linear layout added to
fragment_crime.xml



Editing Attributes in Properties

Select the new LinearLayout in the outline view to display its attributes in the properties view. Expand the Layout Parameters category and then the Margins category.

You want the new LinearLayout's margins to match your other widgets. Select the field next to Left and type 16dp. Do the same for the right margin.

Margins set in properties view

The screenshot shows the Android Studio Properties tab. The 'Layout Para...' category is expanded, revealing 'Width' (match_parent), 'Height' (wrap_content), 'Weight', and 'Gravity'. The 'Margins' category is also expanded, showing 'Margin', 'Left' (16dp), 'Top', 'Right' (16dp), 'Bottom', 'Start', 'End', and 'Orientation'. The 'Right' row is highlighted with a blue background.

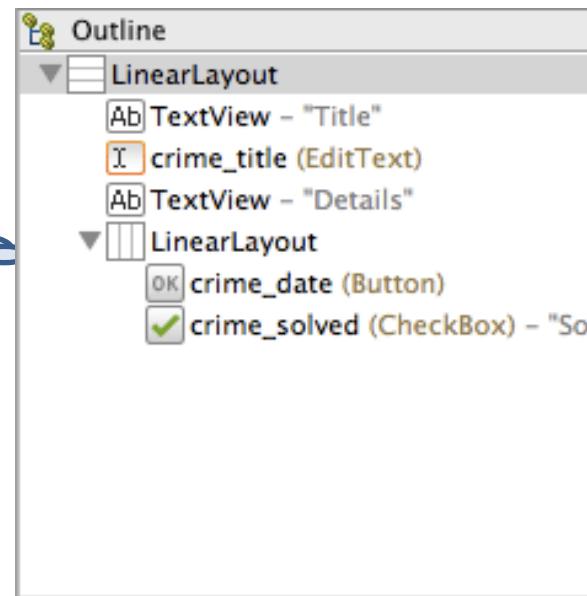
Properties	
<input checked="" type="checkbox"/>	Id
<input type="checkbox"/>	Layout Para...
Width	match_parent
Height	wrap_content
Weight	
Gravity	
<input type="checkbox"/>	Margins
Margin	
Left	16dp
Top	
Right	16dp
Bottom	
Start	
End	
Orientation	

Reorganizing Widgets in the Outline View

The next step is to make the Button and CheckBox children of the new LinearLayout. Return to the graphical layout tool, and, in the outline view, select the Button and drag it on top of the LinearLayout.

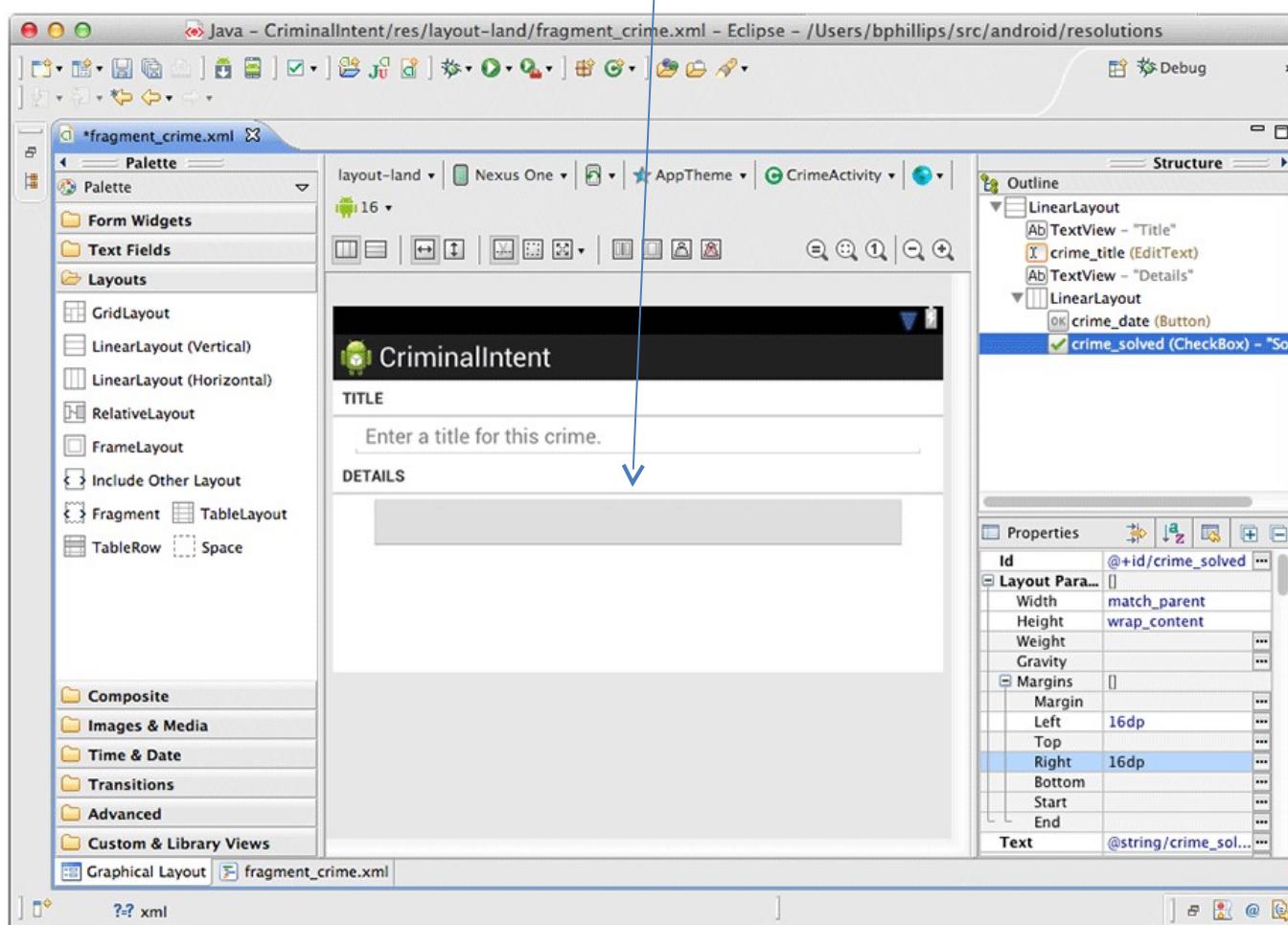
The outline should reflect that the Button is now a child of the new LinearLayout. Do the same for the CheckBox.

Button set as child of
new LinearLayout



Preview Does Not Show All Widgets

The first-defined Button child obscures the CheckBox



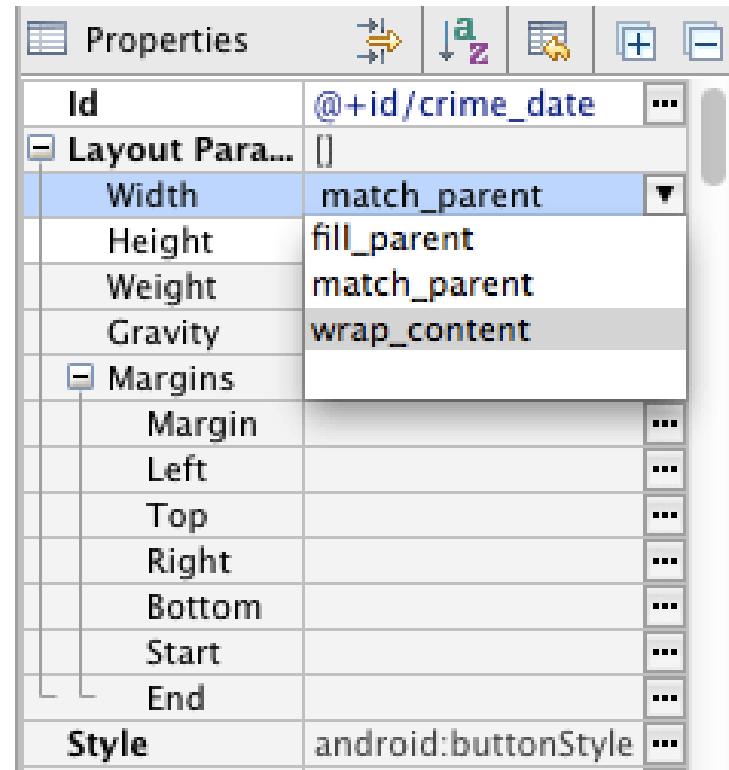
Updating Child Layout Parameters

First, select the date button in the outline. In the properties view, click on the current Width value and change it to wrap_content.

Next, delete both of the button's 16dp margin values. The button will not need these margins now that it is inside the LinearLayout.

Finally, find the Weight field in the Layout Parameters section and set its value to 1.

Check the preview to confirm that both widgets are now visible. Then save your file and return to the XML to confirm your changes.



XML for the graphically-created layout

(layout-land/fragment_crime.xml)

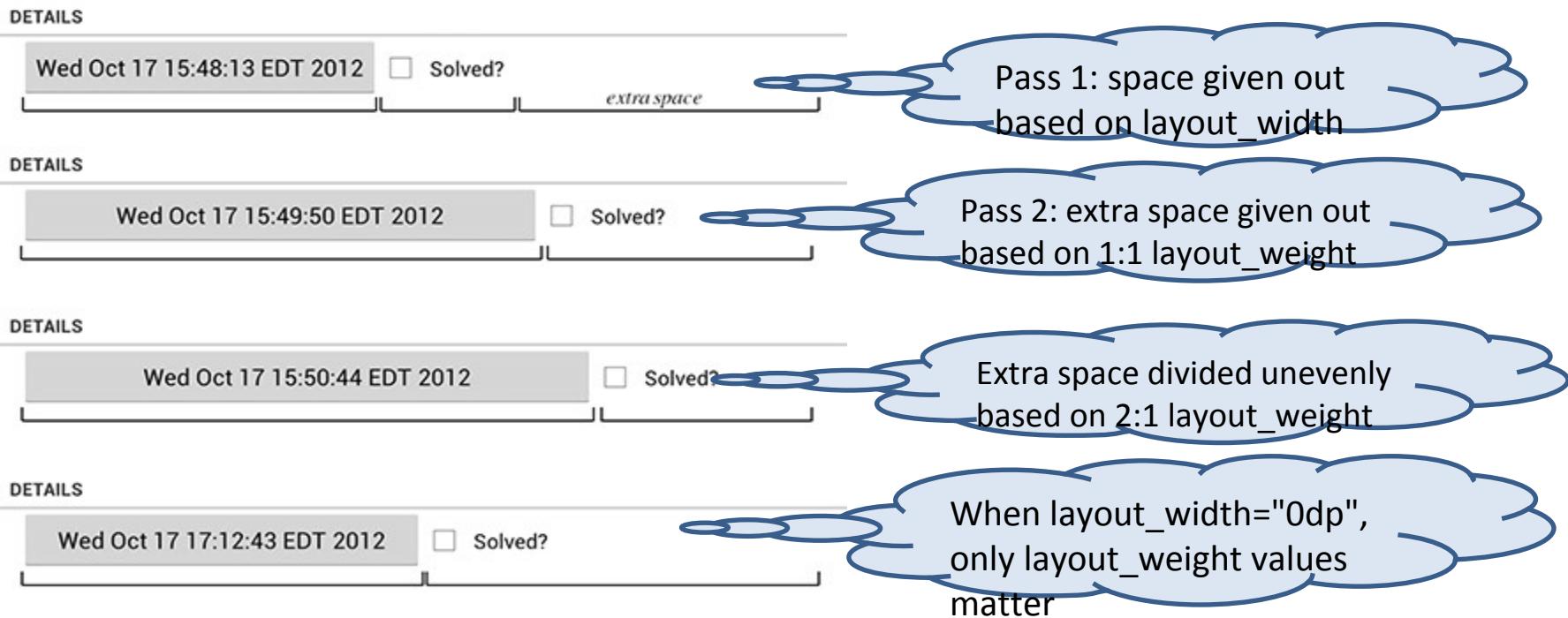
```
...<Button  
      android:id="@+id/crime_date"  
      android:layout_width="wrap_content"  
      android:layout_height="wrap_content"  
      android:layout_weight="1" />  
<CheckBox  
      android:id="@+id/crime_solved"  
      android:layout_width="wrap_content"  
      android:layout_height="wrap_content"  
      android:layout_weight="1"  
      android:text="@string/crime_solved_label" />  
</LinearLayout>  
</LinearLayout>
```

```
<TextView  
      android:layout_width="match_parent"  
      android:layout_height="wrap_content"  
      android:text="@string/crime_details_label"  
      style="?android:listSeparatorTextViewStyle"  
      />  
<LinearLayout  
      android:layout_width="match_parent"  
      android:layout_height="wrap_content"  
      android:layout_marginLeft="16dp"  
      android:layout_marginRight="16dp" >
```

How android:layout_weight Works

LinearLayout makes two passes to set the width of a view.

In the **first pass**, LinearLayout looks at layout_width. The value for layout_width for both the Button and CheckBox is now wrap_content, so each view will get only enough space to draw itself. In the **next pass**, LinearLayout allocates any extra space based on the values for layout_weight .



Summary of Graphical Layout Tool

The graphical layout tool is useful, and Android is improving it with every ADT release. However, it can still be slow and buggy at times.

Sometimes, you may want to go back to typing XML. You can switch between making changes in the graphical layout tool and in XML. Just be sure to save the file before switching tabs to be safe.

Widget IDs and Multiple Layouts

The two layouts that you have created for CriminalIntent do not vary significantly, but there may be times when your layouts will. When this is the case, you should ensure that widgets actually exist before you access them in code.

If you have a widget in one layout and not another, use null-checking in the code to determine if the widget is present in the current orientation before calling methods on it:

```
Button landscapeOnlyButton = (Button)v.findViewById(R.id.landscapeOnlyButton);
if (landscapeOnlyButton != null) {
    // Set it up
}
```

Finally, remember that a widget must have the same `android:id` attribute in every layout in which it appears so that your code can find it.

Challenge: 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 “Oct 12, 2012.”

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, “Tuesday, Oct 12, 2012.”

Android UI Display List with ListFragment

Javed Hasan
BJIT Limited

Lesson Content

CriminalIntent New Features Overview

- Display List of Crimes

Update CriminalIntent Model Layer

- Singleton and Centralized Data Storage

Create a ListFragment

- Add new widgets, edit properties, reorganize widgets

Design an Abstract Activity for Hosting a Fragment

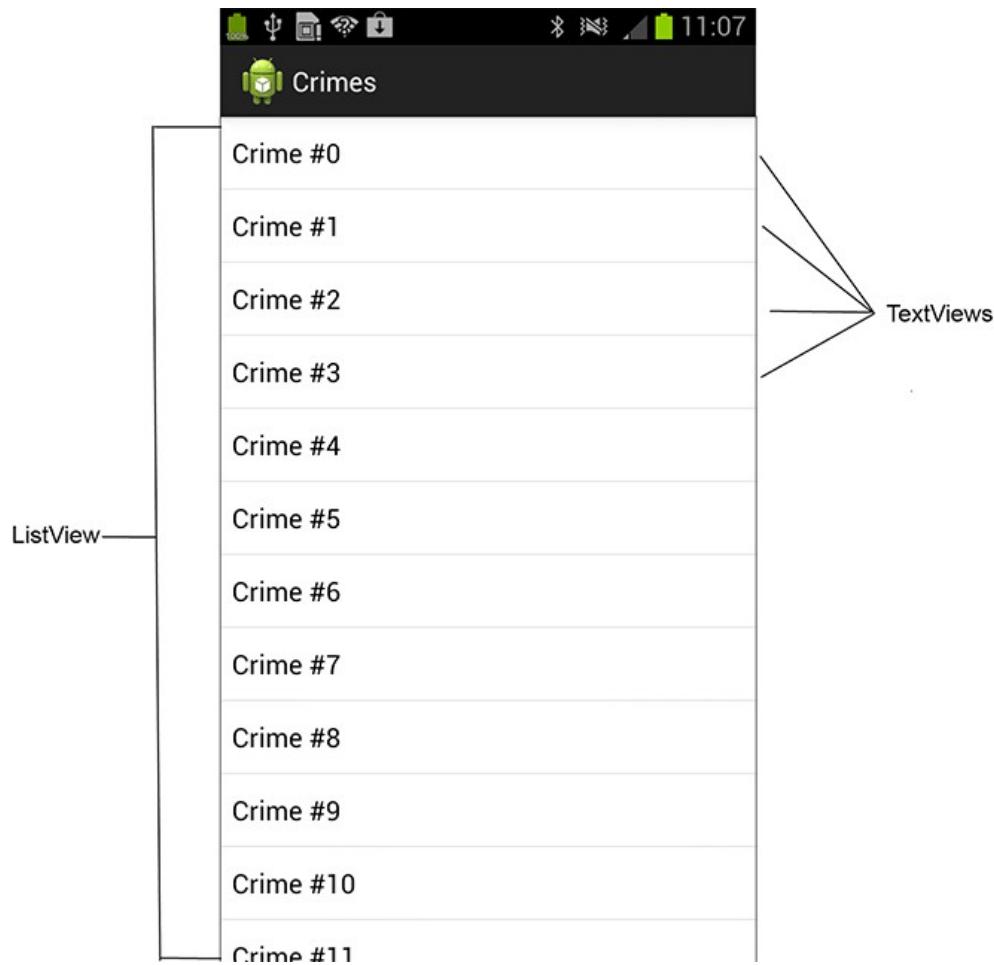
- Update CrimeActivity & Create CrimeListActivity from Abstract FragmentActivity

ListFragment, ListView, and ArrayAdapter

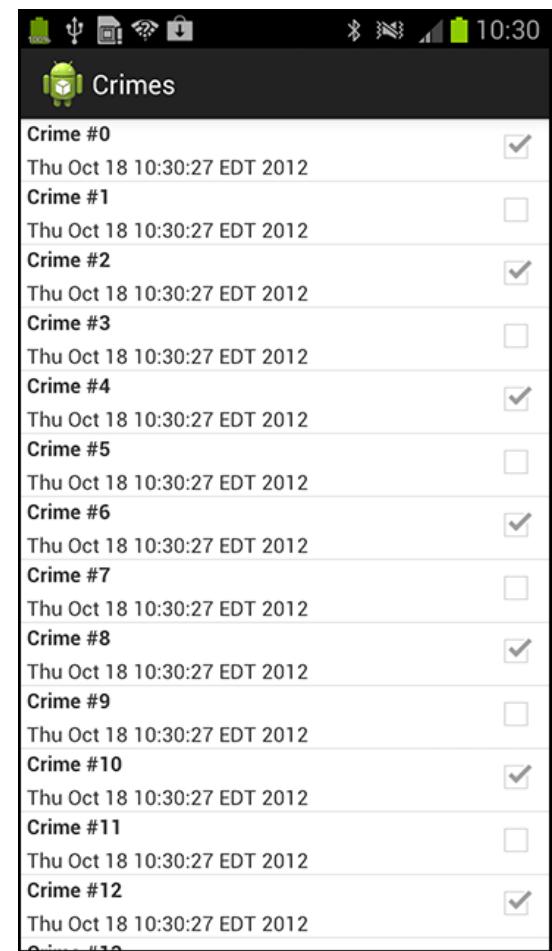
- Customizing List Items

Main Screen for This Lesson

Step 1: Simple Prototype



Step 2: Final Version

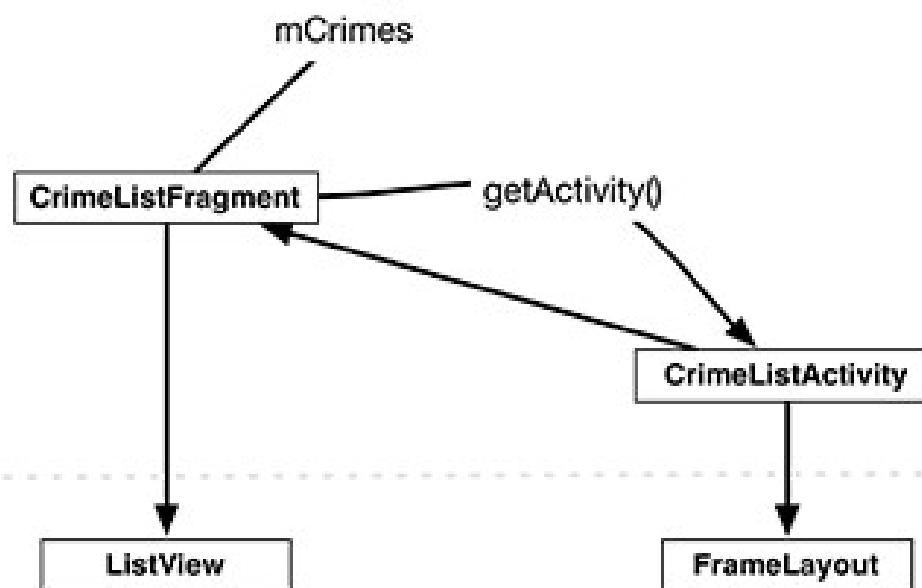


Object Diagram Part for This Lesson

Model



Controller



View

Update Model Layer: CrimeLab Singleton

A **singleton** is a class that allows only one instance of itself to be created.

A singleton exists as long as the application stays in memory, so storing the list in a singleton will keep the crime data available no matter what happens with activities, fragments, and their lifecycles.

Setting up the singleton
(CrimeLab.java)

```
public class CrimeLab {  
    private static CrimeLab sCrimeLab;  
    private Context mAppContext;  
  
    private CrimeLab(Context applicationContext) {  
        mAppContext = applicationContext;  
    }  
  
    public static CrimeLab get(Context c) {  
        if (sCrimeLab == null) {  
            sCrimeLab = new CrimeLab(c.getApplicationContext());  
        }  
        return sCrimeLab;  
    }  
}
```

Update Model Layer: CrimeLab Singleton

Setting up the ArrayList of Crime objects (CrimeLab.java)

```
public class CrimeLab {  
  
    private ArrayList<Crime> mCrimes;  
  
    private static CrimeLab sCrimeLab;  
  
    private Context mAppContext;  
  
    private CrimeLab(Context appContext) {  
        mAppContext = appContext;  
        mCrimes = new ArrayList<Crime>();  
    }  
  
    public static CrimeLab get(Context c) {  
        ...  
    }  
  
    public ArrayList<Crime> getCrimes() {  
        return mCrimes;  
    }  
  
    public Crime getCrime(UUID id) {  
        for (Crime c : mCrimes) {  
            if (c.getId().equals(id))  
                return c;  
        }  
        return null;  
    }  
}
```

Generating crimes (CrimeLab.java)

```
private CrimeLab(Context appContext) {  
  
    mAppContext = appContext;  
    mCrimes = new ArrayList<Crime>();  
    for (int i = 0; i < 100; i++) {  
        Crime c = new Crime();  
        c.setTitle("Crime #" + i);  
        c.setSolved(i % 2 == 0); // Every other one  
        mCrimes.add(c);  
    }  
}
```

Creating a ListFragment

Adding `onCreate(Bundle)` to the new fragment (`CrimeListFragment.java`)

```
public class CrimeListFragment extends ListFragment {  
  
    @Override  
  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        getActivity().setTitle(R.string.crimes_title);  
    }  
  
}
```

Adding string resource for the new activity title (`strings.xml`)

...

```
<string name="crime_solved_label">Solved?</string>  
<string name="crimes_title">Crimes</string>  
</resources>
```

Accessing crimes in `CrimeListFragment` (`CrimeListFragment.java`)

```
public class CrimeListFragment extends ListFragment {  
  
    private ArrayList<Crime> mCrimes;  
  
    @Override  
  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        getActivity().setTitle(R.string.crimes_title);  
        mCrimes = CrimeLab.get(getActivity()).getCrimes();  
    }  
  
}
```

Abstract Fragment Hosting Activity

activity_crime.xml is generic. So, rename it as activity_fragment.xml to reuse it.

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android=
    "http://schemas.android.com/apk/res/android"
    android:id="@+id/fragmentContainer"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
/>
```

Add a generic superclass
(SingleFragmentActivity.java)

```
public abstract class SingleFragmentActivity extends FragmentActivity {
    protected abstract Fragment createFragment();

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_fragment);
        FragmentManager fm = getSupportFragmentManager();
        Fragment fragment = fm.findFragmentById(R.id.fragmentContainer);

        if (fragment == null) {
            fragment = createFragment();
            fm.beginTransaction()
                .add(R.id.fragmentContainer, fragment)
                .commit();
        }
    }
}
```

CrimeListActivity and CrimeActivity Update

Implement CrimeListActivity (CrimeListActivity.java)

```
public class CrimeListActivity extends  
SingleFragmentActivity {  
  
    @Override  
    protected Fragment createFragment() {  
        return new CrimeListFragment();  
    }  
  
}
```

Clean up CrimeActivity (CrimeActivity.java)

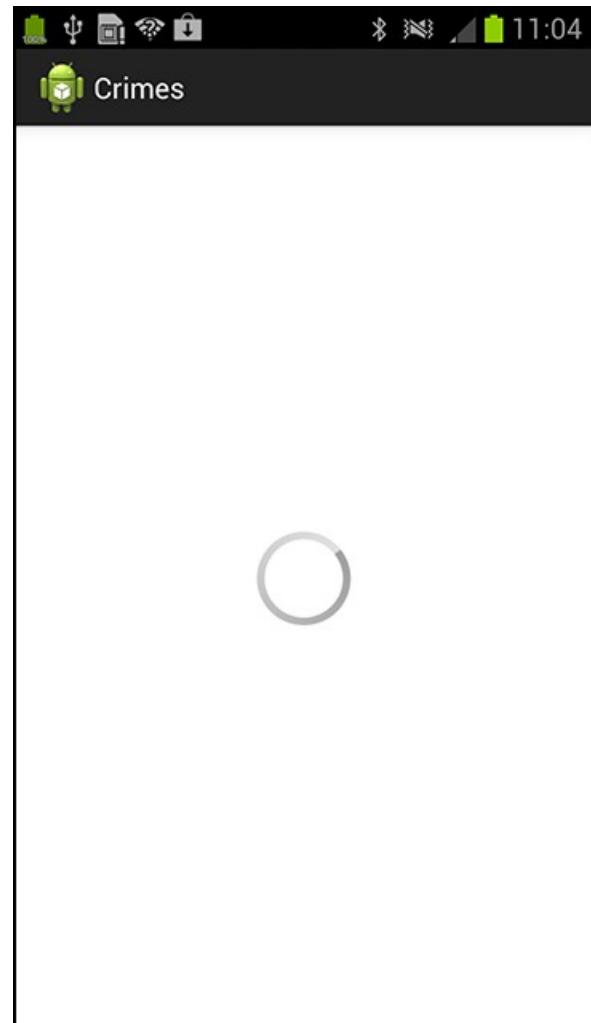
```
public class CrimeActivity extends FragmentActivity SingleFragmentActivity {  
    public void onCreate(Bundle savedInstanceState){  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_fragment);  
        FragmentManager fm = getSupportFragmentManager();  
        Fragment fragment = fm.findFragmentById(R.id.fragmentContainer);  
        if(fragment == null){  
            fragment = new CrimeFragment();  
            fm.beginTransaction()  
                .add(R.id.fragmentContainer, fragment)  
                .commit();  
        }  
    }  
    @Override  
    protected Fragment createFragment() {  
        return new CrimeFragment();  
    }  
}
```

Declare CrimeListActivity in Manifest

Declaring CrimeListActivity as the launcher activity (AndroidManifest.xml)

```
...  
<application  
    android:allowBackup="true"  
    android:icon="@drawable/ic_launcher"  
    android:label="@string/app_name"  
    android:theme="@style/AppTheme" >  
    <activity android:name=".CrimeListActivity">  
        <intent-filter>  
            <action android:name="android.intent.action.MAIN" />  
            <category android:name="android.intent.category.LAUNCHER" />  
        </intent-filter>  
    </activity>  
    <activity android:name=".CrimeActivity"  
        android:label="@string/app_name">  
        <intent-filter>  
            <action android:name="android.intent.action.MAIN" />  
            <category android:name="android.intent.category.LAUNCHER" />  
        </intent filter>  
    </activity>  
</application>  
</manifest>
```

Blank CrimeListActivity screen



ListFragment, ListView and ArrayAdapter 1/2

ListView is a subclass of ViewGroup, and each item is displayed as a child View object of the ListView.

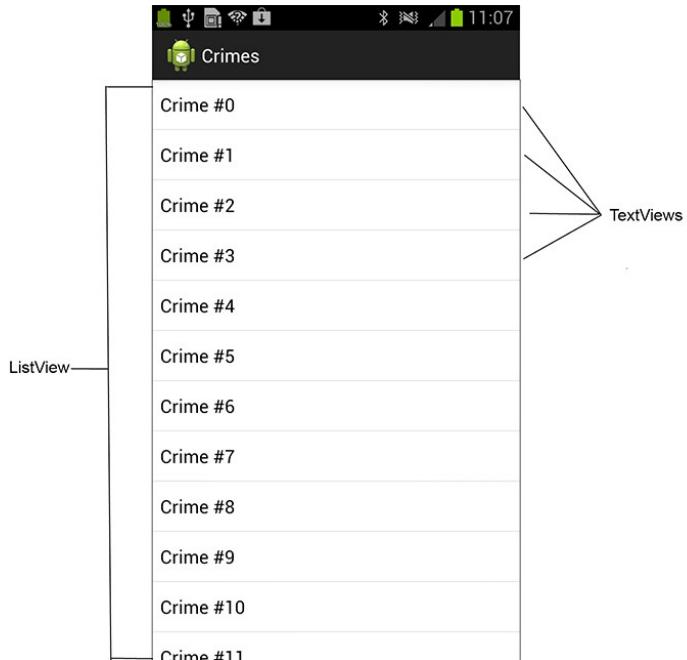
A View object only needs to exist when it is on screen. Lists can be enormous, and unnecessarily creating and storing view objects for an entire list could cause performance and memory problems.

An adapter is a controller object that sits between the ListView and the data set containing the data that the ListView should display.

The ListView asks to adapter for a view object when it needs to display a certain list item.

The adapter is responsible for

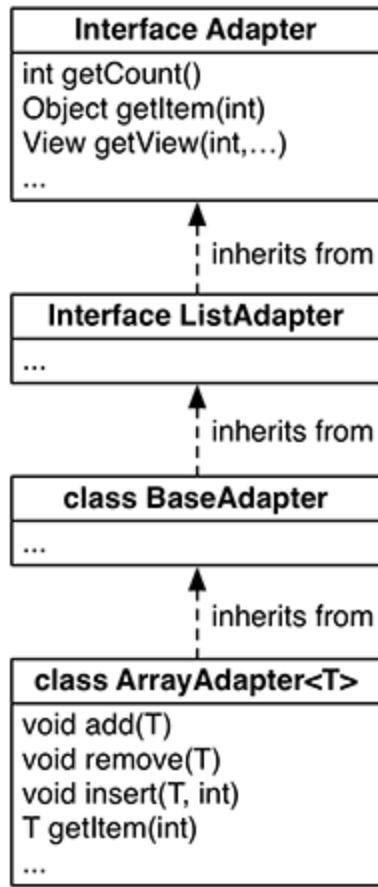
- Creating the necessary view object
- Populating it with data from the model layer
- Returning the view object to the ListView



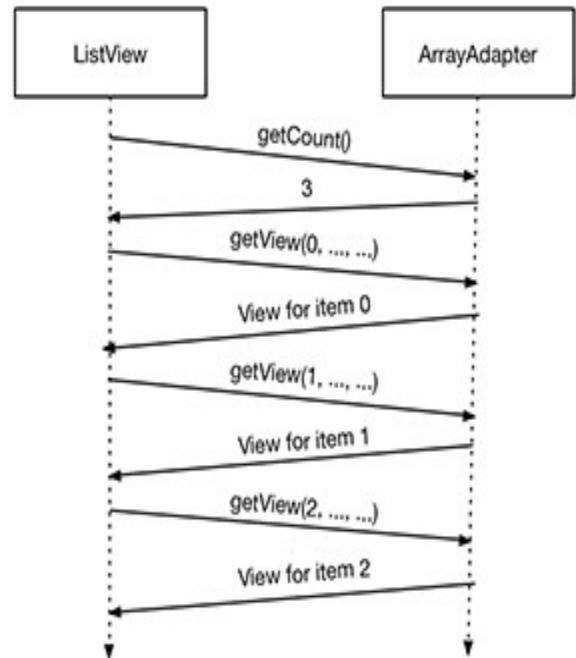
ListFragment, ListView and ArrayAdapter 2/2

An adapter is an instance of a class that implements the Adapter interface.

We will use **ArrayAdapter<T>** adapter that knows how to work with data in an array (or an ArrayList).



When the **ListView** needs a view object to display, it will have a conversation with its adapter.



Create ArrayAdapter<T>

Setting up the ArrayAdapter (CrimeListFragment.java)

```
@Override  
  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    getActivity().setTitle(R.string.crimes_title);  
    mCrimes = CrimeLab.get(getActivity()).getCrimes();  
  
    ArrayAdapter<Crime> adapter =  
        new ArrayAdapter<Crime>(getActivity(),  
            android.R.layout.simple_list_item_1,  
            mCrimes);  
  
    setListAdapter(adapter);  
}
```

Overriding Crime.toString() (Crime.java)

```
...  
  
public Crime() {  
    mId = UUID.randomUUID();  
    mDate = new Date();  
}  
  
public String toString() {  
    return mTitle;  
} ...
```

Source code for android.R.layout.simple_list_item_1

```
<TextView  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    android:id="@+id/text1"  
    style="?android:attr/listItemFirstLineStyle"  
    android:paddingTop="2dip"  
    android:paddingBottom="3dip"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content" />
```



Responding to List Item Clicks

To respond to the user touching a list item, you override another convenience method of `ListFragment`:

```
public void onListItemClick(ListView l, View v, int position, long id)
```

Whether the user “clicks” with a hardware button, a soft key, or a touch of a finger, the result still goes through `onListItemClick(...)`

The `getListAdapter()` method is a `ListFragment` convenience method that returns the adapter that is set on the `ListFragment`’s list view. You then call the adapter’s `getItem(int)` method using the position parameter of `onListItemClick(...)` and cast the result to a `Crime`.

Overriding `onListItemClick(...)` to log Crime title (`CrimeListFragment.java`)

```
public class CrimeListFragment extends ListFragment {  
  
    private static final String TAG = "CrimeListFragment";  
  
    ...  
  
    @Override  
    public void onListItemClick(ListView l, View v, int position, long id) {  
        Crime c = (Crime)(ListAdapter()).getItem(position);  
        Log.d(TAG, c.getTitle() + " was clicked");  
    }  
}
```

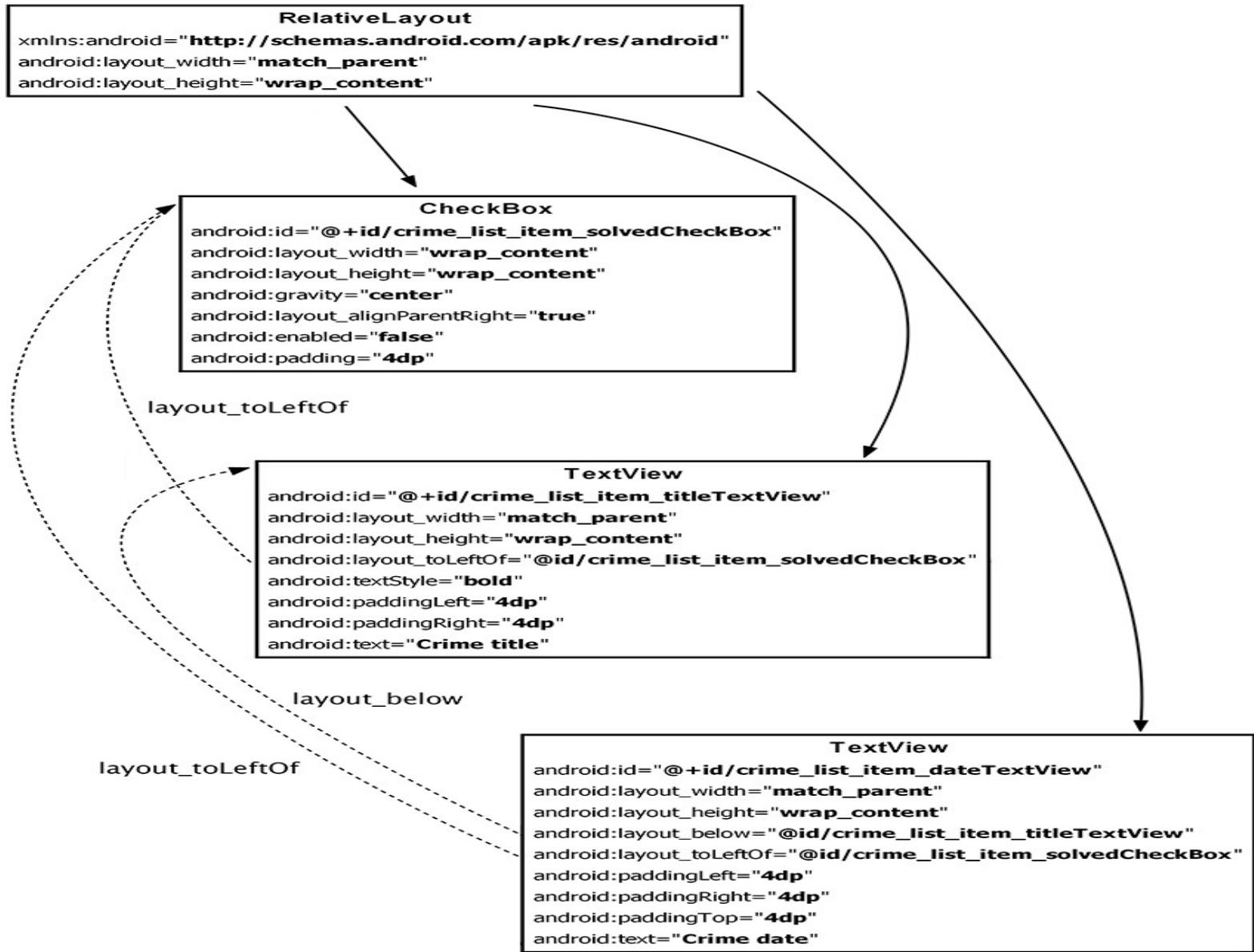
Customizing List Items

Implementing custom list items requires two things:

- Create a new layout that defines the view for the list item
- Create a subclass of `ArrayAdapter<T>` that knows how to create, populate, and return the view defined in the new layout

Crime #3	<input type="checkbox"/>
Thu Oct 18 10:30:27 EDT 2012	
Crime #4	<input checked="" type="checkbox"/>
Thu Oct 18 10:30:27 EDT 2012	
Crime #5	<input type="checkbox"/>
Thu Oct 18 10:30:27 EDT 2012	
Crime #6	<input checked="" type="checkbox"/>
Thu Oct 18 10:30:27 EDT 2012	

Custom list item layout (list_item_crime.xml)



Create an Adapter Subclass 1/2

Adding custom adapter as inner class (CrimeListFragment.java)

```
public void onListItemClick(ListView l, View v,  
    int position, long id) {  
    Crime c = (Crime)(getListAdapter()).getItem(position);  
    Log.d(TAG, c.getTitle() + " was clicked");  
}  
  
private class CrimeAdapter extends  
    ArrayAdapter<Crime> {  
    public CrimeAdapter(ArrayList<Crime> crimes) {  
        super(getActivity(), 0, crimes);  
    }  
}  
}
```

Using a CrimeAdapter (CrimeListFragment.java)

```
ArrayAdapter<Crime> adapter =  
    new ArrayAdapter<Crime>(this,  
        android.R.layout.simple_list_item_1,  
        mCrimes);  
CrimeAdapter adapter = new CrimeAdapter(mCrimes);  
setListAdapter(adapter);  
}  
  
public void onListItemClick(ListView l,  
    View v, int position, long id) {  
    Crime c = (Crime)(getListAdapter()).getItem(position);  
    Crime c =  
        ((CrimeAdapter)getListAdapter()).getItem(position);  
    Log.d(TAG, c.getTitle() + " was clicked");  
}
```

Creating an adapter subclass 2/2

Overriding getView(...) (CrimeListFragment.java)

```
private class CrimeAdapter extends ArrayAdapter<Crime> {  
  
    public CrimeAdapter(ArrayList<Crime> crimes) {  
        super(getActivity(), 0, crimes);  
    }  
  
}
```

```
@Override  
public View getView(int position, View convertView,  
    ViewGroup parent) {  
    // If we weren't given a view, inflate one  
    if (convertView == null) {  
        convertView = getActivity().getLayoutInflater()  
            .inflate(R.layout.list_item_crime, null);  
    }  
}
```

```
// Configure the view for this Crime  
Crime c = getItem(position);  
TextView titleTextView =  
  
(TextView)convertView.findViewById(R.id.crime_list_item_titleTextView);  
titleTextView.setText(c.getTitle());  
TextView dateTextView =  
  
(TextView)convertView.findViewById(R.id.crime_list_item_dateTextView);  
dateTextView.setText(c.getDate().toString());  
CheckBox solvedCheckBox = (CheckBox)convertView  
    .findViewById(R.id.crime_list_item_solvedCheckBox);  
solvedCheckBox.setChecked(c.isSolved());  
  
return convertView;  
}  
}
```

Making the CheckBox non-focusable

(list_item_crime.xml)

```
...  
<CheckBox  
    android:id="@+id/crime_list_item_solvedCheckBox"  
    android:layout_width="wrap_content"  
    android:layout_height="match_parent"  
    android:gravity="center"  
    android:layout_alignParentRight="true"  
    android:enabled="false"  
    android:focusable="false"  
    android:padding="4dp" />  
...
```

Now with Custom List Items



Android UI Use Fragment Arguments

Javed Hasan
BJIT Limited

Lesson Content

CriminalIntent New Features Overview

- Make List and Details of the App working together

Start an Activity from Fragment

- Put and retrieve Intent extra, update view with crime data
- The downside to direct retrieval

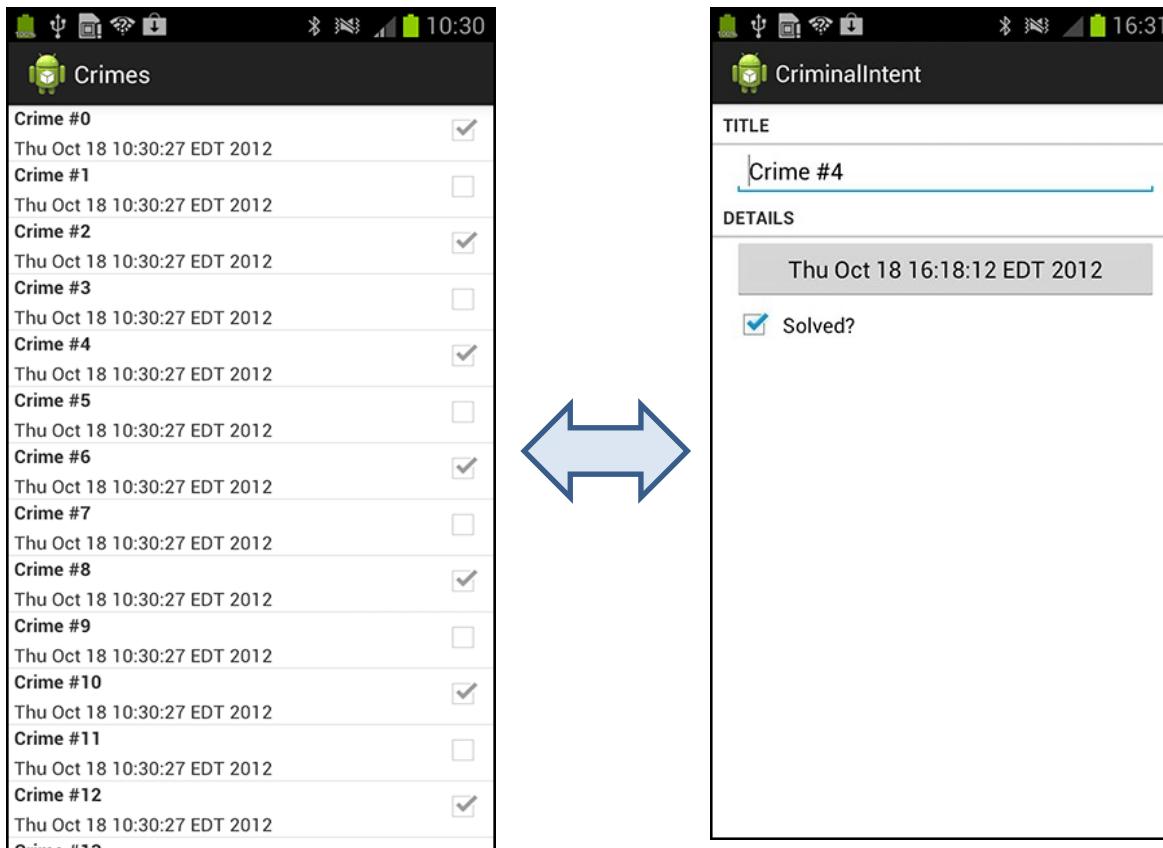
Fragment Arguments

- Attaching arguments to a fragment and retrieving it

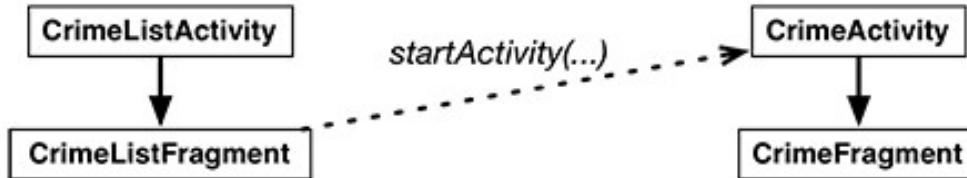
Reload the List with New data

New Feature: List and Detail View Work Together

You will get the list and the detail parts of CriminalIntent working together. When a user clicks on an item in the list of crimes, a new CrimeActivity hosting a CrimeFragment will appear and display the details for a particular instance of Crime.



Starting an Activity from a Fragment



Starting CrimeActivity (CrimeListFragment.java)

```
public void onListItemClick(ListView l, View v, int position, long id) {  
    // Get the Crime from the adapter  
    Crime c = ((CrimeAdapter)getListAdapter()).getItem(position);  
    Log.d(TAG, c.getTitle() + " was clicked");  
  
    // Start CrimeActivity  
    Intent i = new Intent(getActivity(), CrimeActivity.class);  
    startActivity(i);  
}
```

Put and Retrieve Intent Extra

Starting CrimeActivity with an extra (CrimeListFragment.java)

```
public void onListItemClick(ListView l, View v, int position, long id) {  
    // Get the Crime from the adapter  
    Crime c = ((CrimeAdapter) getListAdapter()).getItem(position);  
  
    // Start CrimeActivity  
    Intent i = new Intent(getActivity(), CrimeActivity.class);  
    i.putExtra(CrimeFragment.EXTRA_CRIME_ID, c.getId());  
    startActivity(i);  
}
```

Retrieving the extra and fetching the Crime (CrimeFragment.java)

```
public class CrimeFragment extends Fragment {  
    public static final String EXTRA_CRIME_ID =  
        "com.bignerdranch.android.criminalintent.crime_id";  
    private Crime mCrime;  
    ...  
  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        mCrime = new Crime();  
        UUID crimeId = (UUID) getActivity().getIntent()  
            .getSerializableExtra(EXTRA_CRIME_ID);  
        mCrime = CrimeLab.get(getActivity()).getCrime(crimeId);  
    }  
}
```

Updating CrimeFragment's View with Crime Data

Updating view objects (CrimeFragment.java)

```
@Override
```

```
public View onCreateView(LayoutInflater inflater, ViewGroup parent,
```

```
    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() {
```

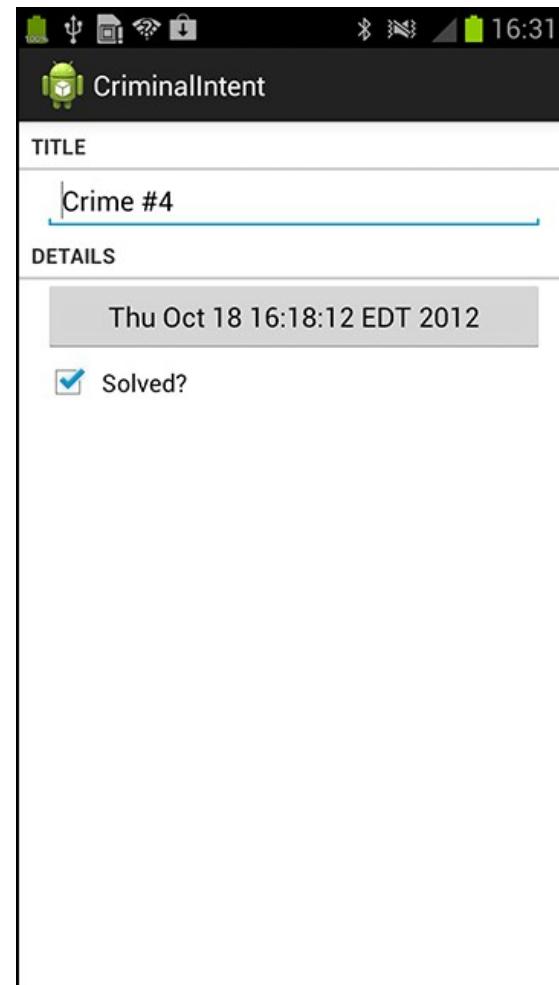
```
...
```

```
});
```

```
...
```

```
return v;
```

```
}
```



The Downside to Direct Retrieval

Having the fragment access the intent that belongs to the hosting activity makes for simple code. However, it costs you the encapsulation of your fragment. **CrimeFragment** is no longer a reusable building block because it expects that it will always be hosted by an activity whose Intent defines an extra named **EXTRA_CRIME_ID**.

This may be a reasonable expectation on **CrimeFragment's** part, but it means that CrimeFragment, as currently written, cannot be used with just any activity.

A better solution is to stash the mCrimId someplace that belongs to **CrimeFragment** rather than keeping it in CrimeActivity's personal space. The **CrimeFragment** could then retrieve this data without relying on the presence of a particular extra in the activity's intent. The "someplace" that belongs to a fragment is known as its **arguments** bundle.

Fragment Arguments

Every fragment instance can have a Bundle object attached to it. This bundle contains key-value pairs that work just like the intent extras of an Activity. Each pair is known as an **argument**.

To attach the arguments bundle to a fragment, you call **Fragment.setArguments(Bundle)**. Attaching arguments to a fragment must be done after the fragment is created but before it is added to an activity.

Android programmers follow a convention of adding a static method named **newInstance()** to the **Fragment** class. This method creates the fragment instance and bundles up and sets its arguments.

Writing a **newInstance(UUID)** method
(CrimeFragment.java)

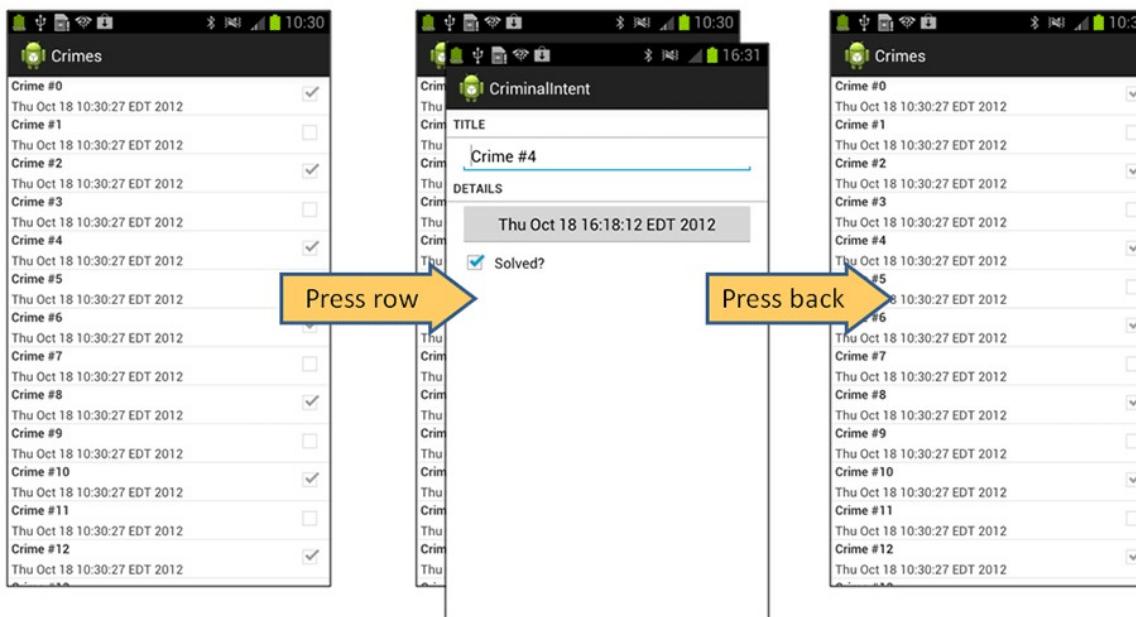
```
public static CrimeFragment newInstance(UUID crimeId) {  
    Bundle args = new Bundle();  
    args.putSerializable(EXTRA_CRIME_ID, crimeId);  
    CrimeFragment fragment = new CrimeFragment();  
    fragment.setArguments(args);  
    return fragment;  
}
```

Using **newInstance(UUID)**
(CrimeActivity.java)

```
@Override  
protected Fragment createFragment() {  
    return new CrimeFragment();  
    UUID crimeId = (UUID) getIntent()  
        .getSerializableExtra(CrimeFragment.EXTRA_CRIME_ID);  
    return CrimeFragment.newInstance(crimeId);  
}
```

Reloading the List 1/2

When you changes some crime's data, list view's adapter needs to be informed that the data set has changed so that it can refetch the data and reload the list. You can work with the ActivityManager's back stack to reload the list at the right moment.



When the **CrimeListActivity** is resumed, it receives a call to **onResume()** from the OS. When **CrimeListActivity** receives this call, its **FragmentManager** calls **onResume()** on the **CrimeListFragment** fragments.

In **CrimeListFragment**, override **onResume()** to reload the list.

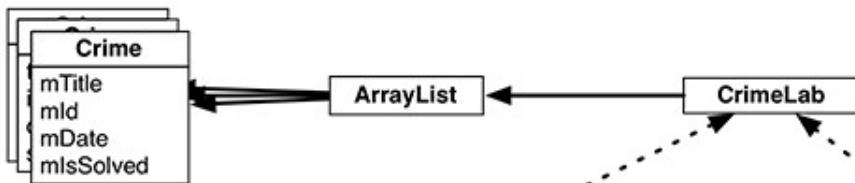
Reloading the List 2/2

Reloading the list in onResume()
(CrimeListFragment.java)

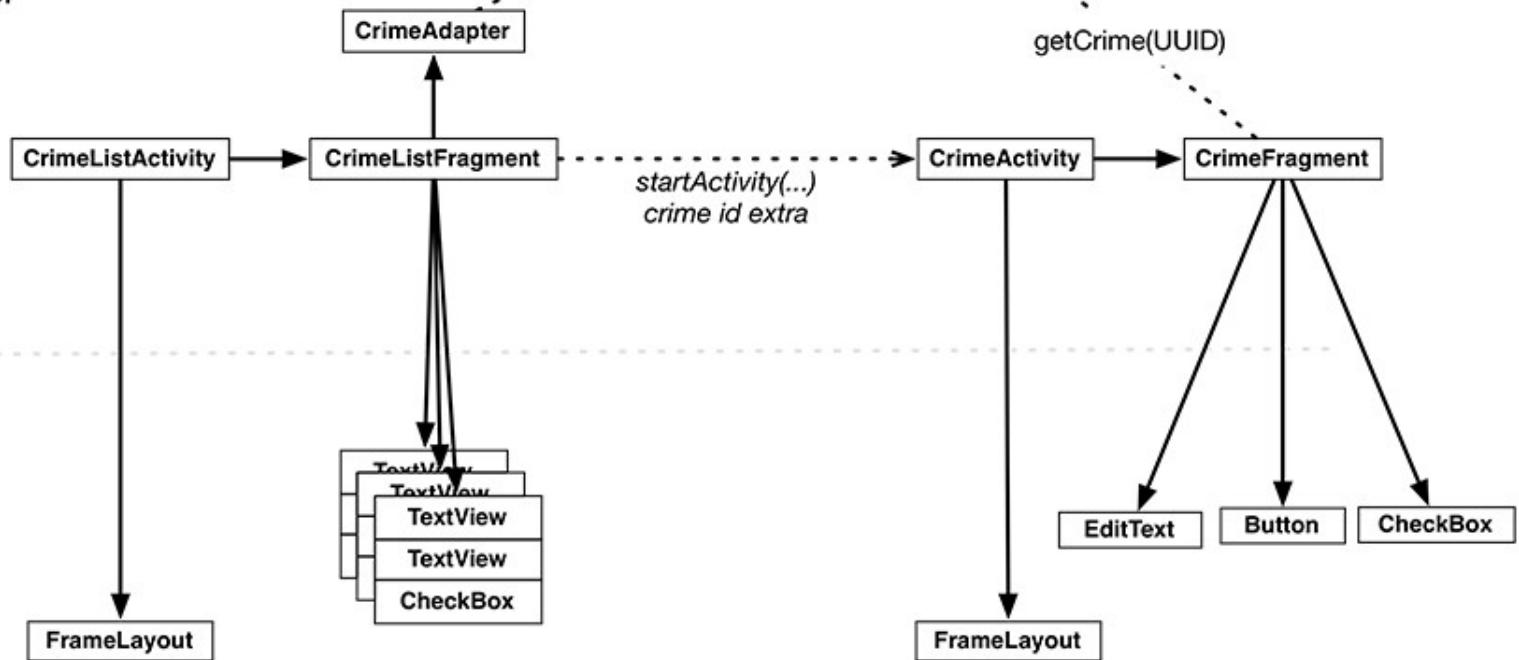
```
@Override  
public void onResume() {  
    super.onResume();  
    ((CrimeAdapter) getListAdapter()).notifyDataSetChanged();  
}
```

Updated Object Diagram

Model



Controller



Android UI Using View Pager

Javed Hasan
BJIT Limited

Lesson Content

CriminalIntent New Features Overview

- Show Crime details view using ViewPager

Object Diagram of CriminalIntent

Setup ViewPager in CrimePagerActivity

- Wire up the ViewPager & its Pager Adapter in CrimePagerActivity

Integrate CrimePagerActivity with CrimeListFragment

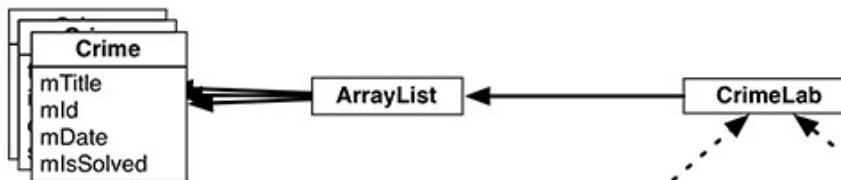
New Feature: Use ViewPager to Show Crime Details

You will create a new activity to host CrimeFragment. This activity's layout will consist of an instance of **ViewPager**. Adding a ViewPager to your UI lets users navigate between list items by swiping across the screen to “page” forward or backward through the crimes..

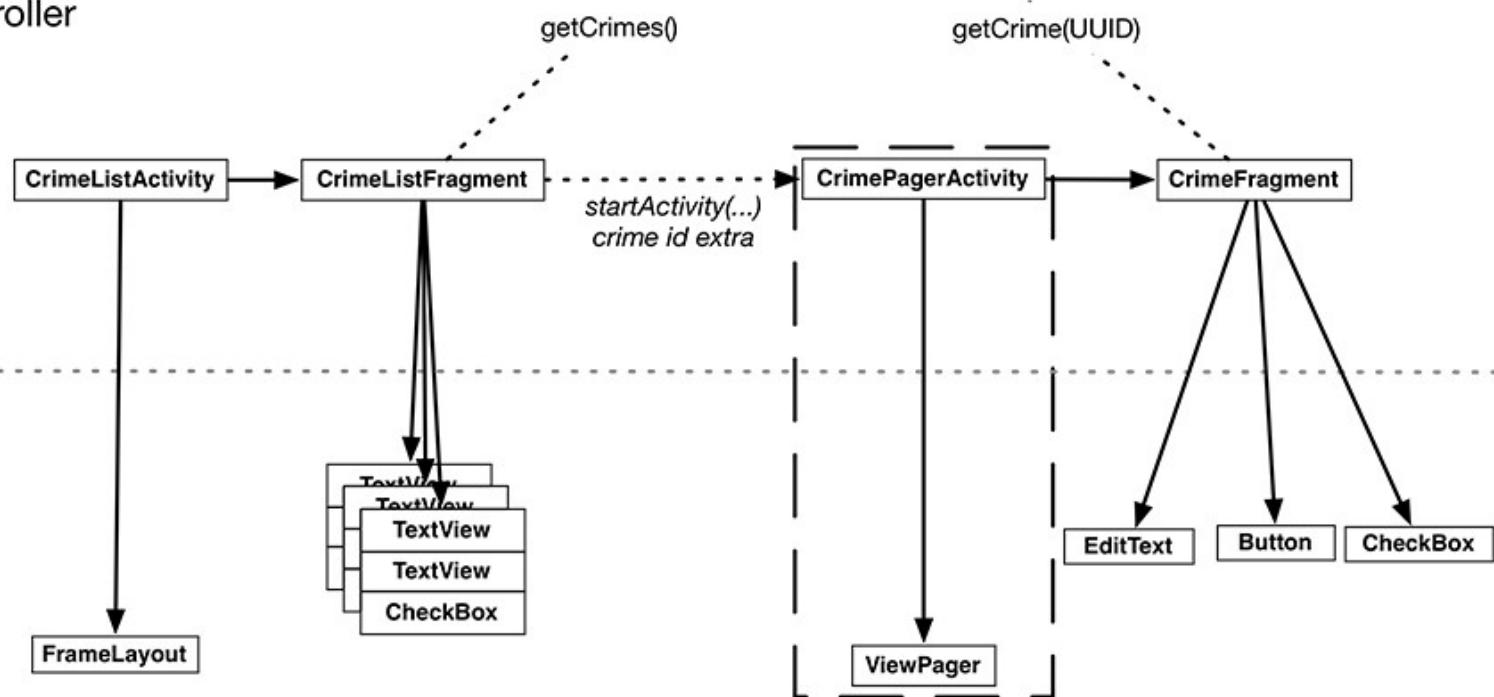


Object Diagram of CriminalIntent

Model



Controller



View

Setup ViewPager in CrimePagerActivity Programmatically

Step 1. Create Standalone ID resources for ViewPager(res/values/ids.xml)

```
<?xml version="1.0" encoding="utf-8"?>  
<resources>  
  
    <item type="id" name="viewPager" />  
  
</resources>
```

Step 2. Create ViewPager as Content View
Programmatically Of CrimePagerActivity
(CrimePagerActivity.java)

```
@Override  
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    mViewPager = new ViewPager(this);  
    mViewPager.setId(R.id.viewPager);  
    setContentView(mViewPager);  
}
```

ViewPager and PagerAdapter

A **ViewPager** is like an **AdapterView** (the superclass of **ListView**). A **ViewPager** requires a **FragmentStatePagerAdapter** to get views from it.

ViewPager and **FragmentStatePagerAdapter** communicates using 2 methods: **getCount()** and **getItem(int)**.

When your **getItem(int)** method is called for a position in your array of crimes, you will return a **CrimeFragment** configured to display the crime at that position.

This **PagerAdapter** is adding the fragments you return to your activity and helping ViewPager identify the fragments' views so that they can be placed correctly.

Set up pager adapter (**CrimePagerActivity.java**)

```
public class CrimePagerActivity extends FragmentActivity {  
    private ViewPager mViewPager;  
    private ArrayList<Crime> mCrimes;  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        mViewPager = new ViewPager(this);  
        mViewPager.setId(R.id.viewPager);  
        setContentView(mViewPager);  
        mCrimes = CrimeLab.get(this).getCrimes();  
        FragmentManager fm = getSupportFragmentManager();  
        mViewPager.setAdapter(new FragmentStatePagerAdapter(fm) {  
            @Override  
            public int getCount() {  
                return mCrimes.size();  
            }  
            @Override  
            public Fragment getItem(int pos) {  
                Crime crime = mCrimes.get(pos);  
                return CrimeFragment.newInstance(crime.getId());  
            }  
        });  
    }  
}
```

Integrate CrimePagerActivity 1/2

Fire it up (CrimeListFragment.java)

```
@Override  
public void onListItemClick(ListView l, View v, int position,  
long id) {  
    // Get the Crime from the adapter  
    Crime c =  
        ((CrimeAdapter) getListAdapter()).getItem(position);  
    // Start CrimeActivity  
    Intent i = new Intent(getActivity(), CrimeActivity.class);  
    // Start CrimePagerActivity with this crime  
    Intent i = new Intent(getActivity(),  
        CrimePagerActivity.class);  
    i.putExtra(CrimeFragment.EXTRA_CRIME_ID, c.getId());  
    startActivity(i);  
}
```

Add CrimePagerActivity to manifest (AndroidManifest.xml)

```
<?xml version="1.0" encoding="utf-8"?>  
<manifest ...>  
    ...  
    <application ...>  
        ...  
        <activity  
            android:name=".CrimeActivity"  
            android:label="@string/app_name">  
        </activity>  
        <activity android:name=".CrimePagerActivity"  
            android:label="@string/app_name">  
        </activity>  
    </application>  
</manifest>
```

Integrate CrimePagerActivity 2/2

By default, the ViewPager shows the first item in its PagerAdapter. You can have it show the crime that was selected by setting the ViewPager's current item to the index of the selected crime.

At the end of
CrimePagerActivity.onCreate(...), find the index of the crime to display by looping through and checking each crime's ID.

Set initial pager item (CrimePagerActivity.java)

```
public class CrimePagerActivity extends FragmentActivity {  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        ...  
        FragmentManager fm = getSupportFragmentManager();  
        mViewPager.setAdapter(new FragmentStatePagerAdapter(fm) {  
            ...  
        });  
        UUID crimeId = (UUID) getIntent()  
            .getSerializableExtra(CrimeFragment.EXTRA_CRIME_ID);  
        for (int i = 0; i < mCrimes.size(); i++) {  
            if (mCrimes.get(i).getId().equals(crimeId)) {  
                mViewPager.setCurrentItem(i);  
                break;  
            }  
        }  
    }  
}
```

ViewPager OnPageListener Interface

You can replace the activity's title that appears on the action bar (or the title bar on older devices) with the title of the current Crime using **ViewPager.OnPageChangeListener** interface.

OnPageChangeListener is how you listen for changes in the page currently being displayed by ViewPager. When the page changes, you set **CrimePagerActivity**'s title to the title of the Crime.

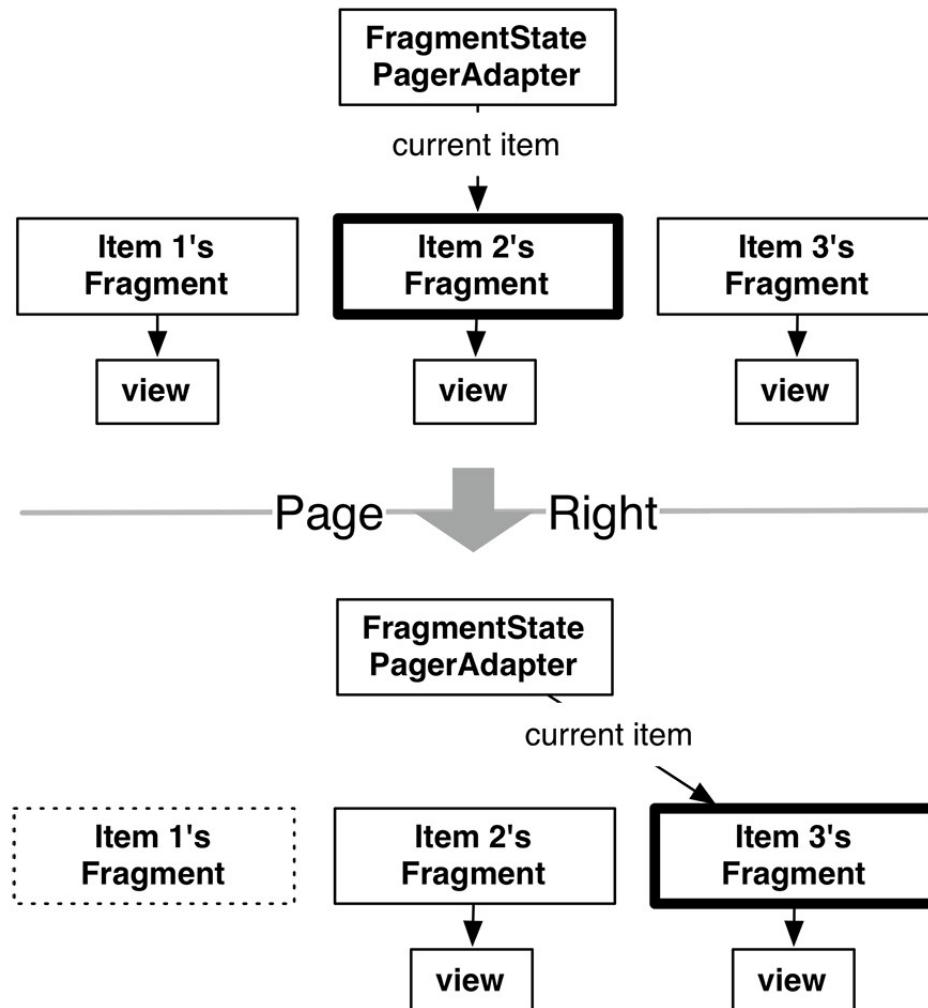
The **onPageScrolled(...)** method tells you exactly where your page is going to be, **onPageScrollStateChanged(...)** tells you whether the page animation is being actively dragged, settling to a steady state, or idling.

```
@Override  
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    mViewPager = new ViewPager(this);  
    mViewPager.setId(R.id.viewPager);  
    setContentView(mViewPager);  
    mCrimes = CrimeLab.get(this).getCrimes();  
    FragmentManager fm = getSupportFragmentManager();  
    mViewPager.setAdapter(new FragmentStatePagerAdapter(fm)  
    {...});  
    mViewPager.setOnPageChangeListener(new  
    ViewPager.OnPageChangeListener() {  
        public void onPageScrolled(int state) {}  
        public void onPageScrolled(int pos, float posOffset, int posOffsetPixels) {}  
        public void onPageSelected(int pos) {  
            Crime crime = mCrimes.get(pos);  
            if (crime.getTitle() != null) {  
                setTitle(crime.getTitle());  
            }  
        }  
    });  
    ...  
}
```

FragmentStatePagerAdapter vs. Fragment PagerAdapter

FragmentPagerAdapter is used exactly like **FragmentStatePagerAdapter**. It only differs in how it unloads your fragments when they are no longer needed.

With **FragmentStatePagerAdapter**, your unneeded fragment is destroyed. A transaction is committed to completely remove the fragment from your activity's **FragmentManager**.



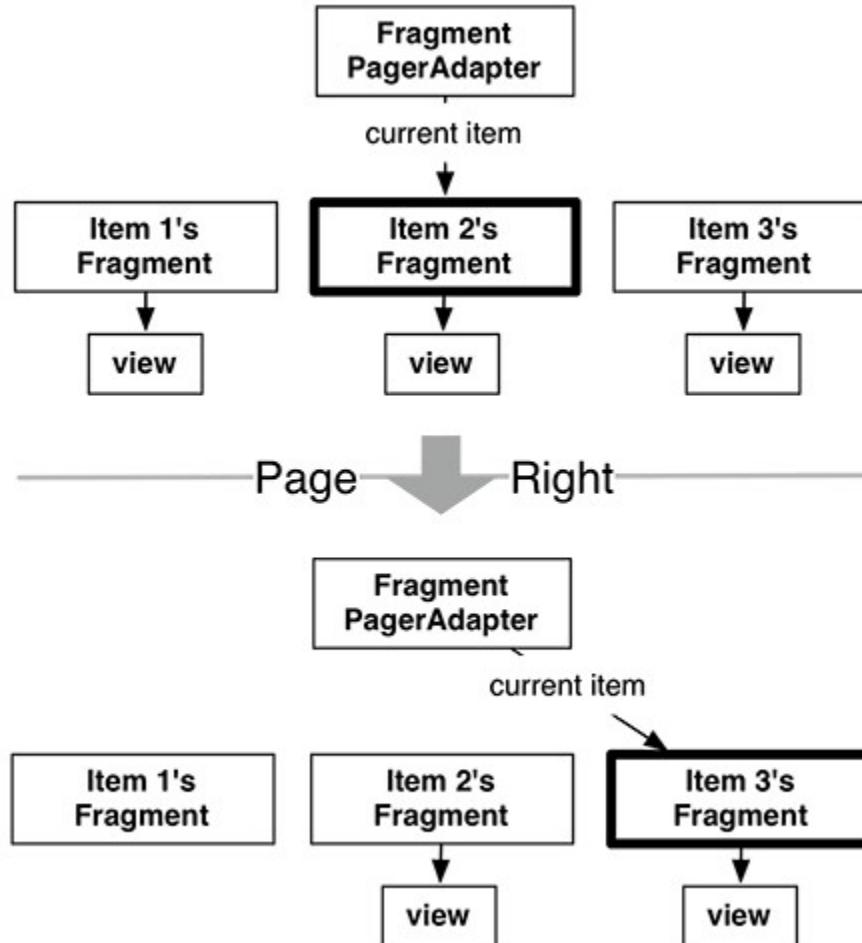
FragmentStatePagerAdapter vs. Fragment PagerAdapter

By comparison,

FragmentPagerAdapter does nothing of the kind. When your fragment is no longer needed, FragmentPagerAdapter calls `detach(Fragment)` on the transaction instead of `remove(Fragment)`. This destroys the fragment's view, but leaves the fragment instance alive in the FragmentManager.

FragmentStatePagerAdapter is suitable for long list of items, as it manages memory more efficiently.

FragmentStatePagerAdapter is suitable when your interface has small and fixed number of fragments, say, tabbed interface.



Android UI Dialogs

Javed Hasan
BJIT Limited

Lesson Content

CriminalIntent New Features and Design Overview

- User can change the date of a crime
- Object Diagram of CriminalIntent

Dialogs, Alert Dialogs

- Basics, Example

Dialog Fragment

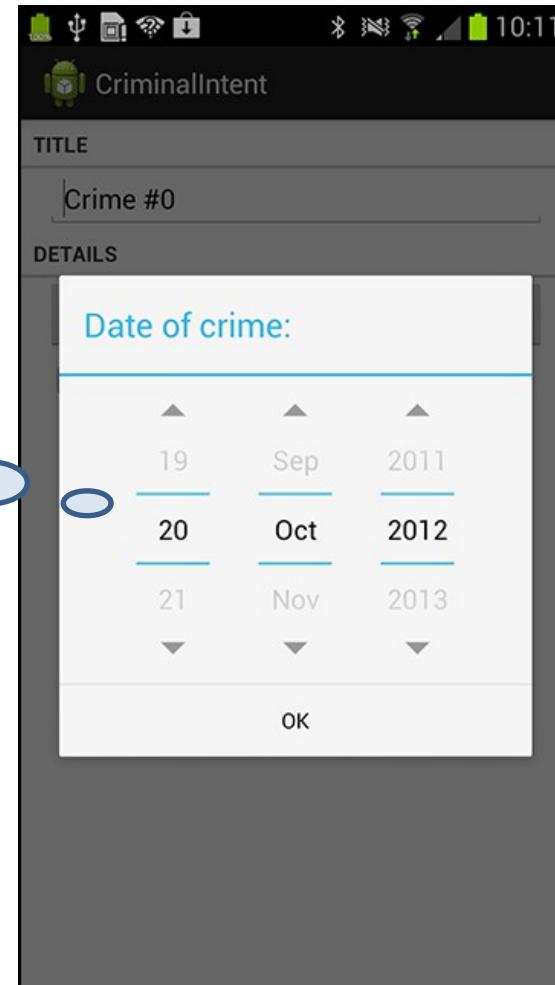
- Create, Show and Setup Content

Pass Data between Two Fragments

- Pass and Return data between two fragments
- Flexibility in Presenting DialogFragment

New Feature: Change Date of a Crime

Alert Dialog. User can change the date of a crime

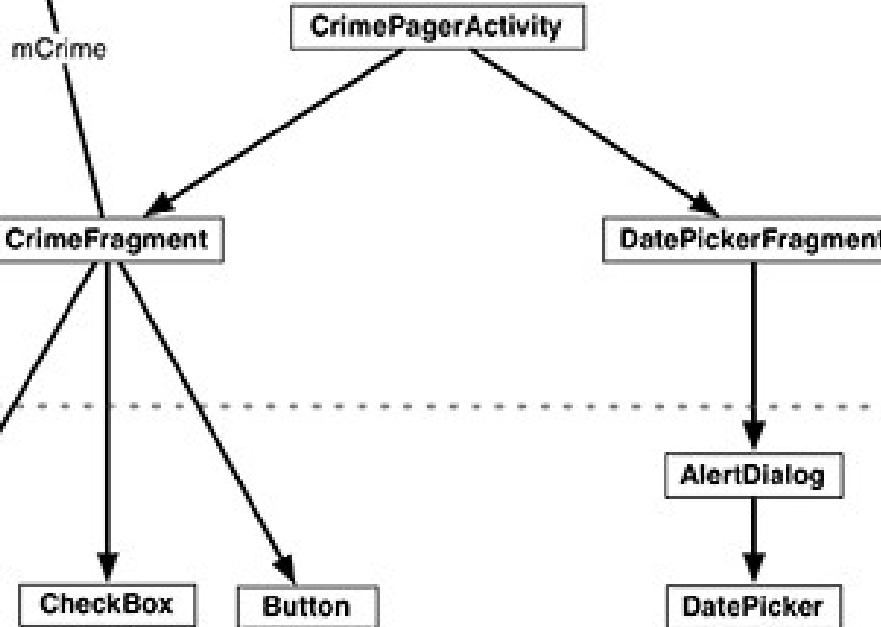


Object Diagram of CrimePagerActivity with Dialog Fragment and Crime Fragment

Model



Controller



View

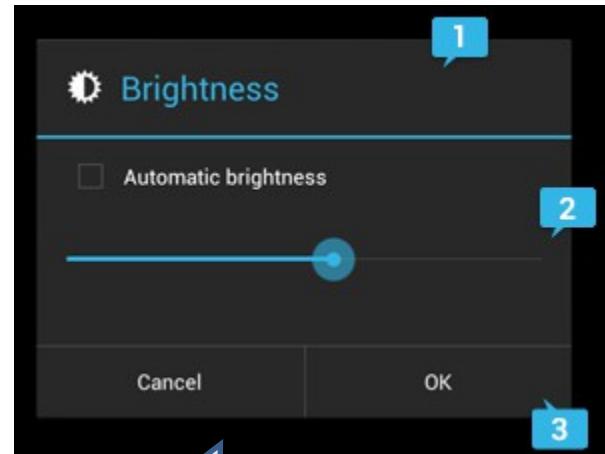
Basics of Dialog and Alert Dialog

A **dialog** is a small window that prompts the user to make a decision or enter additional information.

An **AlertDialog** is a dialog that can show a title, up to three buttons, a list of selectable items, or a custom layout.

You should use a **DialogFragment** as a container for your dialog. The DialogFragment class provides all the controls you need to create your dialog and manage its appearance.

Using DialogFragment to manage the dialog ensures that it correctly handles lifecycle events such as when the user presses the Back button or rotates the screen. The DialogFragment class also allows you to reuse the dialog's UI as an embeddable component in a larger UI, just like a traditional Fragment.



Alert Dialog

- 1.Title: Optional.
- 2.Content Area: Display a message, a list or other custom layout
- 3.Action Buttons: Should be no more than three action buttons

Create a DialogFragment

Create a new class named **DatePickerFragment** and make **DialogFragment** its super class.

DialogFragment includes the following method:

```
public Dialog onCreateDialog(Bundle savedInstanceState)
```

The FragmentManager of the hosting activity calls this method as part of putting the DialogFragment on screen.

Creating a DialogFragment
(DatePickerFragment.java)

```
public class DatePickerFragment extends DialogFragment {  
    @Override  
    public Dialog onCreateDialog(Bundle savedInstanceState) {  
        return new AlertDialog.Builder(getActivity())  
            .setTitle(R.string.date_picker_title)  
            .setPositiveButton(android.R.string.ok, null)  
            .create();  
    }  
}
```

Show Dialog Content 1/2

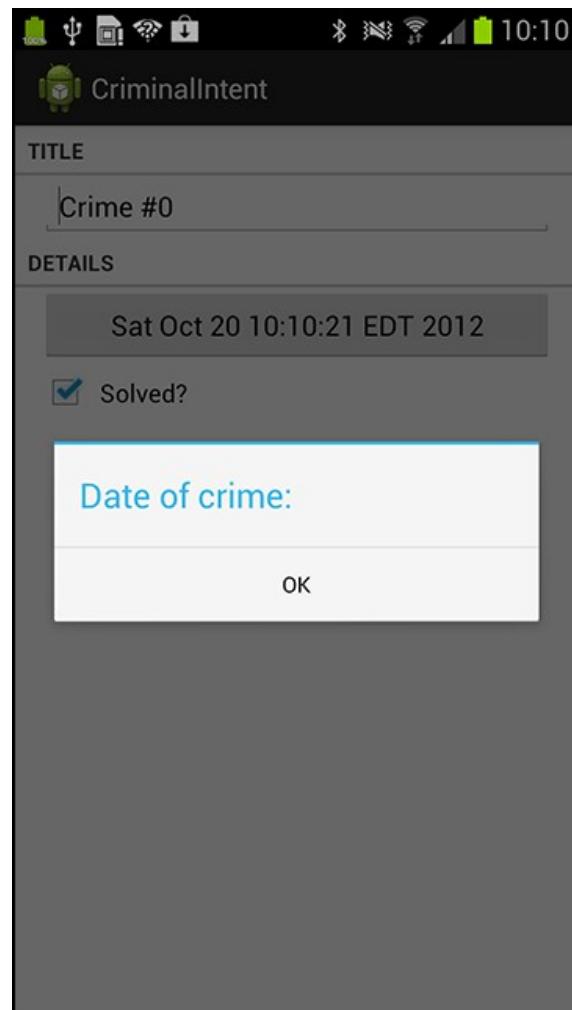
Like all fragments, instances of **DialogFragment** are managed by the **FragmentManager** of the hosting activity.

To show a DialogFragment, use below method:

```
public void show(FragmentManager manager, String tag)
```

The **string parameter** uniquely identifies the DialogFragment in the FragmentManager's list.

In **CrimeFragment**, add a constant for the **DatePickerFragment's** tag. Then, in `onCreateView(...)`, remove the code that disables the date button and set a `View.OnClickListener` that shows a **DatePickerFragment** when the date button is pressed.



Show Dialog Content 1/2

Show your DialogFragment (CrimeFragment.java)

```
public class CrimeFragment extends Fragment {  
    public static final String EXTRA_CRIME_ID =  
        "com.bignerdranch.android.criminalintent.crime_id";  
    private static final String DIALOG_DATE = "date";  
    ...  
    @Override  
    public View onCreateView(LayoutInflater inflater,  
        ViewGroup parent,  
        Bundle savedInstanceState){  
        ...  
        mDateButton =  
            (Button)v.findViewById(R.id.crime_date);  
        mDateButton.setText(mCrime.getDate().toString());  
        mDateButton.setEnabled(false);  
    }
```

```
mDateButton.setOnClickListener(new  
    View.OnClickListener() {  
        public void onClick(View v) {  
            FragmentManager fm = getActivity()  
                .getSupportFragmentManager();  
            DatePickerFragment dialog = new  
            DatePickerFragment();  
            dialog.show(fm, DIALOG_DATE);  
        }  
    });  
    mSolvedCheckBox =  
        (CheckBox)v.findViewById(R.id.crime_solved);  
    ...  
    return v;  
}
```

Setup Dialog Content

You can add DatePicker widget using AlertDialog.Builder method:

```
public AlertDialog.Builder.setView(View view)
```

This method configures the dialog to display the passed-in View object between the dialog's title and its button(s).



DatePicker layout (layout/dialog_date.xml)

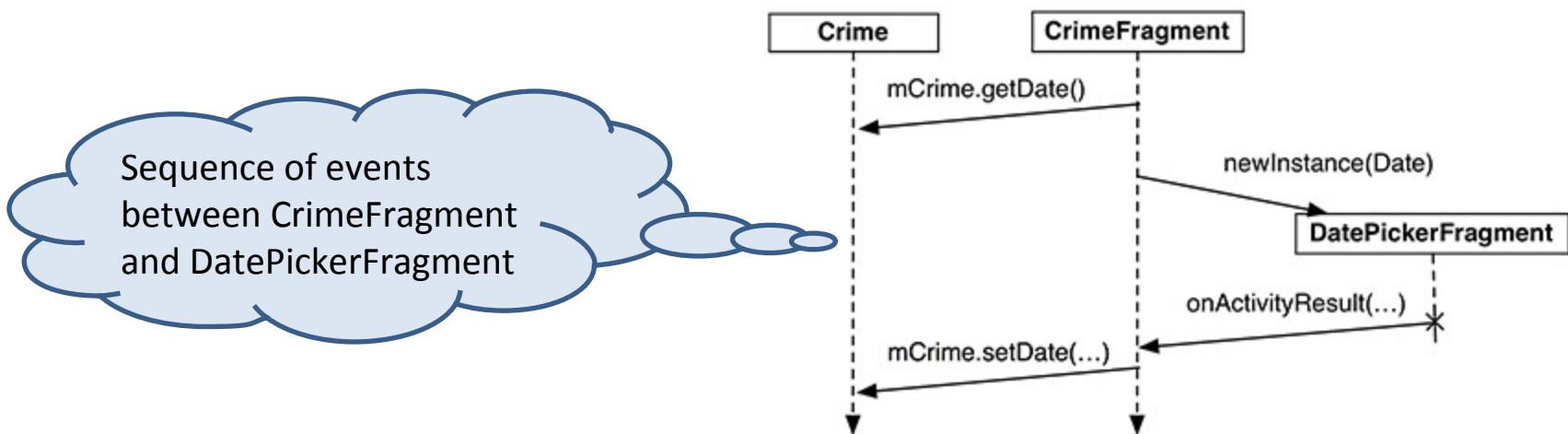
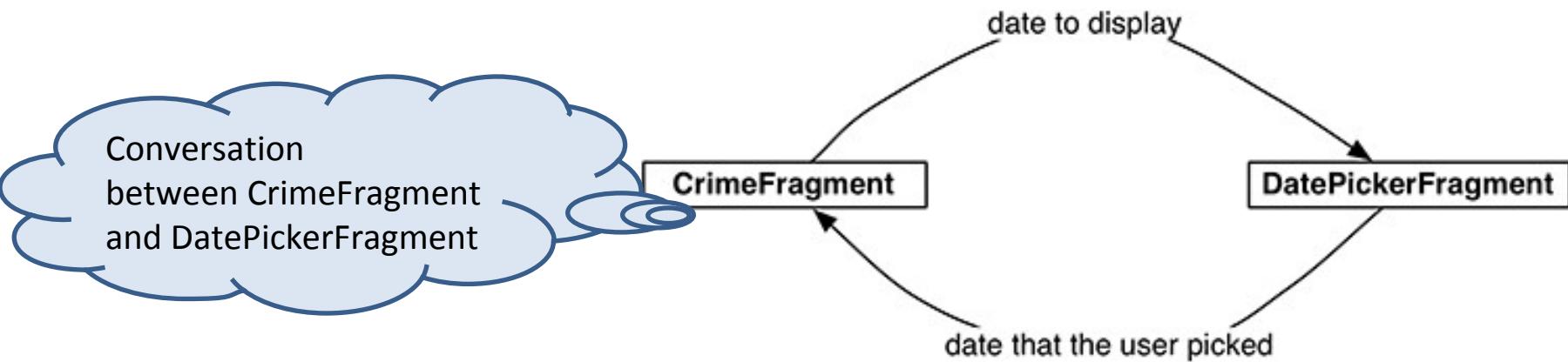
```
DatePicker
xmlns:android="http://schemas.android.com/apk/res/android"
android:id="@+id/dialog_date_picker"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:calendarViewShown="false"
```

Add DatePicker to AlertDialog
(DatePickerFragment.java)

```
@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    View v = getActivity().getLayoutInflater()
        .inflate(R.layout.dialog_date, null);
    return new AlertDialog.Builder(getActivity())
        .setView(v)
        .setTitle(R.string.date_picker_title)
        .setPositiveButton(android.R.string.ok, null)
        .create();
}
```

Pass Data Between Two Fragments

You need to pass data between two fragments that are hosted by the same activity
– CrimeFragment and DatePickerFragment.



Passing Data to DatePickerFragment 1/3

To get data into your **DatePickerFragment**, you are going to stash the date in **DatePickerFragment**'s arguments **bundle**, where the **DatePickerFragment** can access it.

Creating and setting fragment arguments is typically done in a `newInstance()` method that replaces the fragment constructor. In `DatePickerFragment.java`, add a `newInstance(Date)` method.

Adding a `newInstance(Date)` method (`DatePickerFragment.java`)

```
public class DatePickerFragment extends DialogFragment {  
    public static final String EXTRA_DATE =  
        "com.bignerdranch.android.criminalintent.date";  
  
    private Date mDate;  
  
    public static DatePickerFragment newInstance(Date date) {  
        Bundle args = new Bundle();  
        args.putSerializable(EXTRA_DATE, date);  
  
        DatePickerFragment fragment = new DatePickerFragment();  
        fragment.setArguments(args);  
  
        return fragment;  
    }  
  
    ...  
}
```

Passing Data to DatePickerFragment 2/3

In **CrimeFragment**, remove the call to the DatePickerFragment constructor and replace it with a call to
DatePickerFragment.newInstance(Date).

Add call to newInstance()
(CrimeFragment.java)

```
@Override
public View onCreateView(LayoutInflater inflater,
    ViewGroup parent, Bundle savedInstanceState) {
    ...
    mDateButton = (Button)v.findViewById(R.id.crime_date);
    mDateButton.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            FragmentManager fm = getActivity()
                .getSupportFragmentManager();
            DatePickerFragment dialog = new DatePickerFragment();
            dialog.show(fm, DIALOG_DATE);
        }
    });
    return v;
}
```

Passing Data to DatePickerFragment 3/3

DatePickerFragment needs to initialize the DatePicker using the information held in the Date. However, initializing the DatePicker requires integers for the month, day, and year. Date is more of a timestamp and cannot provide integers like this directly.

To get the integers you need, you must create a **Calendar** object and use the Date to configure the Calendar. Then you can retrieve the required information from the Calendar.

In `onCreateDialog(...)`, get the Date from the arguments and use it and a Calendar to initialize the DatePicker.

Date extraction and DatePicker initialization
(`DatePickerFragment.java`)

```
@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    mDate = (Date) getArguments().getSerializable(EXTRA_DATE);
    // Create a Calendar to get the year, month, and day
    Calendar calendar = Calendar.getInstance();
    calendar.setTime(mDate);
    int year = calendar.get(Calendar.YEAR);
    int month = calendar.get(Calendar.MONTH);
    int day = calendar.get(Calendar.DAY_OF_MONTH);
    View v = getActivity().getLayoutInflater()
        .inflate(R.layout.dialog_date, null);
    DatePicker datePicker = (DatePicker)v.findViewById(R.id.dialog_date_datePicker);
    datePicker.init(year, month, day, new OnDateChangedListener() {
        public void onDateChanged(DatePicker view, int year, int month, int day) {
            // Translate year, month, day into a Date object using a calendar
            mDate = new GregorianCalendar(year, month, day).getTime();
            // Update argument to preserve selected value on rotation
            getArguments().putSerializable(EXTRA_DATE, mDate);
        }
    });
    ...
}
```

Return Data to CrimeFragment 1/3

Set Target Fragment

Set CrimeFragment the *target* fragment of DatePickerFragment.

To create this relationship, you call the following **Fragment** method:

```
public void setTargetFragment(Fragment fragment,  
    int requestCode)
```

The FragmentManager keeps track of the target fragment and request code.

You can retrieve them by calling `getTargetFragment()` and `getTargetRequestCode()` on the fragment that has set the target.

Set target fragment (CrimeFragment.java)

```
public class CrimeFragment extends Fragment {  
    ...  
    private static final int REQUEST_DATE = 0;  
    ...  
    @Override  
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,  
        Bundle savedInstanceState) {  
        ...  
        mDateButton.setOnClickListener(new View.OnClickListener() {  
            public void onClick(View v) {  
                FragmentManager fm = getActivity()  
                    .getSupportFragmentManager();  
                DatePickerFragment dialog = DatePickerFragment  
                    .newInstance(mCrime.getDate());  
                dialog.setTargetFragment(CrimeFragment.this, REQUEST_DATE);  
                dialog.show(fm, DIALOG_DATE);  
            }  
        });  
        return v;  
    }  
    ... }
```

Return Data to CrimeFragment 2/3

Sending Data to the Target Fragment

When dealing with two fragments hosted by the same activity, you can borrow `Fragment.onActivityResult(...)` and call it directly on the target fragment to pass back data. It has exactly what you need:

- a request code that matches the code passed into `setTargetFragment(...)` to tell the target who is returning the result
- a result code to determine what action to take
- an Intent that can have extra data

Calling back to your target (`DatePickerFragment.java`)

```
private void setResult(int resultCode) {  
    if (getTargetFragment() == null) return;  
  
    Intent i = new Intent();  
    i.putExtra(EXTRA_DATE, mDate);  
    getTargetFragment().onActivityResult(getTargetRequestCode(),  
        resultCode, i);  
}  
  
@Override  
public Dialog onCreateDialog(Bundle savedInstanceState) {  
    ...  
    return new AlertDialog.Builder(getActivity())  
        ..setView(v)  
        .setTitle(R.string.date_picker_title)  
        .setPositiveButton(android.R.string.ok, null)  
        .setPositiveButton(  
            android.R.string.ok,  
            new DialogInterface.OnClickListener() {  
                public void onClick(DialogInterface dialog, int which) {  
                    setResult(Activity.RESULT_OK);  
                }  
            }).create();  
}
```

Return Data to CrimeFragment 3/3

Responding to Dialog

In CrimeFragment, override
onActivityResult(...) to retrieve the extra, set
the date on the Crime, and refresh the
text of the date button.

Responding to the dialog (CrimeFragment.java)

```
@Override  
public void onActivityResult(int requestCode, int resultCode,  
Intent data) {  
    if (resultCode != Activity.RESULT_OK) return;  
    if (requestCode == REQUEST_DATE) {  
        Date date = (Date)data  
            .getSerializableExtra(DatePickerFragment.EXTRA_DATE);  
        mCrime.setDate(date);  
        mDateButton.setText(mCrime.getDate().toString());  
    }  
}
```

Some Code Refactoring

Cleaning up with updateDate() (CrimeFragment.java)

```
public class CrimeFragment extends Fragment {
```

```
...  
  
private void updateDate() {  
    mDateButton.setText(mCrime.getDate().toString());  
}  
  
@Override  
  
public View onCreateView(LayoutInflater inflater,  
    ViewGroup parent, Bundle savedInstanceState){  
    View v = inflater.inflate(R.layout.fragment_crime, parent,  
        false);  
    ...  
}
```

```
mDateButton =  
(Button)v.findViewById(R.id.crime_date);  
mDateButton.setText(mCrime.getDate().toString());  
updateDate();  
  
...  
}
```

```
@Override  
  
public void onActivityResult(int requestCode, int resultCode,  
    Intent data) {  
    if (resultCode != Activity.RESULT_OK) return;  
    if (requestCode == REQUEST_DATE) {  
        Date date = (Date)data  
            .getSerializableExtra(DatePickerFragment.EXTRA_DATE);  
        mCrime.setDate(date);  
        mDateButton.setText(mCrime.getDate().toString());  
        updateDate();  
    }  
}
```

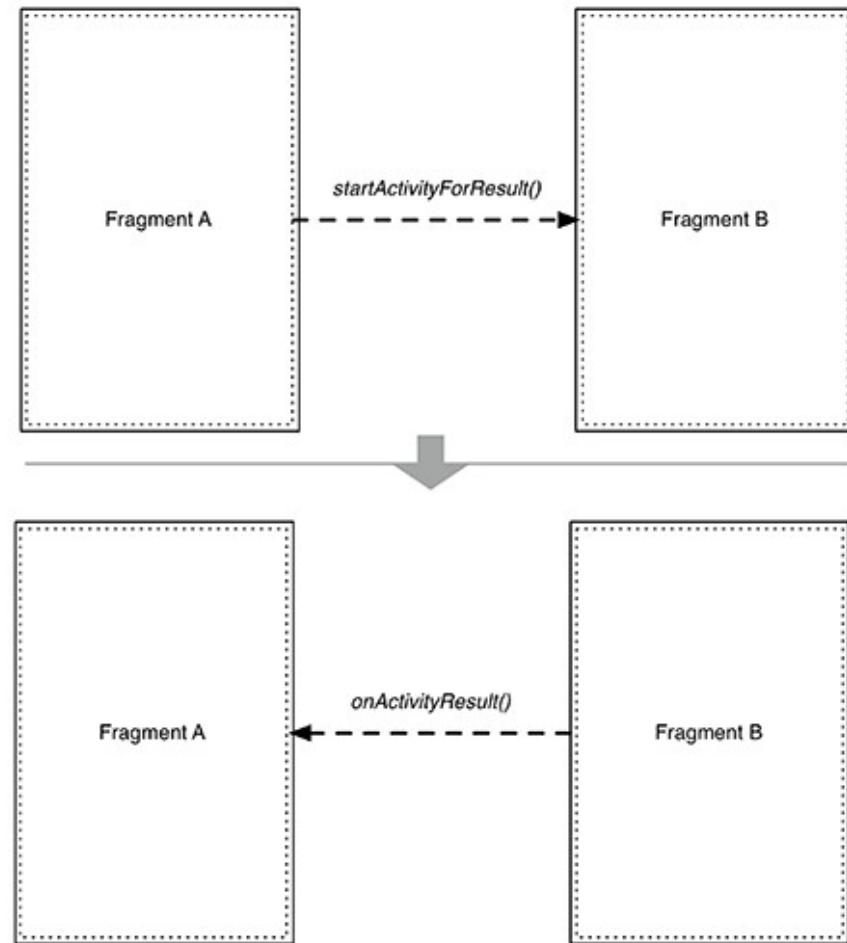
Flexibility in Presenting Dialog Fragment 2/2

Inter-activity communication on phones:

On a phone, you do not have much screen real estate, so you would likely use an activity with a full-screen fragment to ask the user for input.

This child activity would be started by a fragment of the parent activity calling `startActivityForResult(...)`.

On the death of the child activity, the parent activity would receive a call to `onActivityResult(...)`, which would be forwarded to the fragment that started the child activity.

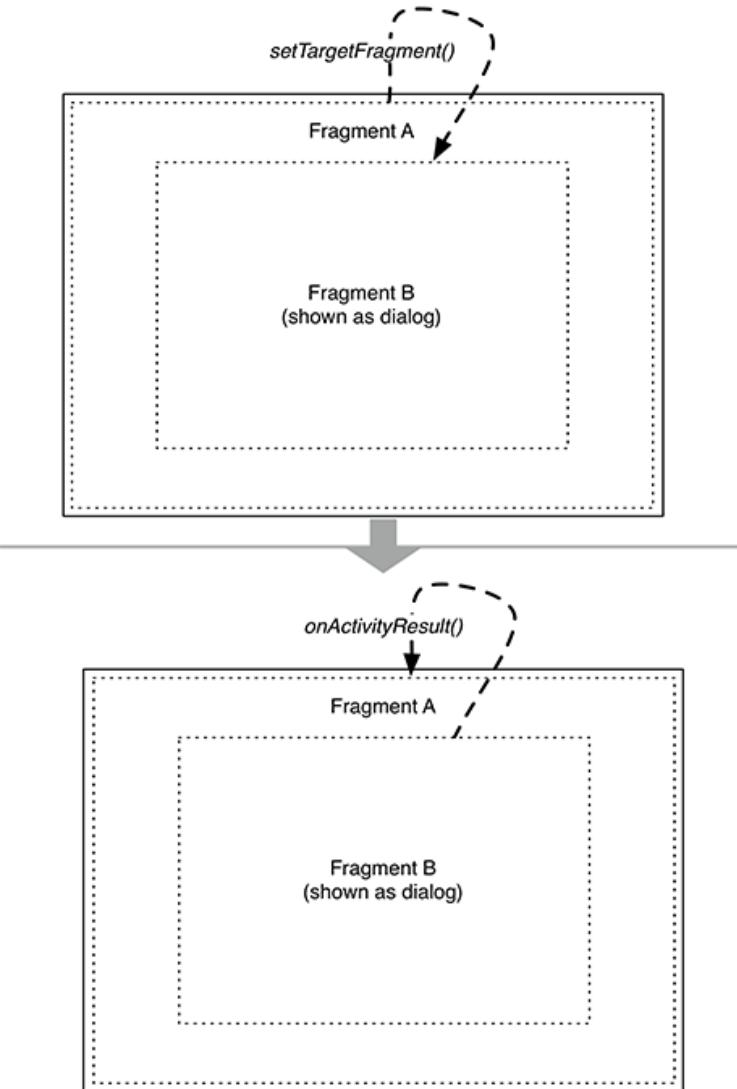


Flexibility in Presenting Dialog Fragment 1/2

Inter-fragment communication on tablets

On a tablet, where you have plenty of room, it is often better to present a **DialogFragment** to the user to get the same input.

In this case, you set the target fragment and call `show(...)` on the dialog fragment. When dismissed, the dialog fragment calls `onActivityResult(...)` on its target.



Challenge: More Dialogs

For an easy challenge, write another dialog fragment named **TimePickerFragment** that allows the user to select what time of day the crime occurred using a **TimePicker** widget. Add another button to **CrimeFragment** that will display a **TimePickerFragment**.

For a harder challenge, stay with a single button interface, but have that button display a dialog that offers the user a choice: change the time or change the date. Then fire up a second dialog when the user makes a selection.

Android UI

Audio Playback Using Media Player

Javed Hasan
BJIT Limited

Lesson Content

HelloMoon App Features and Design Overview

- User can change the date of a crime
- Object Diagram for HelloMoon

Using Layout Fragment

- Define Layout, Fragment, and Layout of the Fragment
- Lifecycle of a layout fragment

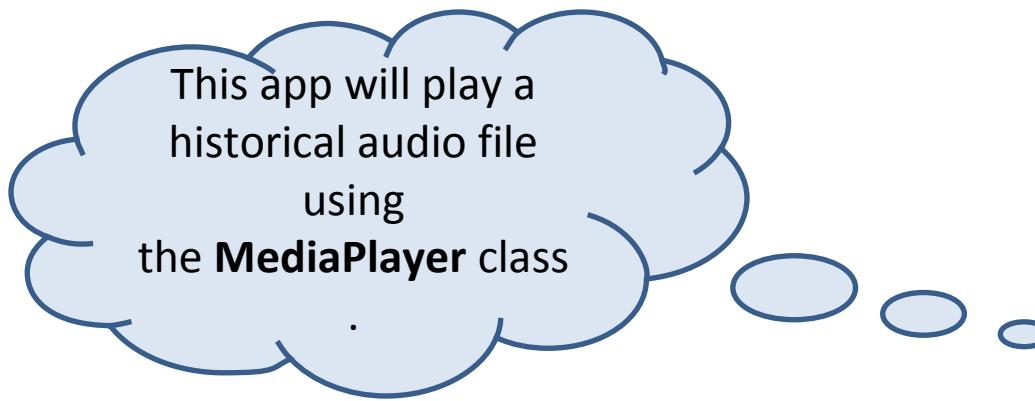
Audio Playback

- Simple Audio Playback with Play and Stop Feature

Challenge

- Implement the Pause feature
- Playing Video

Feature: Play Media File



Note: **MediaPlayer** is the Android class for audio and video playback. It can play media from different sources (like a local file or streamed from the Internet) and in many different formats (like WAV, MP3, Ogg Vorbis, MPEG-4, and 3GPP).



Object Diagram for HelloMoon

Model

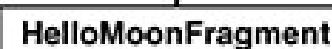


mPlayer



mPlayer

Controller



→

↓

View



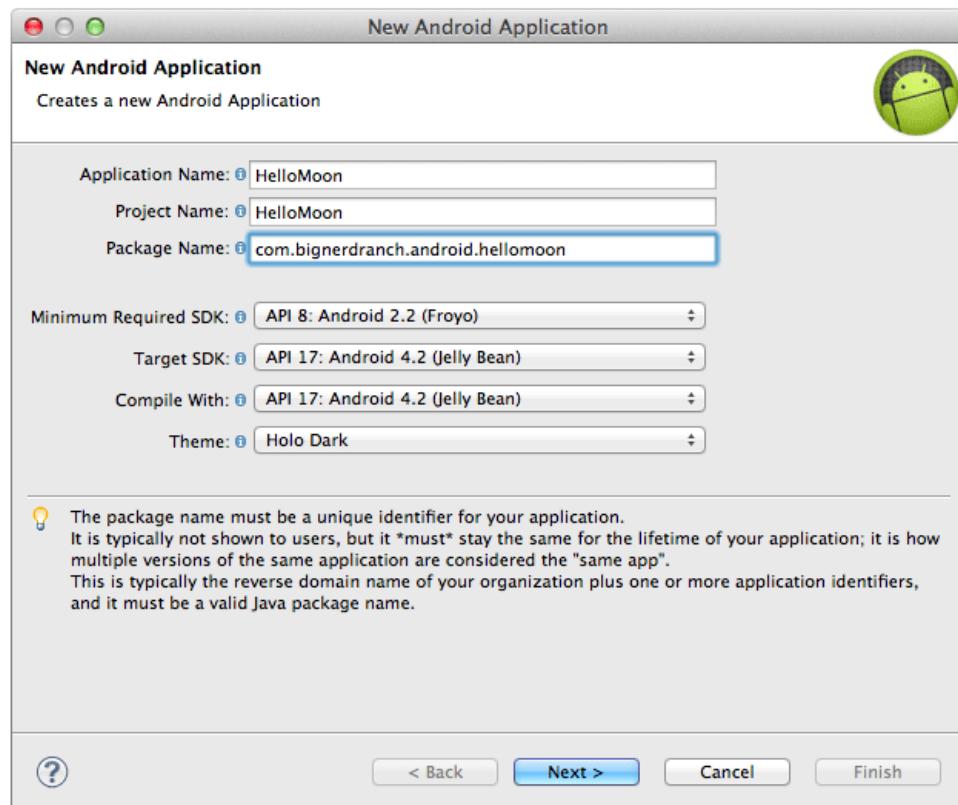
mPlayButton



mStopButton



Create HelloMoon Project



Adding strings (strings.xml)

```
<?xml version="1.0" encoding="utf-8"?>  
<resources>  
    <string name="app_name">HelloMoon</string>  
    <string name="hello_world">Hello world!</string>  
    <string name="menu_settings">Settings</string>  
    <string name="hellomoon_play">Play</string>  
    <string name="hellomoon_stop">Stop</string>  
    <string name="hellomoon_description">Neil Armstrong  
        stepping  
        onto the moon</string>  
</resources>
```

Adding Audio Resource File

Within the solutions, find the following files:

13_Audio/HelloMoon/res/drawable-mdpi/armstrong_on_moon.jpg

13_Audio/HelloMoon/res/raw/one_small_step.wav

For this simple app, we have created a single `armstrong_on_moon.jpg` file for screens of medium density (~160 dpi), which Android considers the baseline. Copy `armstrong_on_moon.jpg` to the `drawable-mdpi` directory.

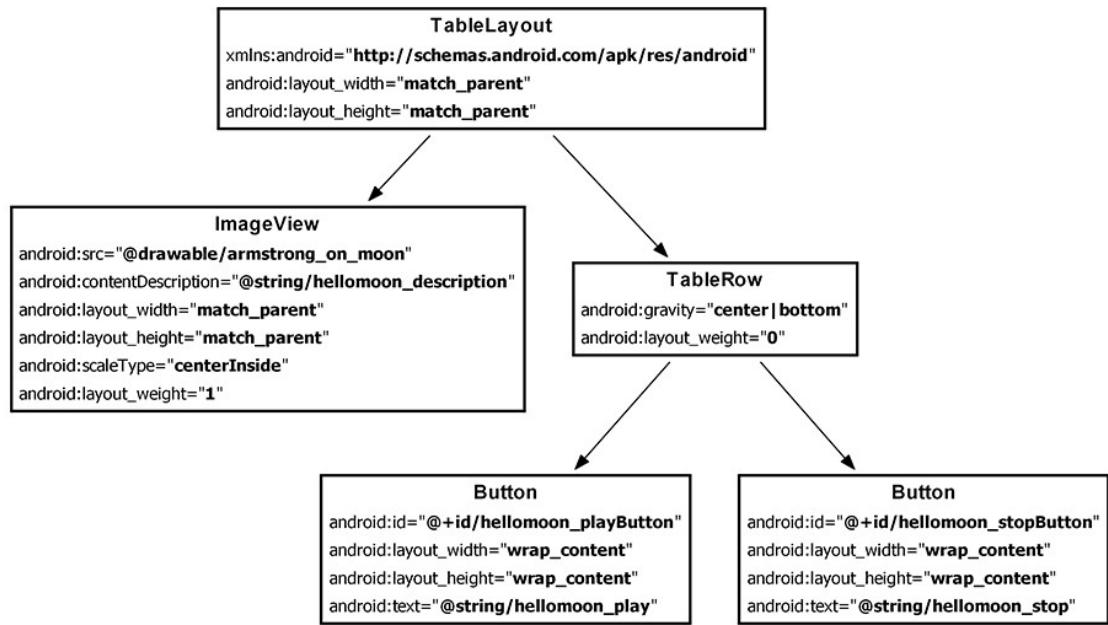
The audio file will be placed in the `res/raw` directory. The `raw` directory is a catch-all for resources that do not need special handling from Android's build system.

The `res/raw` directory is not created by default for a project, so you have to add it. (Right-click the `res` directory and select `New → Folder`.) Then copy `one_small_step.wav` to the new directory.

You can also copy `13_Audio/HelloMoon/res/raw/apollo_17_stroll.mpg` into `res/raw/`. There is information and a challenge at the end of this session about playing video that will use this file.

Using Layout Fragment

1. Define Layout Fragment



Create a new Android XML layout file named fragment_hello_moon.xml. Make its root element a **TableLayout**.

TableLayout helps arrange your buttons side by side in equally sized columns.

ImageView is kept as a direct child of TableLayout to span the entire screen. Otherwise, like **Buttons**, it needs to share space with other columns.

Using Layout Fragment

1. Define Layout Fragment

Preview the layout in the graphical tool. What color background are you seeing? You selected Holo Dark for HelloMoon's theme in the wizard. However, in some cases, the wizard ignores your theme choice and always assigns a light theme. Let's see how to fix that.

Manually resetting the app theme

```
...  
<application  
    android:allowBackup="true"  
    android:icon="@drawable/ic_launcher"  
    android:label="@string/app_name"  
    android:theme="@style/AppTheme">  
  
...  
</application>  
</manifest>
```

Modifying default styles file (res/values/styles.xml)

```
→<style name="AppBaseTheme" parent="android:Theme.Light">  
<style name="AppBaseTheme" parent="android:Theme">
```

Using Layout Fragment

2. Creating HelloMoonFragment

Create a new class named `HelloMoonFragment` and make its superclass `android.support.v4.app.Fragment`. Override `HelloMoonFragment.onCreateView(...)` to inflate the layout you just defined and get references to the buttons.

Initial HelloMoonFragment setup (`HelloMoonFragment.java`)

```
public class HelloMoonFragment extends Fragment {  
    private Button mPlayButton;  
    private Button mStopButton;  
  
    @Override  
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,  
        Bundle savedInstanceState) {  
        View v = inflater.inflate(R.layout.fragment_hello_moon, parent, false);  
        mPlayButton = (Button)v.findViewById(R.id.hellomoon_playButton);  
        mStopButton = (Button)v.findViewById(R.id.hellomoon_stopButton);  
        return v;  
    }  
}
```

Using Layout Fragment

3. Creating HelloMoonFragment

When you use a layout fragment, you specify the fragment class in a fragment element. Open `activity_hello_moon.xml` and replace its contents with the fragment element shown below:

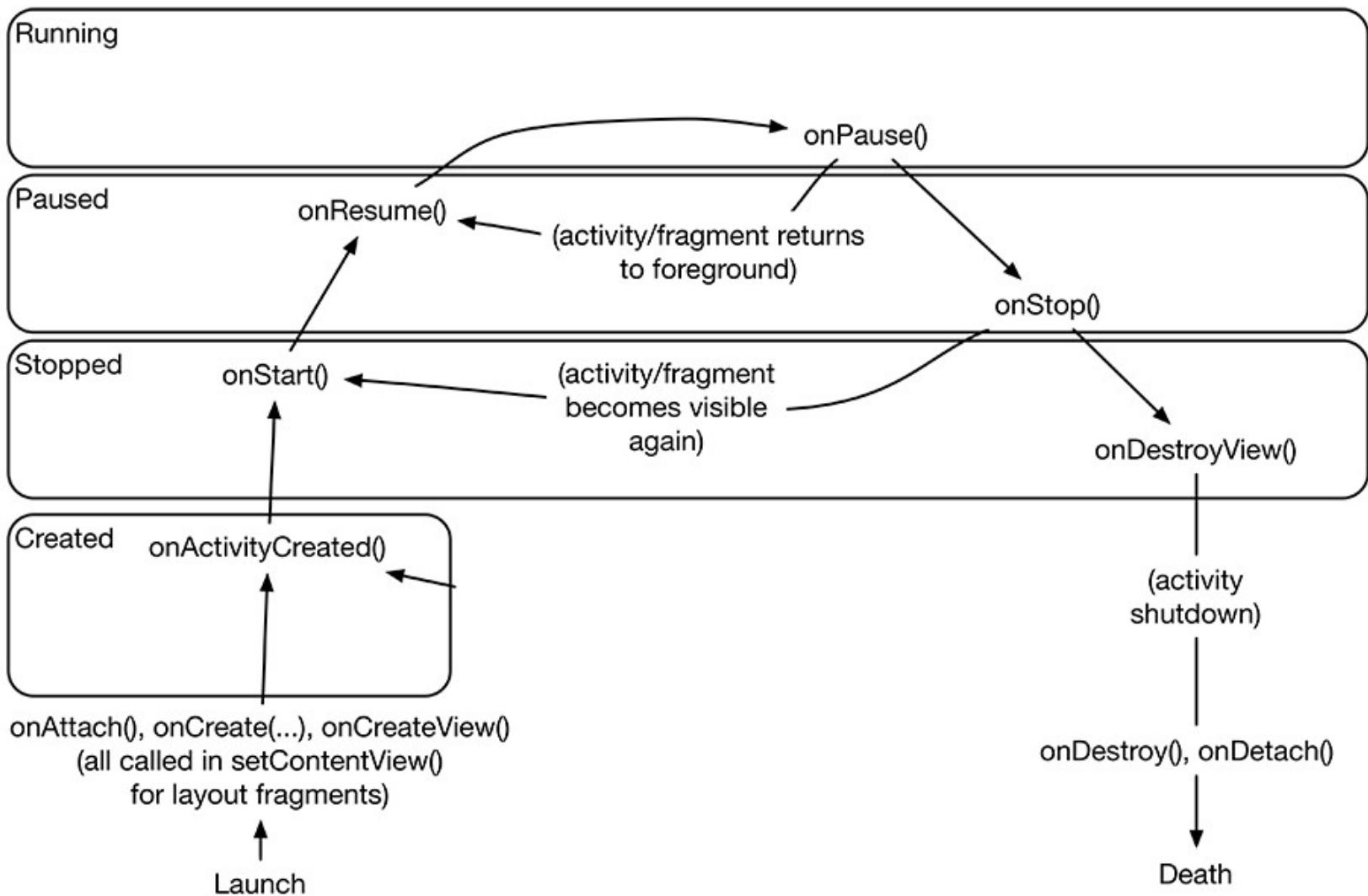
Creating a layout fragment (`activity_hello_moon.xml`)

```
<?xml version="1.0" encoding="utf-8"?>  
<fragment xmlns:android="http://schemas.android.com/apk/res/android"  
    android:id="@+id/helloMoonFragment"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:name="com.bignerdranch.android.hellomoon.HelloMoonFragment">  
</fragment>
```

Make HelloMoonActivity a FragmentActivity (`HelloMoonActivity.java`)

```
public class HelloMoonActivity extends FragmentActivity {  
  
    /** Called when the activity is first created. */  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_hello_moon);  
    }  
}
```

Lifecycle of a Layout Fragment



Limitations of Layout Fragment

- You can override the fragment's lifecycle methods to respond to events, but you have no control over when these methods are called.
- You cannot commit fragment transactions that remove, replace, or detach a layout fragment. You are stuck with what you have when the activity is created.
- You cannot attach arguments to a layout fragment. Attaching arguments must be done after the fragment is created and before it is added to the FragmentManager. With layout fragments, these events all happen out of your reach.

But with a simple app or with a static part of a complex app, using a layout fragment can be a reasonable choice.

Audio Playback 1/2

Create a new class in the
com.bignerdranch.android.hellomoon package named
AudioPlayer. Leave its superclass as
java.lang.Object..

In AudioPlayer.java, add a member variable to hold an instance of MediaPlayer and methods to stop and play this instance.

In stop(), we release the MediaPlayer.
Otherwise, MediaPlayer will hold on to the audio decoder hardware and other system resources. These resources are shared across all apps.

```
public class AudioPlayer {  
  
    private MediaPlayer mPlayer;  
  
    public void stop() {  
        if (mPlayer != null) {  
            mPlayer.release();  
            mPlayer = null;  
        }  
    }  
  
    public void play(Context c) {  
        mPlayer = MediaPlayer.create(c, R.raw.one_small_step);  
        mPlayer.start();  
    }  
}
```

Audio Playback 2/2

For safety, keep exactly one MediaPlayer around and keep it as long as it is playing something.

Calling stop() at the beginning of play(Context) prevents the creation of multiple instances ofMediaPlayer.

Calling stop() when the file has finished playing releases your hold on the MediaPlayer instance.

You also need to call AudioPlayer.stop() in HelloMoonFragment to prevent the MediaPlayer from continuing playback after the fragment has been destroyed.

There can be only one (AudioPlayer.java)

```
...
public void play(Context c) {
    stop();
    mPlayer = MediaPlayer.create(c, R.raw.one_small_step);
    mPlayer.setOnCompletionListener(new MediaPlayer.OnCompletionListener() {
        public void onCompletion(MediaPlayer mp) {
            stop();
        }
    });
    mPlayer.start();
}
```

Override onDestroy() (HelloMoonFragment.java)

```
...
@Override
public void onDestroy() {
    super.onDestroy();
    mPlayer.stop();
}
```

Wiring Buttons to Play and Stop

Hook up Play button (HelloMoonFragment.java)

```
public class HelloMoonFragment extends Fragment {  
    private AudioPlayer mPlayer = new AudioPlayer();  
    private Button mPlayButton;  
    private Button mStopButton;  
    @Override  
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,  
        Bundle savedInstanceState) {  
        View v = inflater.inflate(R.layout.fragment_hello_moon, parent, false);  
        mPlayButton = (Button)v.findViewById(R.id.hellomoon_playButton);  
        mPlayButton.setOnClickListener(new View.OnClickListener() {  
            public void onClick(View v) {  
                mPlayer.play(getActivity());  
            }  
        });  
        mStopButton = (Button)v.findViewById(R.id.hellomoon_stopButton);  
        mStopButton.setOnClickListener(new View.OnClickListener() {  
            public void onClick(View v) {  
                mPlayer.stop();  
            }  
        });  
        return v;  
    }  
}
```

Challenge 1: Pausing Audio Playback

Give users the option to pause the audio playback. Head to the reference for the MediaPlayer class to find the methods you will need to implement pausing.

You have only scratched the surface of using MediaPlayer in this chapter. For information on what else MediaPlayer can do, read Android's "MediaPlayer Developer Guide" at <http://developer.android.com/guide/topics/media/mediaplayer.html>.

Challenge 2: Playing Video

Modify HelloMoon so that it can also play video. If you did not fetch the [apollo_17_stroll.mpg](#) file earlier, go back to the solutions file and copy it from this chapter's project into your res/raw directory. Then play it using one of the methods described above.

When it comes to playing video, you have a couple more options. One way to do it is to use MediaPlayer, just like you have been. All you need to do is hook up a place to display the video.

Imagery that updates quickly (like video) is displayed in a **SurfaceView** on Android. Actually, it is displayed on a **Surface**, which the **SurfaceView** hosts. You can get at the **Surface** by obtaining the **SurfaceView's SurfaceHolder**.

You will learn about these things in detail in a later session. For now, all you need to know is that you can hook up a **SurfaceHolder** to a **MediaPlayer** by calling **MediaPlayer.setDisplay(SurfaceHolder)**.

Android UI Retained Fragments

Javed Hasan
BJIT Limited

Lesson Content

Retaining a Fragment

- Rotation Problem in HelloMoon
- How to Retain a Fragment

Rotation and Retained Fragments

- Default rotation with a UI fragment
- Rotation with a retained UI fragment
- Rotation Handling and onSaveInstanceState(Bundle)

Rotation Problem in HelloMoon

Currently, HelloMoon does not handle rotation gracefully. Run HelloMoon, play the audio, and then rotate the device. The audio will stop abruptly.

Here is the problem: On rotation, the **HelloMoonActivity** is destroyed. As this is happening, the **FragmentManager** is responsible for destroying the **HelloMoonFragment**. The **FragmentManager** calls the fragment's waning lifecycle methods: **onPause()**, **onStop()**, and **onDestroy()**.

In **HelloMoonFragment.onDestroy()**, you release the MediaPlayer, which stops the playback.

Though Fragment has **onSaveInstanceState(Bundle)** method, saving out the state of the MediaPlayer object and restoring it later will still interrupt playback and annoy your users.

Retaining a Fragment

Calling `setRetainInstance(true)`
(HelloMoonFragment.java)

...

```
private Button mPlayButton;  
private Button mStopButton;  
  
@Override  
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setRetainInstance(true);  
}
```

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

When a fragment is retained, the fragment is not destroyed with the activity. Instead, it is preserved and passed along intact to the new activity.

When you retain a fragment, you can count on all of its **instance variables** (like `mPlayButton`, `mPlayer`, and `mStopButton`) to keep the same values. When you reach for them, they are simply there.

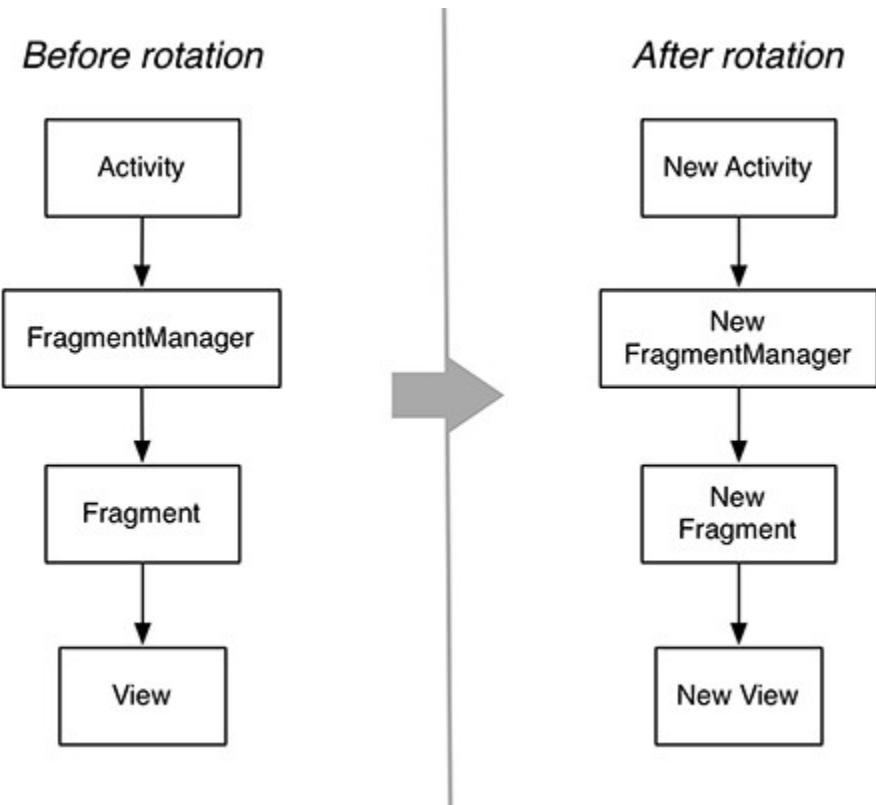
Run HelloMoon again. Play the audio, rotate the device, and confirm that playback continues unimpeded.

Rotation with a Retained UI Fragment

During a configuration change, the **FragmentManager** first destroys the views of the fragments in its list.

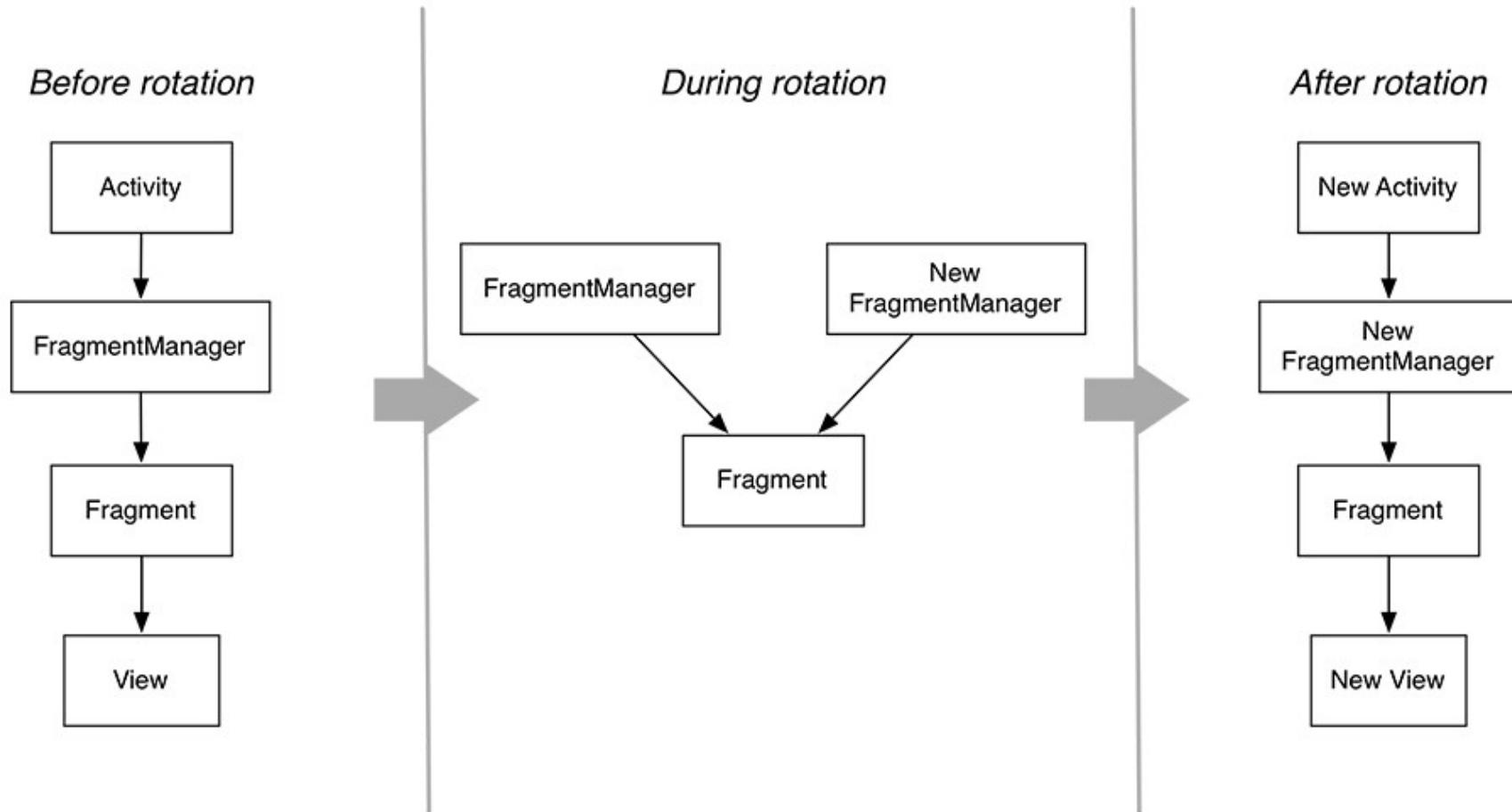
Next, the **FragmentManager** checks the **retainInstance** property of each fragment. If it is false, which it is by default, then the **FragmentManager** destroys the fragment instance.

The fragment and its view will be recreated by the new **FragmentManager** of the new activity “on the other side.”



Rotation with a Retained UI Fragment

On the other hand, if `retainInstance` is true, then the fragment's view is destroyed, but the fragment itself is not. When the new activity is created, the new **FragmentManager** finds the retained fragment and recreates its view.



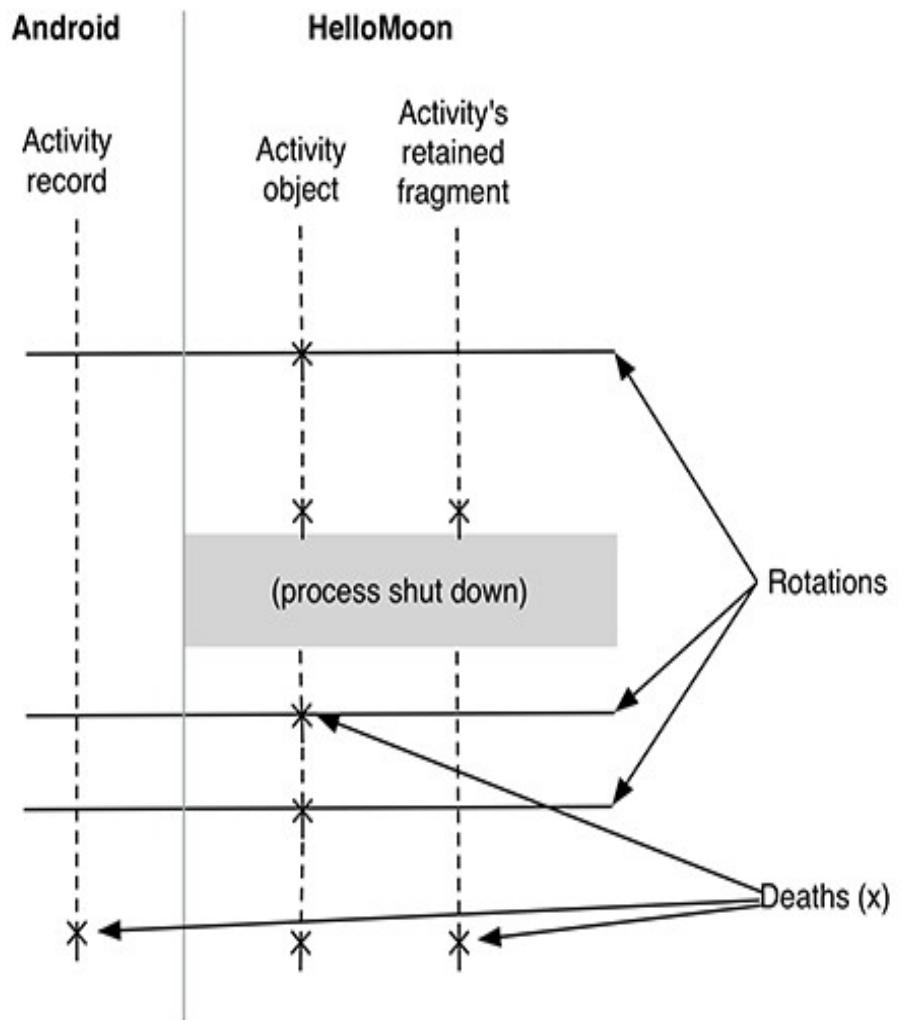
Fragment Lifecycle Revisited

The major difference between overriding `Fragment.onSaveInstanceState(...)` and retaining the fragment is how long the preserved data lasts.

If it needs to last long enough to survive configuration changes, then retaining the fragment is much less work.

If you have something in your activity or fragment that should last a long time, then you should tie it to the `onSaveInstanceState(Bundle)`

Figure the three different lifetimes you have to work with: the life of the activity object (and its unretained fragments), the life of a retained fragment, and the life of the activity record.



Android UI Localization

Javed Hasan
BJIT Limited

Lesson Content

HelloMoon Localization

- Supporting multiple languages
- Localizing resources, default resources

Configuration Qualifiers

- Prioritizing alternative resources
- Multiple qualifiers

Resource Rules and Regulations

- Resource naming
- Resource directory structure

Testing Alternative Resources

HelloMoon Localization

Localization is the process of providing the appropriate resources for your app based on the device's language settings.

You are going to *localize* HelloMoon.

In particular, you will provide Spanish versions of the audio file and the strings file.

When a device's language is set to Spanish, Android will find and use the Spanish resources.



Localizing Resources

Android provides configuration qualifiers for different languages. The language configuration qualifiers are taken from ISO 639-1 codes. For Spanish, the qualifier is -es.

In HelloMoon project, create two new res subdirectories: res/raw-es/ and res/values-es/.

From solutions file, locate

15_Localization/HelloMoon/res/raw-es/one_small_step.wav

15_Localization/HelloMoon/res/values-es/strings.xml

Copy these files into the corresponding directories that you just created.

To confirm, change device language to Spanish and run HelloMoon again.

You can qualify a resource directory with a language-plus-region. For instance, the qualifier for Spanish spoken in Spain is -es-rES, where the r denotes a region qualifier and ES is the ISO 3166-1-alpha-2 code for Spain.

Note: configuration qualifiers are not case-sensitive, but it is good to follow Android's convention here: use a lowercase language code and an uppercase region code prefixed with a lowercase r.

Default Resources

Default resources are resources in unqualified directories. For example, `raw` and `values`.

It is important to provide a default resource. If there is no default to fall back on, then your app will crash if Android looks for a resource and cannot find one that matches the device configuration.

Screen density works differently

A project's drawable directories are typically qualified for screen density with `-mdpi`, `-hdpi`, and `-xhdpi`. Putting default drawable resources in `res/drawable/` is not necessary.

Because Android may choose a drawable from a directory that is qualified with a lower or higher density than the device and then scale the drawable. This is an exception.

Configuration Qualifiers

Below list is the characteristics of the device configuration that have configuration qualifiers that the Android system recognizes for targeting resources.

1. Mobile country code (MCC), optionally followed by mobile network code (MNC)
2. Language code, optionally followed by region code
3. Layout direction
4. Smallest width
5. Available width
6. Available height
7. Screen size
8. Screen aspect
9. Screen orientation
10. and many more

You can find descriptions of these characteristics and examples of specific configuration qualifiers at

<http://developer.android.com/guide/topics/resources/providing-resources.html#AlternativeF>

Prioritizing Alternative Resources

When the device configuration will match more than one alternative resource, qualifiers are given precedence based on the order shown in list (see previous slide).

Let's add another alternative resource to HelloMoon – a landscape version of the `app_name` string resource. Create a `values-land` directory and copy the default strings file into it.



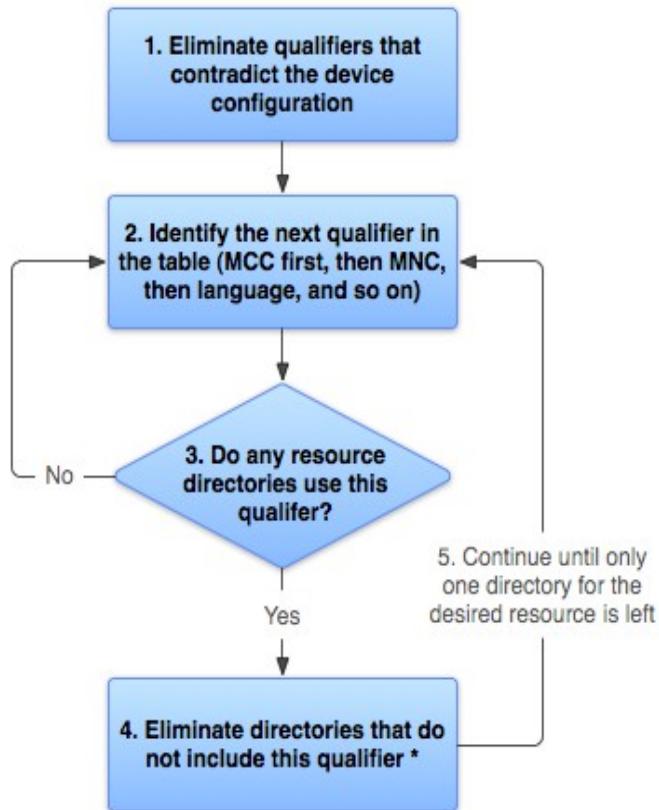
Creating a landscape alternative string resource (`values-land/strings.xml`)

```
<resources><string name="app_name">Hello, Moon! How are you? Is it cold up  
there?</string></resources>
```

Now you have three versions of `app_name`: a default version in `values/strings.xml`, a Spanish alternative in `values-es/strings.xml`, and a landscape alternative in `values-land/strings.xml`.

With your device's language set to Spanish, run HelloMoon and rotate to landscape. The Spanish language alternative has precedence, so you see the string from `values-es/strings.xml`.

Finding the Best Matching Resources



* If the qualifier is screen density, the system selects the "best match" and the process is done

First, consider the three alternatives for the string resource named **app_name**:

values-es/strings.xml, values-land/strings.xml and values-es-land/strings.xml

Device config has Spanish language and a landscape screen orientation

No directories contain an MCC qualifier, so no directories are ruled out, and Android moves down the list to the language qualifier. Two directories (values-es/ and values-es-land/) contain language qualifiers. One directory (values-land/) does not and is ruled out.

Android keeps stepping down the qualifier list. When it reaches screen orientation, it will find one directory with a screen orientation qualifier and one with out. It rules out values-es/, and values-es-land/ is the only directory remaining. Thus, Android uses the resource in values-es-land/.

Resource Rules and Regulations

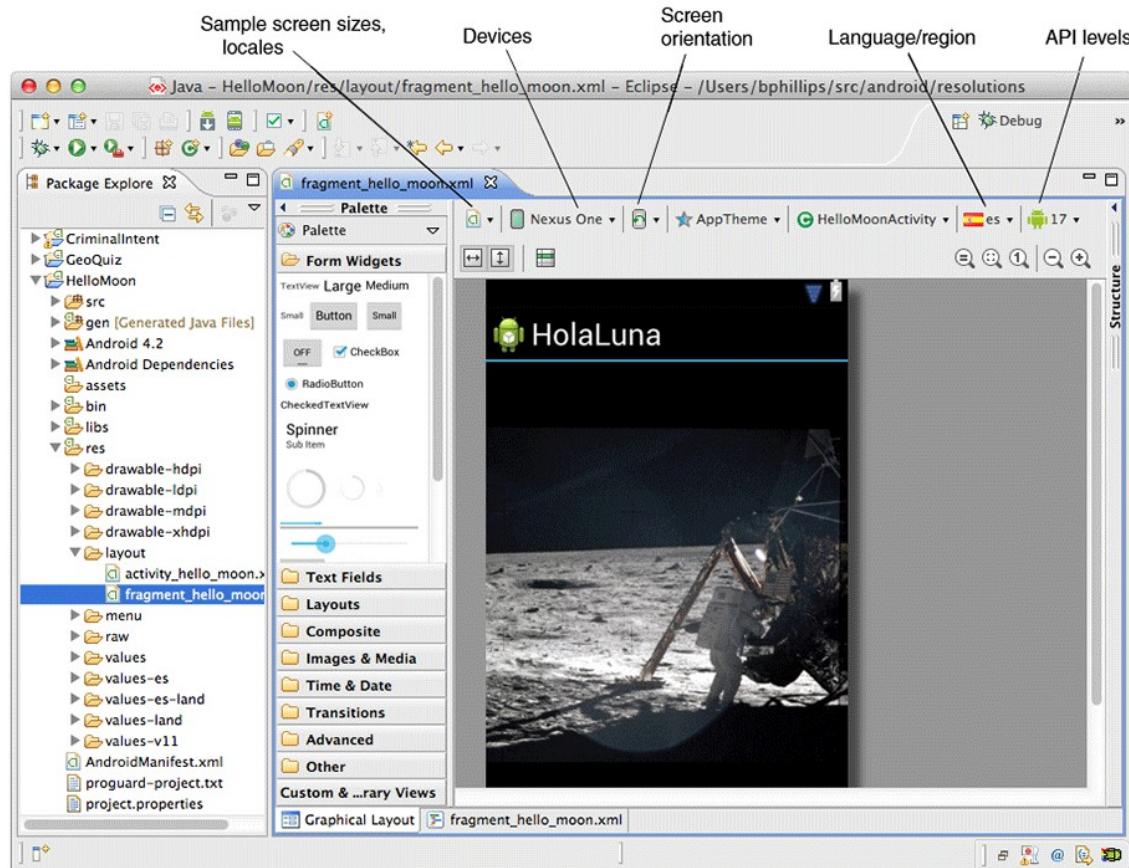
1. Resource names must be lowercase and must not have any spaces
 - one_small_step.wav, app_name,armstrong_on_moon.jpg
2. Whether you reference a resource in XML or code, the reference does not include the file extension. For example, you refer to @drawable/armstrong_on_moon in your layout files and R.drawable.armstrong_on_moon in your code
3. Resources must be saved in a subdirectory of res/. You cannot save resources in the root of res/. Doing so will cause build errors
4. Any additional subdirectories you create in res/ will be ignored. Creating res/my_stuff may not cause an error, but Android will not use any resources you place in there.

You can see a complete list of supported (unqualified) res subdirectories at

<http://developer.android.com/guide/topics/resources/providing-resources.html#Resources>

1. You cannot create additional levels of subdirectories in res/

Testing Alternative Resources



You can test on devices both real and virtual. You can also use the graphical layout tool.

You can ensure that you have included all of your necessary default resources by setting a device to a language that you have not localized any resources for.

Run your app and put it through its paces. Visit all of the views and rotate them. If the app crashes, check LogCat for a “Resource not found...” message to track down the missing default resource.

Android UI

The Action Bar

Javed Hasan
BJIT Limited

Lesson Content

CriminalIntent New Feature

- Add new crime using Options Menu in the Action Bar
- Enable Ancestral Navigation through App Icon

Options Menu

- Define Options Menu in XML
- Create Options Menu
- Respond to Options Menu selection

Enabling Ancestral Navigation

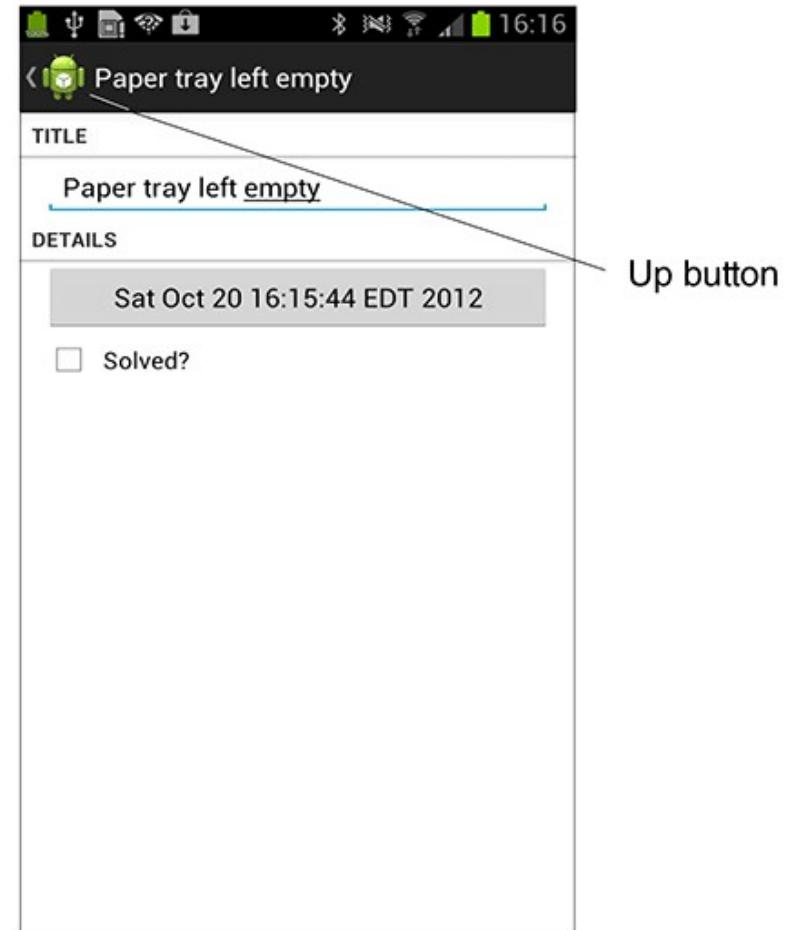
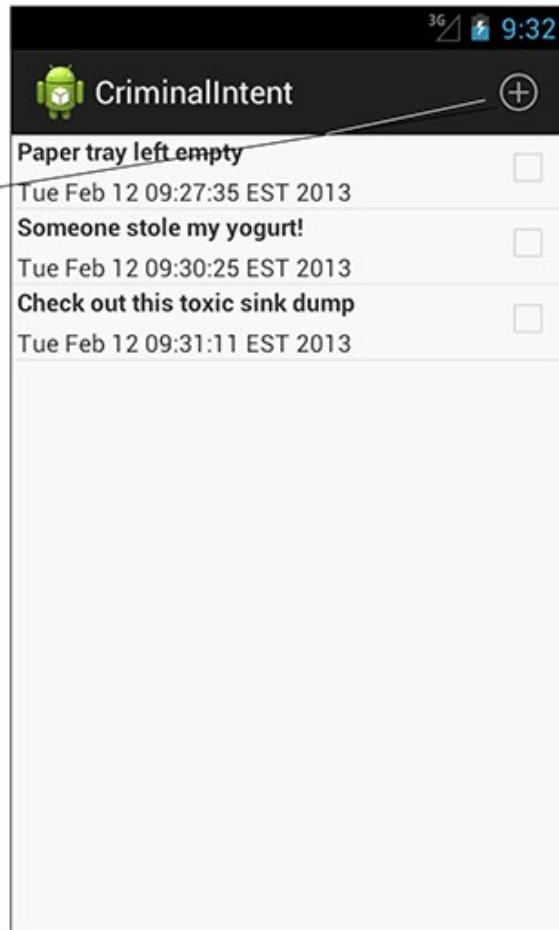
- Enable App icon
- Turn on Up button
- Respond to the Up button

Add an Alternative Menu Item

- Toggle the menu item title

New Feature: Action Bar Options Menu

Menu item to
add a new
crime



Options Menu Overview

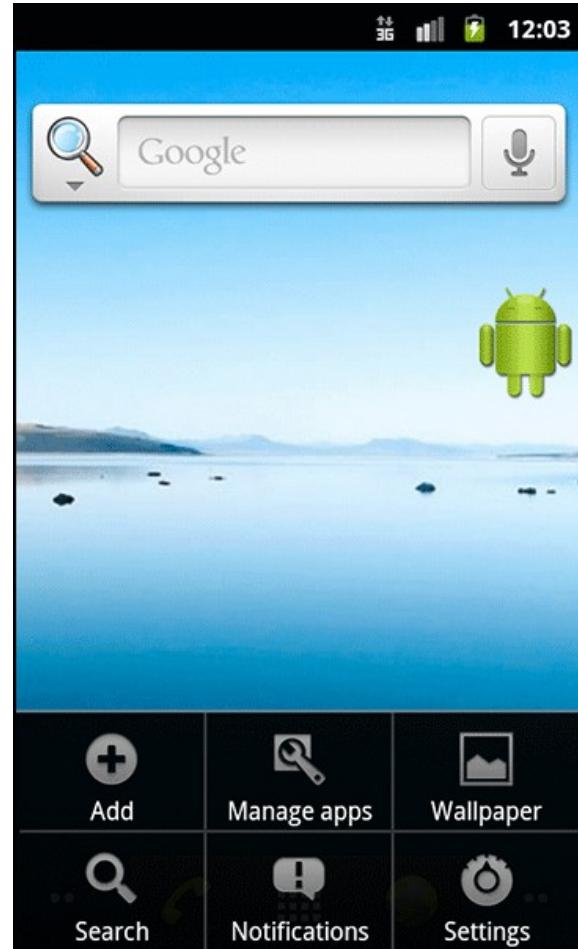
The menu that can appear on the action bar is called the options menu.

An options menu presents users with choices that refer to the entire screen or the app as a whole. Adding a new crime is a good example.

The action bar as the home for options menus is new, but options menus themselves have been around since the beginning of Android.

Luckily, there are very few compatibility problems with options menus. The code is the same, and each device handles how to present the options menu depending on its level of Android.

Pre-Honeycomb options menus

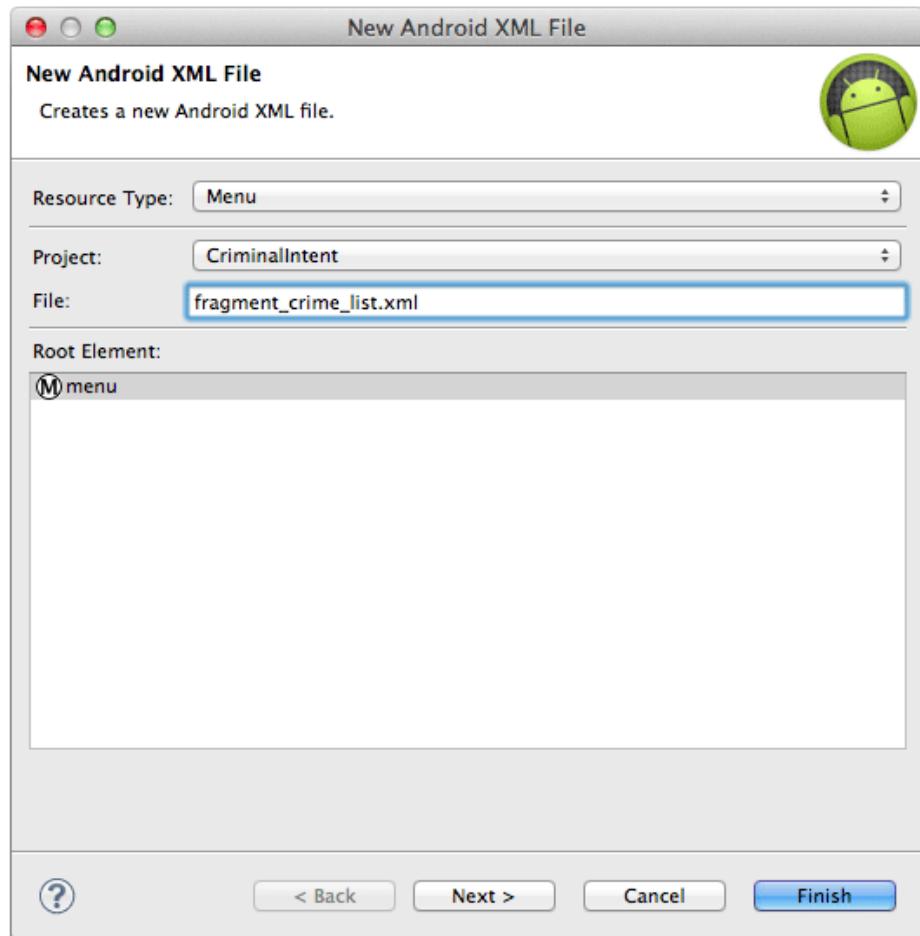


Defining an Options Menu 1/3

In the **package explorer**,
create a menu subdirectory
within res/.

Then right-click on this new
directory and
choose New → Android XML
File.

Ensure that the resource type
of the file you are creating
is **Menu**, and name the file
fragment_crime_list.xml.



Defining an Options Menu 2/3

Creating a menu resource
CrimeListFragment
(fragment_crime_list.xml)

```
<?xml version="1.0" encoding="utf-8"?> <menu  
    xmlns:android="http://schemas.android.com/apk/res/a  
    ndroid"> <item  
        android:id="@+id/menu_item_new_crime"  
        android:icon="@android:drawable/ic_menu_add"  
        android:title="@string/new_crime"  
        android:showAsAction="ifRoom|withText"/> </menu>
```

Adding strings for menus
(res/values/strings.xml)

```
... <string name="crimes_title">Crimes</string> <string  
name="date_picker_title">Date of crime:</string>  
<string name="new_crime">New Crime</string>  
<string name="show_subtitle">Show Subtitle</string>  
<string name="hide_subtitle">Hide Subtitle</string>  
<string name="subtitle">If you see something, say  
something.</string> <string  
name="delete_crime">Delete</string> </resources>
```

The **showAsAction** attribute refers to whether the item will appear on the action bar itself or in the overflow menu.

You have piped together two values, **ifRoom** and **withText**, so the item's icon and text will appear on the action bar if there is room.

If there is room for the icon but not the text, then only the icon will be visible.

If there is no room for either, then the item will be relegated to the **overflow menu**.

In the **android:icon** attribute, the value `@android:drawable/ic_menu_add` references a system icon. A system icon is one that is found on the device rather than in your project's resources.

Defining an Options Menu 3/3



Overflow menu in the action bar

Creating an Options Menu 1/2

The methods for creating the options menu and responding to the selection of a menu item are

```
public void onCreateOptionsMenu(Menu menu, MenuItemInflater  
inflater)  
  
public boolean onOptionsItemSelected(MenuItem item)
```

Inflating an options menu
(CrimeListFragment.java)

```
@Override  
  
public void onResume() {  
    super.onResume();  
    ((CrimeAdapter) getListAdapter()).notifyDataSetChanged();  
}  
  
@Override  
  
public void onCreateOptionsMenu(Menu menu,  
MenuItemInflater inflater) {  
    super.onCreateOptionsMenu(menu, inflater);  
    inflater.inflate(R.menu.fragment_crime_list, menu);  
}
```

The FragmentManager is responsible for calling Fragment.onCreateOptionsMenu(Menu, MenuItemInflater) when the activity receives its onCreateOptionsMenu(...) callback from the OS.

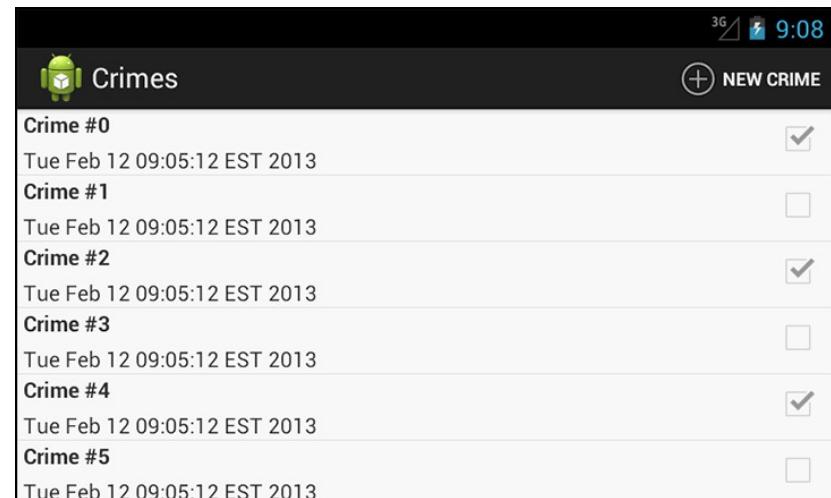
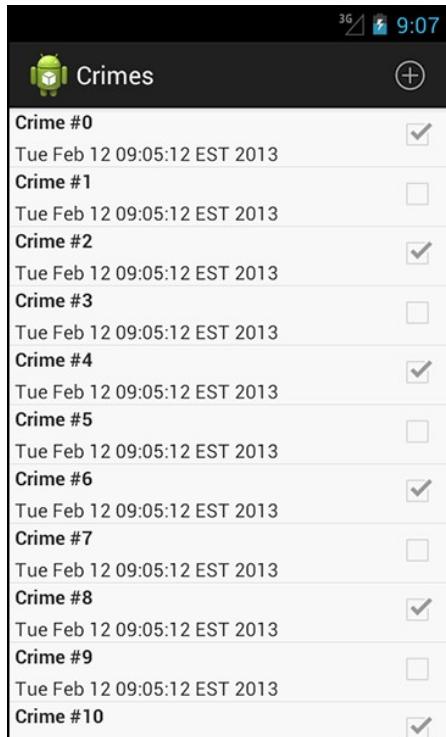
You must explicitly tell theFragmentManager that your fragment should receive a call to onCreateOptionsMenu(...). You do this by calling the following method:

```
public void setHasOptionsMenu(boolean hasMenu)
```

(CrimeListFragment.java)

```
@Override  
  
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setHasOptionsMenu(true);  
  
    getActivity().setTitle(R.string.crimes_title);  
    ...
```

Options Menu in Different Orientations



Responding to Options Menu Selections 1/2

To respond to the user pressing the **New Crime** item, you need a way to add a new Crime to your list of crimes.

Adding a new crime (CrimeLab.java)

```
...
public void addCrime(Crime c) {
    mCrimes.add(c);
}
public ArrayList<Crime> getCrimes() {
    return mCrimes;
}
...
```

Goodbye, random crimes! (CrimeLab.java)

```
public CrimeLab(Context applicationContext) {
    mApplicationContext = applicationContext;
    mCrimes = new ArrayList<Crime>();
    for (int i = 0; i < 100; i++) {
        Crime c = new Crime();
        c.setTitle("Crime #" + i);
        c.setDate(new Date());
        c.setSolved(i % 2 == 0); // Every other one
        mCrimes.add(c);
    }
}
```

Responding to Options Menu Selections 2/2

When the user presses a menu item in the options menu, your fragment receives a callback to the method

`onOptionsItemSelected(MenuItem)`.

This method receives an instance of **MenuItem** that describes the user's selection.

This method returns a boolean value. Once you have handled the menu item, you should return true to indicate that no further processing is necessary.

Responding to menu selection
(`CrimeListFragment.java`)

`@Override`

```
public void onCreateOptionsMenu(Menu menu, MenuItemInflater inflater) {  
    super.onCreateOptionsMenu(menu, inflater);  
    inflater.inflate(R.menu.fragment_crime_list, menu);  
}
```

`@Override`

```
public boolean onOptionsItemSelected(MenuItem item) {  
    switch (item.getItemId()) {  
  
        case R.id.menu_item_new_crime:  
            Crime crime = new Crime();  
            CrimeLab.get(getActivity()).addCrime(crime);  
            Intent i = new Intent(getActivity(), CrimePagerActivity.class);  
            i.putExtra(CrimeFragment.EXTRA_CRIME_ID, crime.getId());  
            startActivityForResult(i, 0);  
            return true;  
  
        default:  
            return super.onOptionsItemSelected(item);  
    }  
}
```

Enabling Ancestral Navigation 1/5

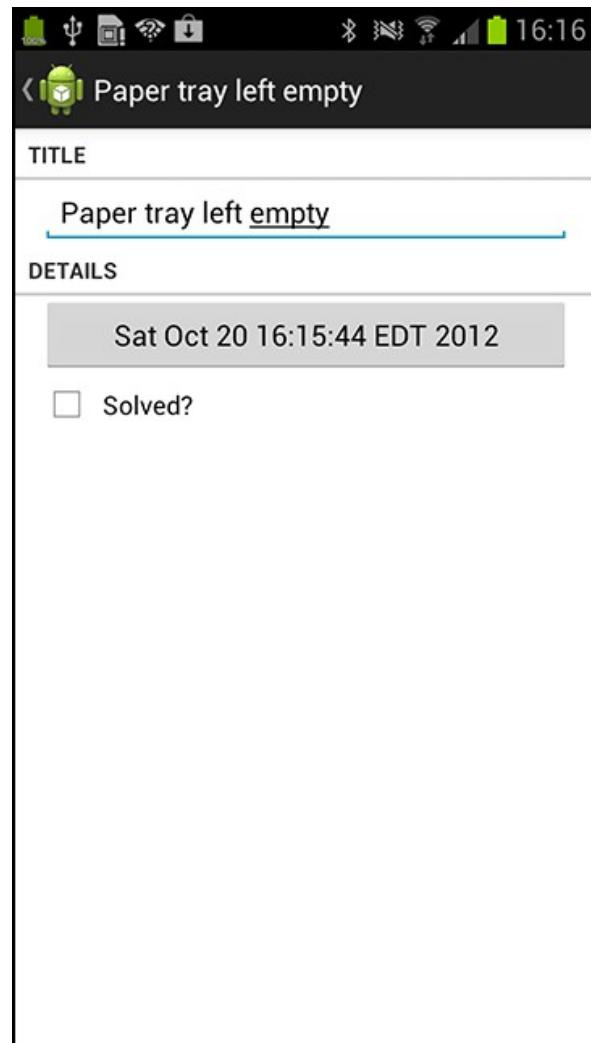
Using the **Back** button is temporal navigation.
It takes you to where you were last.

Ancestral navigation, on the other hand,
takes you up the app hierarchy by one level.

Android makes it easy to use the app icon on
the action bar for ancestral navigation.

Android recommends that you implement the
app icon to go “up” just one level to the
parent of the current activity. When you do
this, the icon becomes the “Up” button.

Typically, you let users know that the app
icon is enabled as an Up button by displaying
a left-pointing caret to the left of the app icon
like the one in [Figure](#)



Enabling Ancestral Navigation 2/5

To enable the app icon to work as a button and get the caret to appear in the fragment's view, you must set a property on the fragment by calling the following method:

```
public abstract void setDisplayHomeAsUpEnabled(boolean  
                                         showHomeAsUp)
```

This method is from API level 11, so you need to wrap it to keep the app Froyo- and Gingerbread-safe and annotate the method to wave off Android Lint.

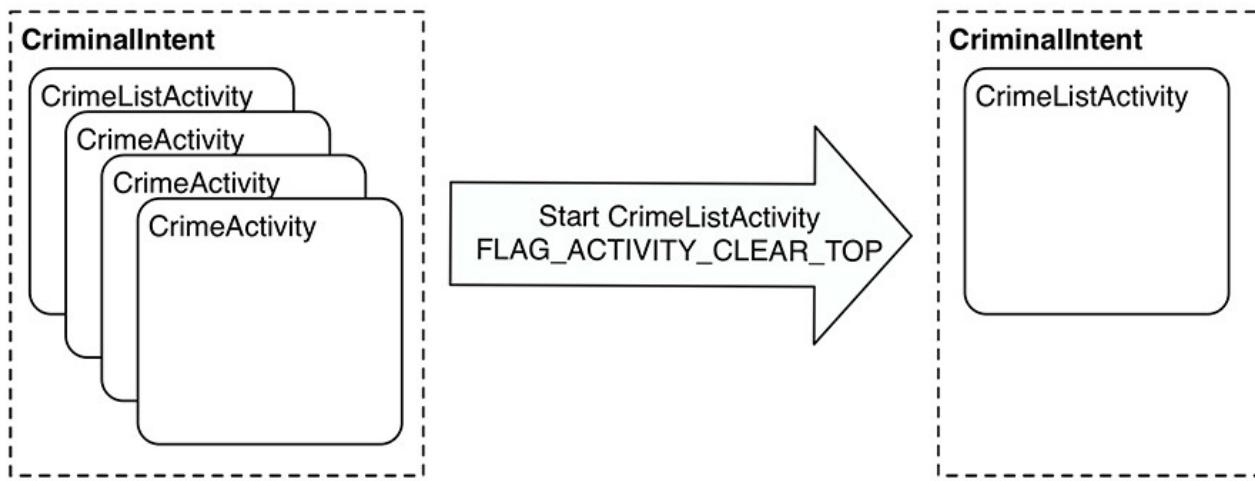
Turn on Up button (CrimeFragment.java)

```
@TargetApi(11)  
 @Override  
 public View onCreateView(LayoutInflater inflater, ViewGroup  
 parent,  
                         Bundle savedInstanceState) {  
     View v = inflater.inflate(R.layout.fragment_crime, parent, false);  
     if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {  
         getActivity().getActionBar().setDisplayHomeAsUpEnabled(true);  
     }  
     ...  
 }
```

Turn on options menu handling (CrimeFragment.java)

```
@Override  
 public void onCreate(Bundle savedInstanceState) {  
     ...  
     setHasOptionsMenu(true);  
 }
```

Enabling Ancestral Navigation 3/5



You want this button to return the user to the list of crimes, so you might think to create an intent and start the instance of `CrimePagerActivity` like this:

```
Intent intent = new Intent(getActivity(), CrimeListActivity.class);
intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
startActivity(intent);
finish();
```

`FLAG_ACTIVITY_CLEAR_TOP` tells Android to look for an existing instance of the activity in the stack, and if there is one, pop every other activity off the stack so that the activity being started will be top-most. **There is a better way.**

Enabling Ancestral Navigation 4/5

Implement ancestral navigation: use a convenience class named **NavUtils** plus some metadata in the manifest.

Add parent activity metadata (AndroidManifest.xml)

```
<activity android:name=".CrimePagerActivity"
    android:label="@string/app_name">
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value=".CrimeListActivity"/>
</activity>
...
...
```

Using NavUtils (CrimeFragment.java)

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case android.R.id.home:
            if (NavUtils.getParentActivityName(getActivity()) != null) {
                NavUtils.navigateUpFromSameTask(getActivity());
            }
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Enabling Ancestral Navigation 5/5

If there is no parent named in the metadata, then you do not want to display the up caret. Back in `onCreateView(...)`, check for a parent before calling `setDisplayHomeAsUpEnabled(true)`.

No parent, no caret (`CrimeFragment.java`)

```
@TargetApi(11)
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup parent,
    Bundle savedInstanceState) {
    View v = inflater.inflate(R.layout.fragment_crime, parent, false);

    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
        if (NavUtils.getParentActivityName(getActivity()) != null) {
            getActivity().getActionBar().setDisplayHomeAsUpEnabled(true);
        }
    }
}

...
}
```

Why is Using NavUtils Better than Starting the Activity Yourself?

First, the NavUtils code is short and sweet. Using NavUtils also centralizes the relationships between your activities in the manifest. If those relationships change, you only have to change a line in your manifest rather than mucking about in your Java code.

Another nice thing about it is that the ancestral relationship is kept separate from the code in your fragment. You can use CrimeFragment in a variety of activities, each of which may have a different parent, but CrimeFragment will still do the right thing.

Run CriminalIntent. Create a crime and then press the app icon to return to the list of crimes. It is hard to tell with CriminalIntent's puny two-level hierarchy, but `navigateUpFromSameTask(Activity)` is implementing "up" and sending the user up one level to CrimePagerActivity's parent.

Add an Alternative Menu Item 1/5

Create the Menu Item

A menu item that applies the action bar should not be visible to users without an action bar. So the first step is to create an alternative menu resource. Create a `menu-v11` folder in your project's `res` directory. Then copy and paste `fragment_crime_list.xml` into this folder.

Adding Show Subtitle menu item (`res/menu-v11/fragment_crime_list.xml`)

```
<?xml version="1.0" encoding="utf-8"?>
<menu
    xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menu_item_new_crime"
        android:icon="@android:drawable/ic_menu_add"
        android:title="@string/new_crime"
        android:showAsAction="ifRoom|withText"/>
    <item android:id="@+id/menu_item_show_subtitle"
        android:title="@string/show_subtitle"
        android:showAsAction="ifRoom"/>
</menu>
```

Responding to Show Subtitle menu item (`CrimeListFragment.java`)

```
@TargetApi(11)
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_item_new_crime:
            ...
            return true;
        case R.id.menu_item_show_subtitle:
            getActivity().getActionBar().setSubtitle(R.string.subtitle);
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Add an Alternative Menu Item 2/5

Toggle on the Menu Item

Now the subtitle is visible, but the menu item still reads *Show Subtitle*. It would be better if the menu item toggled its title and function to show or hide the subtitle.

In `onOptionsItemSelected(...)`, check for the presence of a subtitle when this menu item is selected and take appropriate action.

Responding based on subtitle's presence
(`CrimeListFragment.java`)

```
@TargetApi(11)  
@Override  
public boolean onOptionsItemSelected(MenuItem item) {  
    switch (item.getItemId()) {  
        case R.id.menu_item_new_crime:  
            ...  
            return true;  
        case R.id.menu_item_show_subtitle:  
            if (getActivity().getActionBar().getSubtitle() == null) {  
                getActivity().getActionBar().setSubtitle(R.string.subtitle);  
                item.setTitle(R.string.hide_subtitle);  
            } else {  
                getActivity().getActionBar().setSubtitle(null);  
                item.setTitle(R.string.show_subtitle);  
            }  
            return true;  
        default:  
            return super.onOptionsItemSelected(item);  
    }  
}
```

Add an Alternative Menu Item 3/5

Handling Rotation

If you show the subtitle and then rotate, the subtitle will disappear when the interface is created from scratch.

To fix this, you need an instance variable that tracks the subtitle's visibility and you need to retain CrimeListFragment so that the value of this variable will survive rotation.

In **CrimeListFragment.java**, add a boolean member variable and, in `onCreate(...)`, retain CrimeListFragment and initialize the variable.

```
public class CrimeListFragment extends ListFragment {  
    private ArrayList<Crime> mCrimes;  
    private boolean mSubtitleVisible;  
    private final String TAG = "CrimeListFragment";  
  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        ...  
        setRetainInstance(true);  
        mSubtitleVisible = false;  
    }  
}
```

Add an Alternative Menu Item 4/5

Handling Rotation

Then, in `onOptionsItemSelected(...)`, set this variable when you respond to the menu item selection.

Set subtitleVisible in menu item response (CrimeListFragment.java)

```
@TargetApi(11)  
@Override  
public boolean onOptionsItemSelected(MenuItem item) {  
    switch (item.getItemId()) {  
        case R.id.menu_item_new_crime:  
            ...  
        return true;  
    }  
}
```

```
case R.id.menu_item_show_subtitle:  
    if (getActivity().getActionBar().getSubtitle() == null) {  
        getActivity().getActionBar().setSubtitle(R.string.subtitle);  
        mSubtitleVisible = true;  
        item.setTitle(R.string.hide_subtitle);  
    }  
    else {  
        getActivity().getActionBar().setSubtitle(null);  
        mSubtitleVisible = false;  
        item.setTitle(R.string.show_subtitle);  
    }  
    return true;  
default:  
    return super.onOptionsItemSelected(item);  
}  
}
```

Add an Alternative Menu Item 5/5

Now you need to check to see if the subtitle should be shown.

In CrimeListFragment.java, override
onCreateView(...) and set the subtitle if
mSubtitleVisible is true.

```
@TargetApi(11)  
  
    @Override  
  
    public View onCreateView(LayoutInflater inflater, ViewGroup  
        parent, Bundle savedInstanceState) {  
  
        View v = super.onCreateView(inflater, parent,  
        savedInstanceState);  
  
        if (Build.VERSION.SDK_INT >=  
        Build.VERSION_CODES.HONEYCOMB) {  
  
            if (mSubtitleVisible) {  
  
                getActivity().getActionBar().setSubtitle(R.string.subtitle);  
            }  
        }  
  
        return v;  
    }
```

You also need to check the subtitle's state
in onCreateOptionsMenu(...) to make sure you are
displaying the correct menu item title.

Set menu item title based on
mSubtitleVisible is true
(CrimeListFragment.java)

```
    @Override  
  
    public void onCreateOptionsMenu(Menu menu, MenuInflater  
        inflater) {  
  
        super.onCreateOptionsMenu(menu, inflater);  
  
        inflater.inflate(R.menu.fragment_crime_list, menu);  
  
        MenuItem showSubtitle =  
  
        menu.findItem(R.id.menu_item_show_subtitle);  
  
        if (mSubtitleVisible && showSubtitle != null) {  
  
            showSubtitle.setTitle(R.string.hide_subtitle);  
        } }  
    }
```

Run CriminalIntent. Show the subtitle and then
rotate. The subtitle should appear as expected in
the re-created view.

Challenge: An Empty View for the List 1/2

Currently, when `CriminalIntent` launches, it displays an empty list – a big black void. You should give users something to interact with when there are no items in the list.

`ListView`, because it is a subclass of `AdapterView`, supports a special View called the “empty view” for this very scenario. If you specify a view for the empty view, the `ListView` will automatically switch between showing it when there are no items, and showing the list when there are some items.

You can specify the empty view in code using the following `AdapterView` method:

```
public void setEmptyView(View emptyView)
```

Challenge: An Empty View for the List 2/2

Or you can create a layout in XML that specifies both the ListView and the empty view. If you give them resource IDs of `@android:id/list` and `@android:id/empty` respectively, you can tie into the automatic switching feature.

The current implementation of CrimeListFragment does not inflate its own layout in `onCreateView(...)`, but in order to implement the empty view in a layout, it will need to.

Create a layout XML resource for CrimeListFragment that uses a FrameLayout as its root container, with a ListView and another View of some kind as the empty view.

Make the empty view display a message like “There are no crimes.” Add a button to the view that will trigger the creation of a new crime so that, in this case, the user does not have to go to the options menu or action bar.

Android UI

Saving and Loading Local Files

Javed Hasan
BJIT Limited

Lesson Content

CriminalIntent New Feature

- Saving data into local file
- Loading data from local file

Saving Data

- Saving Data using JSON format

Loading Data

- Loading Data with JSON format

Challenge

- Use external storage to save and load data

Saving and Loading Local Files

You will enable CriminalIntent to save and load its data from a JSON file stored on the device's file system.

Each application on an Android device has a directory in its sandbox. Keeping files in the sandbox protects them from being accessed by other applications.

For CriminalIntent, the full path to the sandbox directory is

`/data/data/com.bignerdranch.android.criminalintent`.

Your application can store files in external storage. Typically, this is an SD card. Files and even whole applications can be stored on an SD card. There are security implications to do so. Most importantly, access to files on external storage is not restricted to your application – anyone can read, write, and delete them.

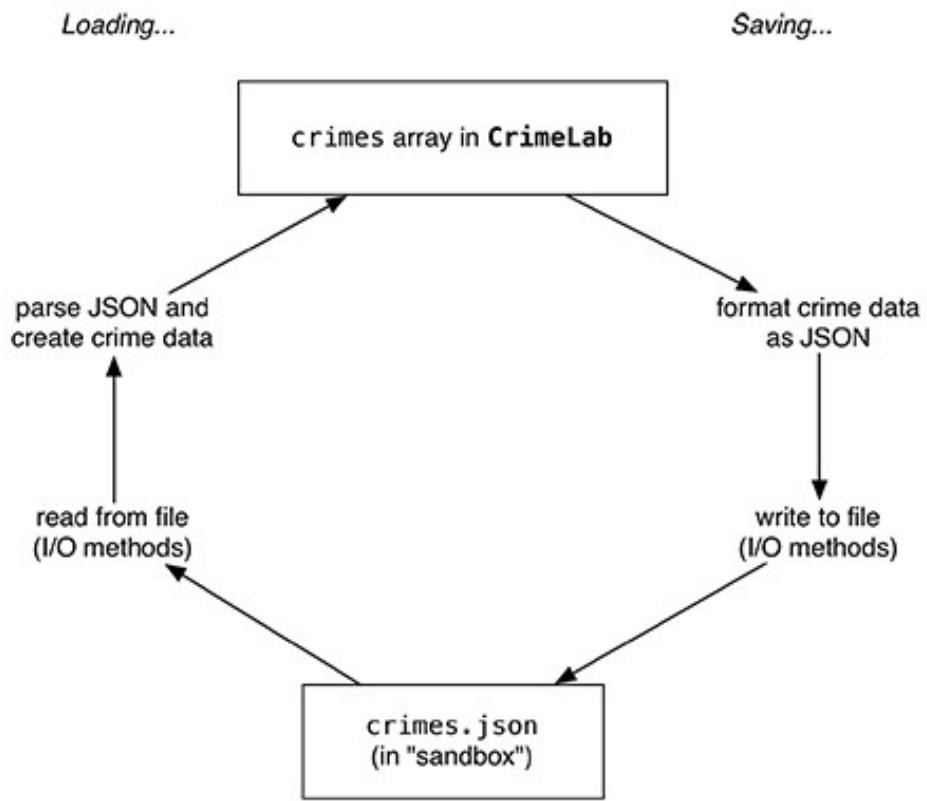
Saving crime data to a JSON file

JSON stands for JavaScript Object Notation, a format that has become popular in recent years, particularly for web services.

Android includes the standard `org.json` package, which has classes that provide simple access to creating and parsing files in JSON format.

The most convenient way to access files available to your application is through I/O methods provided by the Context class.

These methods return instances of standard Java classes like `java.io.File` or `java.io.FileInputStream`.



Saving and Loading Data in CriminalIntent

CrimeLab class will be responsible for triggering the saving and loading of data.

The mechanics of creating and parsing the model objects as JSON will be delegated to a new **CriminalIntentJSONSerializer** class and the existing **Crime** class.

Keep the logic of serializing to JSON in a self-contained unit has a number of advantages.

The class is more easily unit-tested as it does not depend much else to do its work. Also it can be reused in the many places without changes.

```
public class CriminalIntentJSONSerializer {  
    private Context mContext;  
    private String mFilename;  
    public CriminalIntentJSONSerializer(Context c, String f) {  
        mContext = c;  
        mFilename = f;  
    }  
    public void saveCrimes(ArrayList<Crime> crimes)  
        throws JSONException, IOException {  
        // Build an array in JSON  
        JSONArray array = new JSONArray();  
        for (Crime c : crimes)  
            array.put(c.toJSON());  
        // Write the file to disk  
        Writer writer = null;  
        try {  
            OutputStream out = mContext  
                .openFileOutput(mFilename, Context.MODE_PRIVATE);  
            writer = new OutputStreamWriter(out);  
            writer.write(array.toString());  
        } finally {  
            if (writer != null) writer.close();  
        } }}}
```

Making Crime Class Capable of JSON Serialization

In order to save the `mCrimes` array in **JSON** format, you have to be able to save individual instances of **Crime** in **JSON** format.

In `Crime.java`, add the following constants and implementation of `toJSON()` to save a `Crime` in **JSON** format and return an instance of the class **JSONObject** that can be put in a **JSONArray**.

```
public class Crime {  
  
    private static final String JSON_ID = "id";  
    private static final String JSON_TITLE = "title";  
    private static final String JSON_SOLVED = "solved";  
    private static final String JSON_DATE = "date";  
  
    private UUID mId;  
    private String mTitle;  
    private boolean mSolved;  
    private Date mDate = new Date();  
  
    ...  
  
    public JSONObject toJSON() throws JSONException {  
        JSONObject json = new JSONObject();  
        json.put(JSON_ID, mId.toString());  
        json.put(JSON_TITLE, mTitle);  
        json.put(JSON_SOLVED, mSolved);  
        json.put(JSON_DATE, mDate.getTime());  
        return json;  
    }  
  
    ...
```

Saving Crimes in CrimeLab

When is it appropriate to save your data?

A good rule of thumb on mobile applications is: save as often as possible, preferably as soon as changes are made by the user.

The code is already routing the user's changes to the data through the CrimeLab class, so it makes the most sense to save to a file there as well.

```
public class CrimeLab {  
    private static final String TAG = "CrimeLab";  
    private static final String FILENAME = "crimes.json";  
    private ArrayList<Crime> mCrimes;  
    private CriminalIntentJSONSerializer mSerializer;  
    private static CrimeLab sCrimeLab;  
    private Context mApplicationContext;
```

```
    private CrimeLab(Context applicationContext) {  
        mApplicationContext = applicationContext;  
        mCrimes = new ArrayList<Crime>();  
        mSerializer = new CriminalIntentJSONSerializer(mApplicationContext,  
            FILENAME);  
    }  
    ...  
    public void addCrime(Crime c) {  
        mCrimes.add(c);  
    }  
    public boolean saveCrimes() {  
        try {  
            mSerializer.saveCrimes(mCrimes);  
            Log.d(TAG, "crimes saved to file");  
            return true;  
        } catch (Exception e) {  
            Log.e(TAG, "Error saving crimes: ", e);  
            return false;  
        }  
    }  
}
```

Saving Application Data in onPause()

Where should you call **saveCrimes()**?

The **onPause()** lifecycle method is the safest choice. If you wait until **onStop()** or **onDestroy()**, you might miss your chance to save.

A paused activity will be destroyed if the OS needs to reclaim memory, and you cannot count on **onStop()** or **onDestroy()** being called in these circumstances.

Run CriminalIntent. Add a crime or two and then press the device's Home button to pause the activity and save the list of crimes to the file system.

```
...
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case android.R.id.home:
            NavUtils.navigateUpFromSameTask(getActivity());
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
@Override
public void onPause() {
    super.onPause();
    CrimeLab.get(getActivity()).saveCrimes();
}
```

Check LogCat to confirm your success.

Loading Crimes from the Filesystem 1/3

CriminalIntent load crimes from the filesystem when the application is launched.

First, in **Crime.java**, add a constructor that accepts a **JSONObject**.

Next,
in **CriminalIntentJSONSerializer.java**,
add a method
to **CriminalIntentJSONSerializer** for
loading crimes from the filesystem.

Finally, in **CrimeLab's** constructor,
these crimes need to be loaded into
an **ArrayList** when the singleton is first
accessed rather than always creating a
new empty one.

Implementing Crime(JSONObject) (Crime.java)

```
public class Crime {  
  
    ...  
  
    public Crime() {  
        mId = UUID.randomUUID();  
    }  
  
    public Crime(JSONObject json) throws JSONException {  
        mId = UUID.fromString(json.getString(JSON_ID));  
        if (json.has(JSON_TITLE)) {  
            mTitle = json.getString(JSON_TITLE);  
        }  
        mSolved = json.getBoolean(JSON_SOLVED);  
        mDate = new Date(json.getLong(JSON_DATE));  
    }  
  
    public JSONObject toJSON() throws JSONException {  
        JSONObject json = new JSONObject();  
        json.put(JSON_ID, mId.toString());  
        json.put(JSON_TITLE, mTitle);  
        json.put(JSON_SOLVED, mSolved);  
        json.put(JSON_DATE, mDate.getTime());  
        return json;  
    }  
}
```

Loading Crimes from the Filesystem 2/3

Implementing loadCrimes() (CriminalIntentJSONSerializer.java)

```
public CriminalIntentJSONSerializer(Context c, String f) {  
    mContext = c;  
    mFilename = f;  
}  
  
public ArrayList<Crime> loadCrimes() throws IOException,  
JSONException {  
    ArrayList<Crime> crimes = new ArrayList<Crime>();  
    BufferedReader reader = null;  
  
    try {  
        // Open and read the file into a StringBuilder  
        InputStream in = mContext.openFileInput(mFilename);  
        reader = new BufferedReader(new  
InputStreamReader(in));  
  
        StringBuilder jsonString = new StringBuilder();  
        String line = null;  
        while ((line = reader.readLine()) != null) {  
            // Line breaks are omitted and irrelevant  
            jsonString.append(line);  
        }  
    }
```

```
    // Parse the JSON using JSONTokener  
    JSONArray array = (JSONArray) new  
JSONTokener(jsonString.toString())  
        .nextValue();  
  
    // Build the array of crimes from JSONObjects  
    for (int i = 0; i < array.length(); i++) {  
        crimes.add(new Crime(array.getJSONObject(i)));  
    }  
    } catch (FileNotFoundException e) {  
        // Ignore this one; it happens when starting fresh  
    } finally {  
        if (reader != null)  
            reader.close();  
    }  
    return crimes;  
}  
  
public void saveCrimes(ArrayList<Crime> crimes) throws  
JSONException, IOException {  
    // Build an array in JSON  
    JSONArray array = new JSONArray();  
    for (Crime c : crimes)  
        array.put(c.toJSON());  
    ...  
}
```

Loading Crimes from the Filesystem 3/3

Loading crimes (CrimeLab.java)

```
public CrimeLab(Context applicationContext) {  
    mApplicationContext = applicationContext;  
    mSerializer = new  
    CriminalIntentJSONSerializer(mApplicationContext, FILENAME);  
    mCrimes = new ArrayList<Crime>();  
    try {  
        mCrimes = mSerializer.loadCrimes();  
    } catch (Exception e) {  
        mCrimes = new ArrayList<Crime>();  
        Log.e(TAG, "Error loading crimes:", e);  
    }  
}  
  
public static CrimeLab get(Context c) {  
    ...  
}  
...  
}
```

Here you attempt to load crimes; if there are none, then you create a new empty list.

Now `CriminalIntent` can maintain its data between launches of the application.

You can test this in a number of ways. Run the app, add crimes, and make changes. Then navigate to some other application, like the web browser. At this point, `CriminalIntent` will probably get shut down by the OS.

Launch it again, and see that your changes persist. You can also force it to stop and relaunch from Eclipse.

Challenge: Use External Storage

Saving files on internal storage is the right choice for most applications, especially when data privacy is a concern. Some apps, however, might want to write to the device's external storage if it is available. This can be useful if you are storing data that you want to share with other apps or the user, like music, photos or downloads from the web. External storage is almost always more plentiful, too, which makes it the right choice for large media like videos.

To write to external storage, you need to do two things.

First, check to ensure that the external storage is available.

The **android.os.Environment** class has several helpful methods and constants that you can use to determine this.

Second, get a handle on the external files directory (look for a method in **Context** that can give this to you). The rest of the work is basically the same as what you are already doing in **CriminalIntentJSONSerializer**.

Android UI

Context Menus & Contextual Action Mode

Javed Hasan
BJIT Limited

Lesson Content

CriminalIntent New Feature

- Context Menus
- Contextual Action Mode

Define a Context Menu Resource

Implement a Floating Context Menu

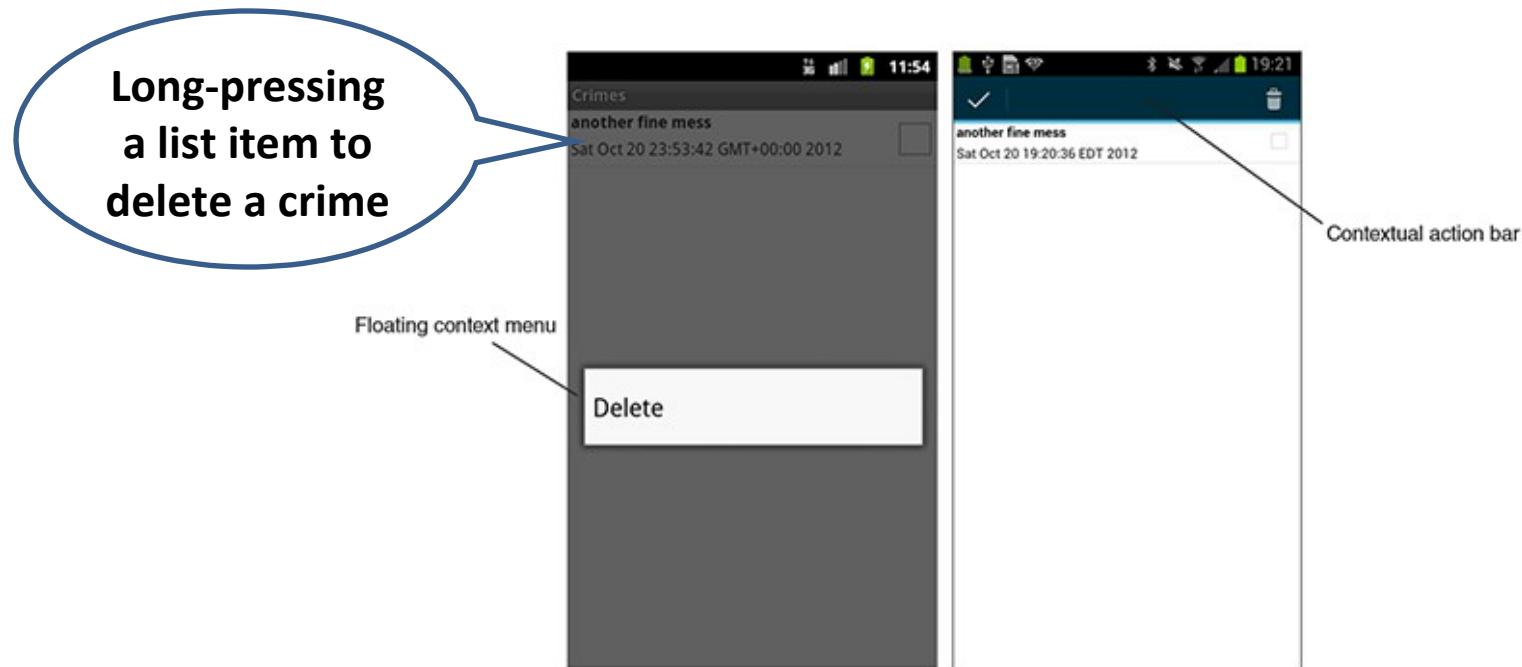
- Create the Menu, Register it
- Responding to an Action

Implement Contextual Action Mode

- Enable multiple selection
- Action mode callbacks in a list view

New Feature Overview

You will enable users to delete crimes from the list by long-pressing a list item. Deleting a crime is a contextual action: it is associated with a particular view on screen (a single list item) rather than the entire screen.



On pre-Honeycomb devices, contextual actions are presented in a floating context menu. On later devices, a contextual action bar is the recommended way to present contextual actions. The contextual action bar appears on top of the activity's action bar and presents actions to the user.

Defining a Context Menu Resource

In res/menu/, create a new Android XML file named crime_list_item_context.xml and give it a menu root element.

The crime list context menu
(crime_list_item_context.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<menu
    xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menu_item_delete_crime"
        android:icon="@android:drawable/ic_menu_delete"
        android:title="@string/delete_crime" />
</menu>
```

Implement Floating Context Menu

You inflate a context menu using the following method:

```
public void onCreateContextMenu(ContextMenu menu, View v, ContextMenu.ContextMenuItemInfo menuInfo)
```

To respond to the user's context menu selection, you implement this Fragment method:

```
public boolean onContextItemSelected(MenuItem item)
```

Creating the context menu (CrimeListFragment.java)

```
@Override
```

```
public boolean onOptionsItemSelected(MenuItem item) {
```

```
    switch (item.getItemId()) {
```

```
    ...
```

```
    default:
```

```
        return super.onOptionsItemSelected(item);
```

```
}
```

```
}
```

```
@Override
```

```
public void onCreateContextMenu(ContextMenu menu, View v, ContextMenu.ContextMenuItemInfo menuInfo) {
```

```
    getActivity().getMenuInflater().inflate(R.menu.crime_list_item_context, menu);
```

```
}
```

Registering for the Context Menu

By default, a long-press on a view does not trigger the creation of a context menu.

You must register a view for a floating context menu by calling the following Fragment method:

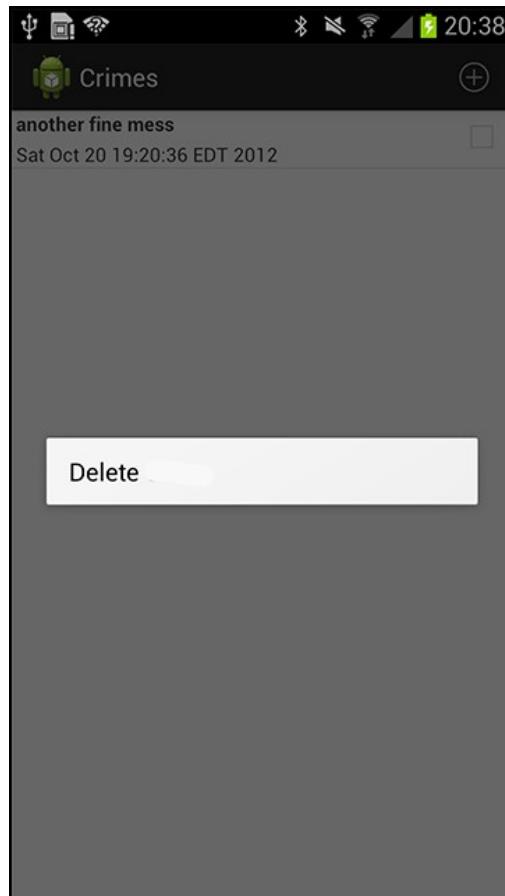
public void registerForContextMenu(View view)
and passing in the view in question.

In **CrimeListFragment.onCreateView(...)**,
get a reference to the ListView and register it.

```
@Override  
public View onCreateView(LayoutInflater inflater, ViewGroup parent,  
    Bundle savedInstanceState) {  
    View v = super.onCreateView(inflater, parent, savedInstanceState);  
  
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {  
        if (mSubtitleVisible) {  
            getActivity().getActionBar().setSubtitle(R.string.subtitle);  
        }  
    }  
  
    ListView listView = (ListView)v.findViewById(android.R.id.list);  
    registerForContextMenu(listView);  
  
    return v;  
}
```

Registering for the Context Menu

Run CriminalIntent, long-press a list item, and confirm that the floating context menu with the Delete Crime menu item appears.



Responding to an Action 1/2

To wire up the menu item, you need a method that can delete a crime from the model. InCrimeLab.java, add a `deleteCrime(Crime)` method.

```
public void addCrime(Crime c) {  
    mCrimes.add(c);  
}  
  
public void deleteCrime(Crime c) {  
    mCrimes.remove(c);  
}
```

The next step is to respond to the menu item selection in **onContextItemSelected(MenuItem)**. The MenuItem has an ID that represents the menu item that was selected. However, it is not enough to know that the user wants to delete a crime. You also need to know *which crime to delete*.

You can get this information by calling **getMenuInfo()** on the MenuItem. This method returns an instance of a class that implements **ContextMenu.ContextMenuItem**.

Responding to an Action 2/2

In **CrimeListFragment**, add an implementation for **onContextItemSelected(MenuItem)** that uses the menu info and the adapter to determine which Crime was long-pressed. Then delete that Crime from the model.

Run CriminalIntent, add a new crime, then long-press to delete it. (To simulate a long-press on the emulator, hold the left mouse button down until the menu appears.)

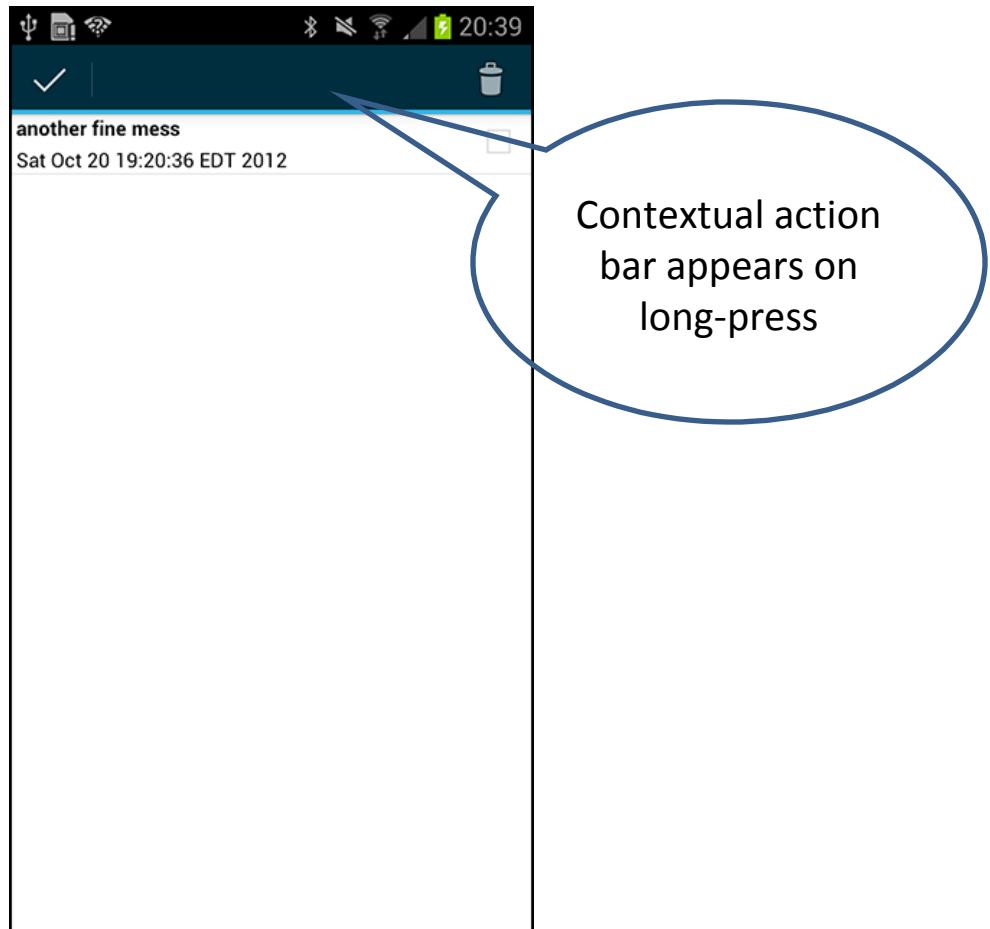
```
@Override  
public void onCreateContextMenu(ContextMenu menu, View v,  
ContextMenuInfo menuInfo) {  
    getActivity().getMenuInflater().inflate(R.menu.crime_list_item_context,  
menu);  
}  
  
@Override  
public boolean onContextItemSelected(MenuItem item) {  
    AdapterContextMenuInfo info =  
(AdapterContextMenuInfo)item.getMenuInfo();  
    int position = info.position;  
    CrimeAdapter adapter = (CrimeAdapter) getListAdapter();  
    Crime crime = adapter.getItem(position);  
    switch (item.getItemId()) {  
        case R.id.menu_item_delete_crime:  
            CrimeLab.get(getActivity()).deleteCrime(crime);  
            adapter.notifyDataSetChanged();  
            return true;  
    }  
    return super.onContextItemSelected(item);  
}
```

Implementing Contextual Action Mode

The recommended way to present contextual actions on newer devices is to have a long-press on a view put the screen into contextual action mode.

When the screen is in contextual action mode, the items defined in your context menu appear in a contextual action bar that overlays the action bar.

The contextual action bar code contains classes and methods that are not available on Froyo or Gingerbread, so you must ensure that the code that you write will not be called where it does not exist.



Enabling Multiple Selection

When a list view is in contextual action mode, you can enable selecting multiple list items at one time.

Any action chosen from the contextual action bar will apply to all of the views that have been selected.

In **CrimeListFragment.onCreateView(...)**, set the list view's choice mode to **CHOICE_MODE_MULTIPLE_MODAL**.

Use build version constants to separate the code that registers the ListView from the code that sets the choice mode.

Setting choice mode (**CrimeListFragment.java**)

```
@TargetApi(11)  
@Override  
public View onCreateView(LayoutInflater inflater, ViewGroup parent,  
    Bundle savedInstanceState) {  
    View v = super.onCreateView(inflater, parent, savedInstanceState);  
    ...  
    ListView listView = (ListView)v.findViewById(android.R.id.list);  
    if (Build.VERSION.SDK_INT < Build.VERSION_CODES.HONEYCOMB) {  
        // Use floating context menus on Froyo and Gingerbread  
        registerForContextMenu(listView);  
    } else {  
        // Use contextual action bar on Honeycomb and higher  
  
        listView.setChoiceMode(ListView.CHOICE_MODE_MULTIPLE_MODAL);  
    }  
    return v;  
}
```

Action Mode Callbacks in a List View 1/4

The next step is to set a listener on the ListView that implements **AbsListView.MultiChoiceModeListener**. This interface contains the following method that calls back when a view has been selected or deselected.

```
public abstract void onItemCheckedStateChanged(ActionMode mode, int position, long id, boolean checked)
```

MultiChoiceModeListener implements another interface – **ActionMode.Callback**. When the screen is put into contextual action mode, an instance of the ActionMode class is created, and the methods in **ActionMode.Callback** call back at different points in the lifecycle of the ActionMode.

There are four required methods in **ActionMode.Callback**:

```
public abstract boolean onCreateActionMode(ActionMode mode, Menu menu)
```

Called when the ActionMode is created. This is where you inflate the context menu resource to be displayed in the contextual action bar.

Action Mode Callbacks in a List View 2/4

public abstract boolean onPrepareActionMode(ActionMode mode, Menu menu)

Called after `onCreateActionMode(...)` and whenever an existing contextual action bar needs to be refreshed with new data.

public abstract boolean onActionItemClicked(ActionMode mode, MenuItem item)

Called when the user selects an action. This is where you respond to contextual actions defined in the menu resource.

public abstract void onDestroyActionMode(ActionMode mode)

Called when the ActionMode is about to be destroyed because the user has canceled the action mode or the selected action has been responded to. The default implementation results in the view(s) being unselected.

In `CrimeListFragment.onCreateView(...)`, set a listener that implements `MultiChoiceModeListener` on the list view. You only need to take action in `onCreateActionMode(...)` and in `onActionItemClicked(ActionMode, MenuItem)`.

Action Mode Callbacks in a List View 3/4

Setting the MultiChoiceModeListener (CrimeListFragment.java)

```
...
} else {
    // Use contextual action bar on Honeycomb and higher
    listView.setChoiceMode(ListView.CHOICE_MODE_MULTIPLE_MODAL);
    listView.setMultiChoiceModeListener(new MultiChoiceModeListener() {

        public void onItemCheckedStateChanged(ActionMode mode, int position,
            long id, boolean checked) {
            // Required, but not used in this implementation
        }

        // ActionMode.Callback methods

        public boolean onCreateActionMode(ActionMode mode, Menu menu) {
            MenuInflater inflater = mode.getMenuInflater();
            inflater.inflate(R.menu.crime_list_item_context, menu);
            return true;
        }

        public boolean onPrepareActionMode(ActionMode mode, Menu menu) {
            return false;
            // Required, but not used in this implementation
        }
    });
}
```

Action Mode Callbacks in a List View 3/4

```
public boolean onActionItemClicked(ActionMode mode, MenuItem item) {  
    switch (item.getItemId()) {  
        case R.id.menu_item_delete_crime:  
            CrimeAdapter adapter = (CrimeAdapter) getListAdapter();  
            CrimeLab crimeLab = CrimeLab.get(getActivity());  
            for (int i = adapter.getCount() - 1; i >= 0; i--) {  
                if (getListView().isItemChecked(i)) {  
                    crimeLab.deleteCrime(adapter.getItem(i));  
                }  
            }  
            mode.finish();  
            adapter.notifyDataSetChanged();  
            return true;  
        default:  
            return false;  
    }  
}  
  
public void onDestroyActionMode(ActionMode mode) {  
    // Required, but not used in this implementation  
}  
});  
}  
return v;}
```

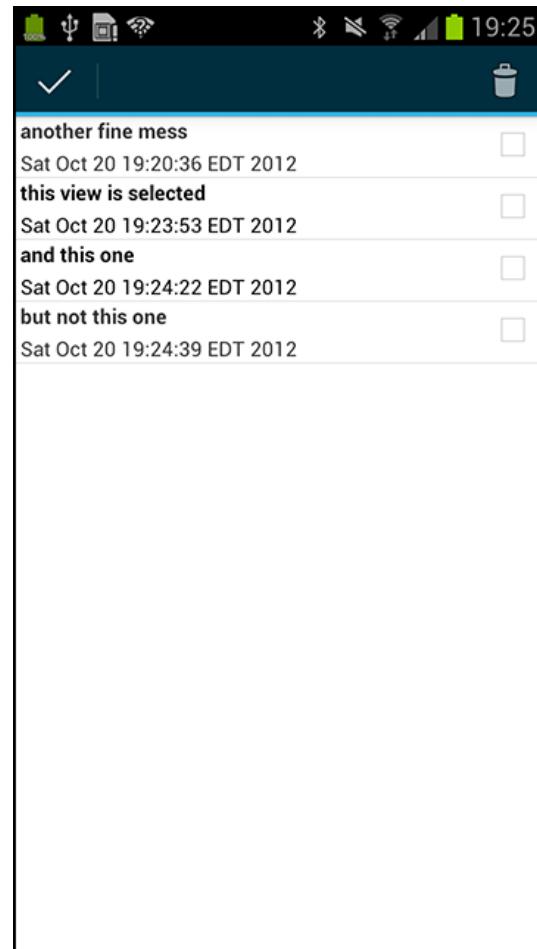
Check Context Action Mode Delete Function

Run CriminalIntent. Long-press a list item to select it and enter contextual action mode. Select other list items by (regular) pressing them. Unselect selected list items in the same way.

Delete a couple of crimes. Pressing the delete icon will finish the action mode and return you to the updated list of crimes.

Alternatively, you can cancel the action mode by pressing the icon on the far left of the contextual action bar.

This will finish the action mode and return you to the list without making any changes.



Changing Activated Item Backgrounds 1/2

Sometimes you need a view to appear differently when it is in the activated state. When a view is in the activated state, it has been marked by the user as being something interesting.

You can change the background based on the view's state using a state list drawable. A state list drawable is an XML resource.

Within this resource, you name a drawable (a bitmap or a color) and then list which states this drawable should appear in.

(You can find other states a view can be in on the **StateListDrawable** reference page.).

A simple state list drawable
(res/drawable/background_activated.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<selector
    xmlns:android="http://schemas.android.com/apk/res/android" >
    <item
        android:state_activated="true"
        android:drawable="@android:color/darker_gray"
    />
</selector>
```

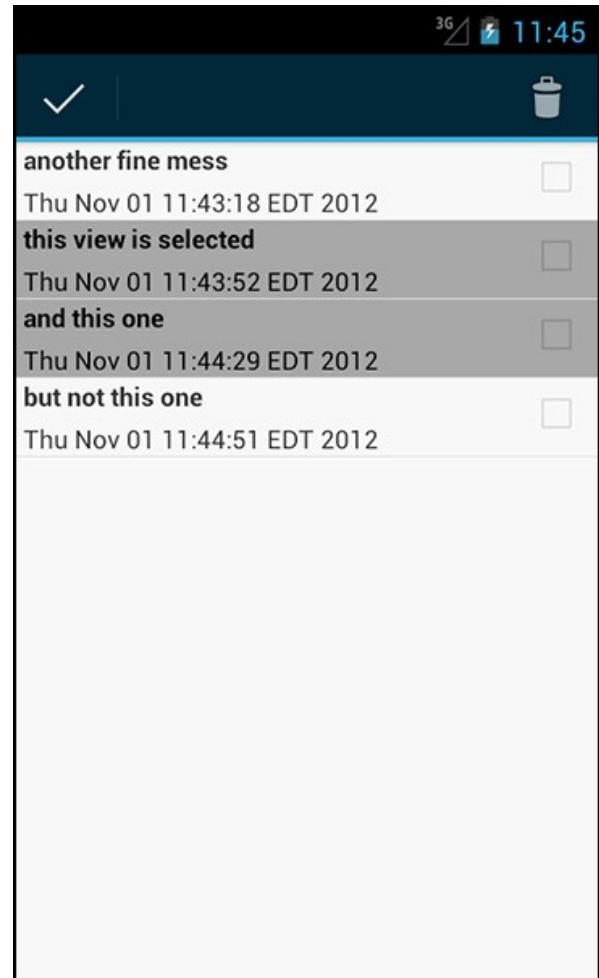
This XML says, “When the view that references this drawable is in the activated state, use the value of `android:drawable`. Otherwise, do nothing.”

Changing Activated Item Backgrounds 2/2

Change list item background
(res/layout/list_item_crime.xml)

```
<RelativeLayout  
    xmlns:android="http://schemas.android.com/apk/res/andr  
    oid"  
  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
  
    android:background="@drawable/background_activated"  
>  
...  
</RelativeLayout>
```

Run CriminalIntent again. This time, your selected items will stand out clearly



Challenge: Deleting from CrimeFragment

Users might appreciate the ability to delete a crime from its detail view as well as from the list. In that case, deleting would apply to the screen as a whole, so the action would belong in an options menu or on the action bar.

Implement a **delete crime option** in CrimeFragment.

Android UI

Camer I: Viewfinder

Javed Hasan
BJIT Limited

Lesson Content

CriminalIntent New Feature and Design Aspects

- Live Camera Preview in Viewfinder
- Object Diagram

Creating Camera UI

- Creating Camera Fragment Layout
- Creating Crime Camera Fragment
- Creating Crime Camera Activity

Using Camera API

- Open and Release Camera
- Surface, SurfaceHolder, SurfaceView
- Determine Preview Size

Starting CrimeCameraActivity from CrimeFragment

- Hide the Status Bar and Action Bar

New Feature Overview

You will create a fragment-based activity and use the **SurfaceView** class and the camera hardware to display a live video preview from the camera.



Live camera preview in viewfinder

Object Diagram for Camera Portion of CriminalIntent

The **Camera** provides hardware-level access to the device's camera(s).

A camera is an exclusive-access resource: only one activity can access a camera at a time.

The instance of **SurfaceView** will be your viewfinder.

A **SurfaceView** is a special view that lets you render content directly to the screen.

Model

Controller

View

CrimeFragment

CrimeCameraActivity

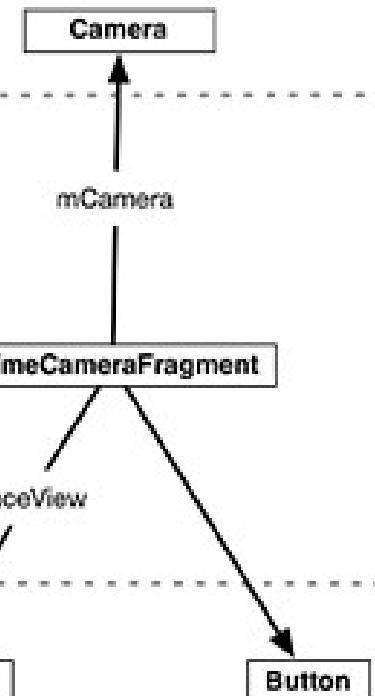
starts

CrimeCameraFragment

mSurfaceView

SurfaceView

Button



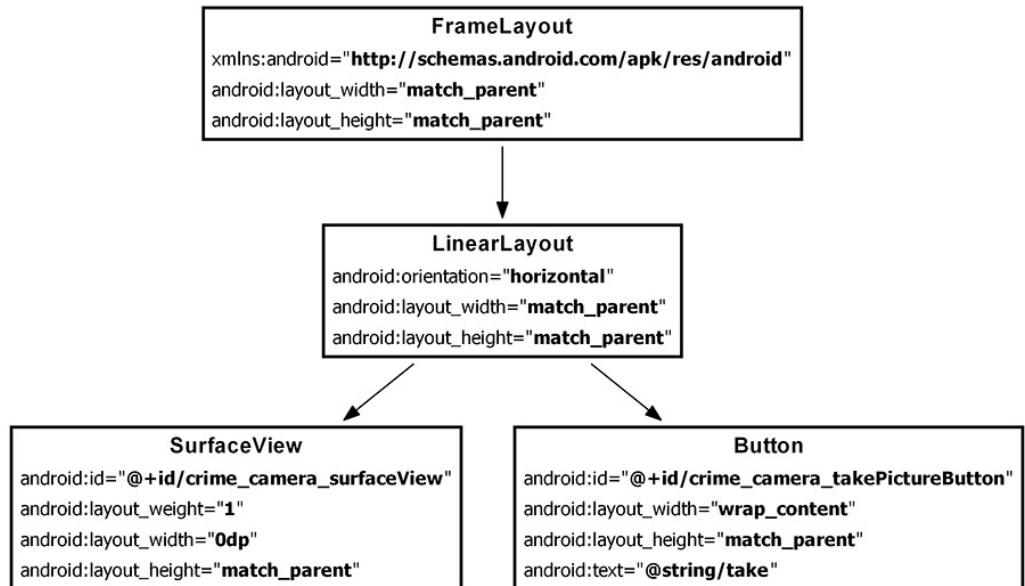
Camera UI: Creating the Fragment Layout 1/2

In **LinearLayout**, you use a combination of `layout_width` and `layout_weight` to arrange its child views.

The Button is given the space it needs due to its `android:layout_width="wrap_content"` attribute, and the SurfaceView gets nothing (`android:layout_width="0dp"`).

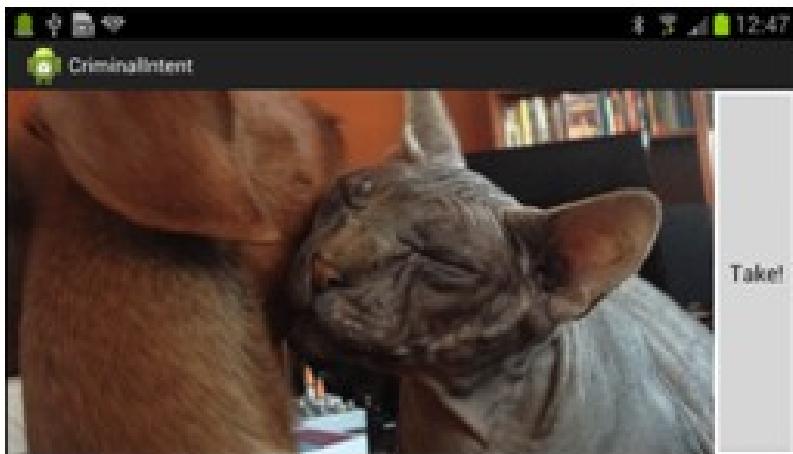
In terms of the space left over, only the SurfaceView has a `layout_weight` attribute, so the SurfaceView gets all of the leftover space.

Layout for CrimeCameraFragment
(`fragment_crime_camera.xml`)



Camera UI: Creating the Fragment Layout 2/2

A viewfinder and button



In strings.xml, add a string resource for the button's text.

Adding string for camera button (strings.xml)

```
...
<string name="show_subtitle">Show Subtitle</string>
<string name="subtitle">Sometimes tolerance is not a
virtue.</string>
<string name="take">Take!</string>
</resources>
```

Camera UI: Creating CrimeCameraFragment

Create a new class named CrimeCameraFragment and make its superclass android.support.v4.app.Fragment.

Then, override onCreateView(...) to inflate the layout and get references to the widgets.

For now, set a listener on the button that simply finishes the hosting activity, which will return the user to the previous screen.

The initial camera fragment (CrimeCameraFragment.java)

```
public class CrimeCameraFragment extends Fragment {  
    private static final String TAG = "CrimeCameraFragment";  
    private Camera mCamera;  
    private SurfaceView mSurfaceView;  
    @Override  
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,  
        Bundle savedInstanceState) {  
        View v = inflater.inflate(R.layout.fragment_crime_camera, parent, false);  
        Button takePictureButton = (Button)v  
            .findViewById(R.id.crime_camera_takePictureButton);  
        takePictureButton.setOnClickListener(new View.OnClickListener() {  
            public void onClick(View v) {  
                getActivity().finish();  
            }  
        });  
        mSurfaceView = (SurfaceView)v.findViewById(R.id.crime_camera_surfaceView);  
        return v;  
    }  
}
```

Camera UI: Creating CrimeCameraActivity

Create the camera activity (CrimeCameraActivity.java)

```
public class CrimeCameraActivity extends  
SingleFragmentActivity {  
  
    @Override  
    protected Fragment createFragment() {  
        return new CrimeCameraFragment();  
    }  
}
```

Adding permissions and camera activity to manifest (AndroidManifest.xml)

```
<?xml version="1.0" encoding="utf-8"?>  
<manifest  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    package="com.bignerdranch.android.criminalintent"  
    android:versionCode="1"  
    android:versionName="1.0" >  
  
    <uses-sdk android:minSdkVersion="8"  
            android:targetSdkVersion="16"/>
```

```
    <uses-permission android:name="android.permission.CAMERA" />  
  
    <uses-feature android:name="android.hardware.camera" />  
  
    <application  
        ...  
        <activity android:name=".CrimeCameraActivity"  
                android:screenOrientation="landscape"  
                android:label="@string/app_name">  
            </activity>  
        </application>  
    </manifest>
```

The `uses-feature` element specifies that your application uses a particular device feature. Adding the `android.hardware.camera` feature ensures that when your app appears on Google Play, it will only be offered to devices that have a camera.

Opening and Releasing Camera

Opening the camera in onResume() (CrimeCameraFragment.java)

```
@TargetApi(9)  
 @Override  
  
 public void onResume() {  
     super.onResume();  
     if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.GINGERBREAD) {  
         mCamera = Camera.open(0);  
     } else {  
         mCamera = Camera.open();  
     }  
 }
```

Releasing the camera in onPause() (CrimeCameraFragment.java)

```
@Override  
  
 public void onPause() {  
     super.onPause();  
     if (mCamera != null) {  
         mCamera.release();  
         mCamera = null;  
     }  
 }
```

You are going to initialize the camera using the static method `Camera.open(int)` and pass in 0 to open the first camera available on the device.

Usually this is the rear-facing camera, but if the device does not have one (e.g. the Nexus 7), then it will open its front-facing camera.

For API level 8, you need to call the parameterless `Camera.open()` instead. Safeguard your code for Froyo by checking the device's build version and calling `Camera.open()` on API level 8.

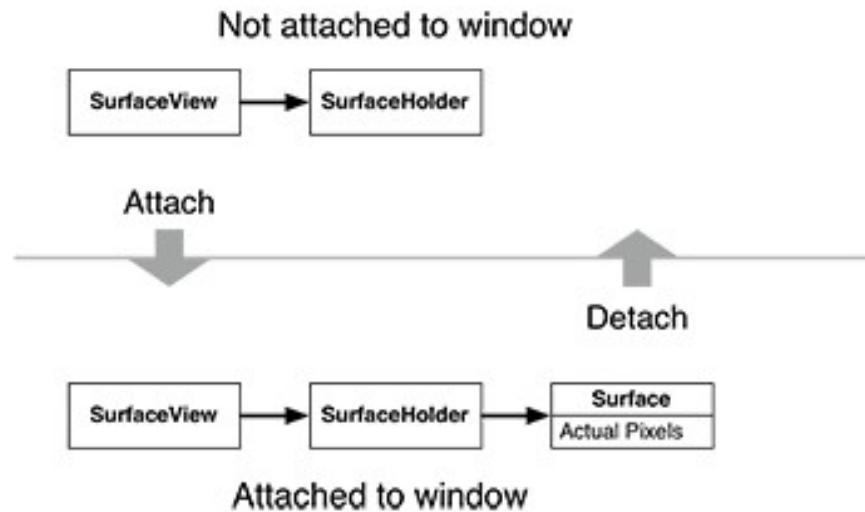
SurfaceView, SurfaceHolder and Surface

A **Surface** represents a buffer of raw pixel data.

SurfaceView provides a dedicated drawing surface embedded inside of a view hierarchy. SurfaceView takes care of placing the surface at the correct location on the screen.

SurfaceView provides an implementation of the **SurfaceHolder** interface. A SurfaceHolder is your connection to the object – Surface.

A **Surface** is created when the SurfaceView appears on screen and destroyed when the SurfaceView is no longer visible. Make sure nothing is drawn to the Surface when it does not exist.



Getting the SurfaceHolder

Getting the SurfaceHolder (CrimeCameraFragment.java)

```
@Override  
@SuppressLint("NewApi")  
public View onCreateView(LayoutInflater inflater, ViewGroup parent,  
    Bundle savedInstanceState) {  
    ...  
    mSurfaceView =  
        (SurfaceView)v.findViewById(R.id.crime_camera_surfaceView);  
    SurfaceHolder holder = mSurfaceView.getHolder();  
    // setType() and SURFACE_TYPE_PUSH_BUFFERS are both  
    // deprecated,  
    // but are required for Camera preview to work on pre-3.0 devices.  
    holder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);  
    return v;  
}
```

The `setType(...)` method and the `SURFACE_TYPE_PUSH_BUFFERS` constant are both deprecated.

But these are required for the camera preview to work on pre-Honeycomb devices.

You make these warnings go away with the `@SuppressLint` annotation.

Surface and Its Client Camera

A Surface's client is the object that wants to draw into its buffer.

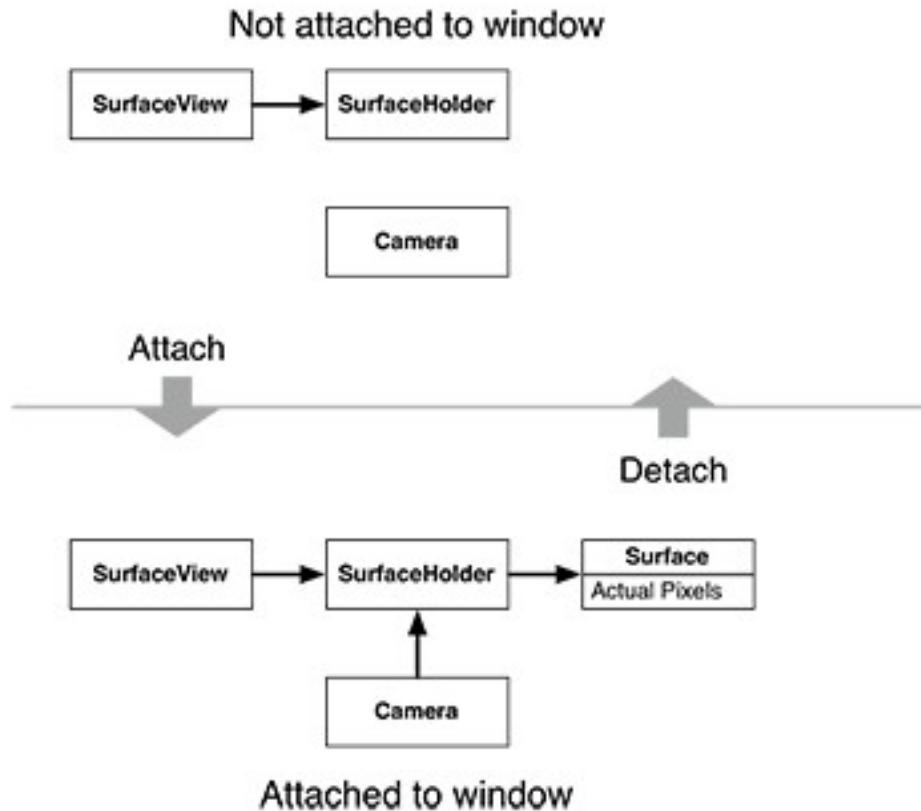
In **CrimeCameraFragment**, the client is the Camera instance.

You have to make sure that nothing is drawn to the Surface when it is not there.

You will need to attach the Camera to your SurfaceHolder when the Surface is created, and detach it when the Surface is destroyed.

SurfaceHolder provides another interface that will let you do this – **SurfaceHolder.Callback**.

This interface listens for events in the lifecycle of a Surface so that you can coordinate the Surface with its client.



Coordinate Surface and Camera 1/2

```
public abstract void surfaceCreated(SurfaceHolder holder)
```

Called when the view hierarchy that the SurfaceView belongs to is put on screen. This is where you connect the Surface with its client.

```
public abstract void surfaceChanged(SurfaceHolder holder, int format, int width, int height)
```

When the surface is being displayed for the first time, the surfaceChanged(...) method will be called. This method informs you of the pixel format and the width and height of the surface. Within this method, you tell your Surface's client how big the drawing area will be.

```
public abstract void surfaceDestroyed(SurfaceHolder holder)
```

When the SurfaceView is removed from the screen, the Surface is destroyed. This will be where you tell your Surface's client to stop using the Surface.

Coordinate Surface and Camera 2/2

Here are the Camera methods you will use to respond to events in the lifecycle of the Surface:

```
public final void setPreviewDisplay(SurfaceHolder holder)
```

This method connects the camera with your Surface. You will call it in `surfaceCreated()`.

```
public final void startPreview()
```

This method starts drawing frames on the Surface. You will call it in `surfaceChanged(...)`.

```
public final void stopPreview()
```

This method stops drawing frames on the Surface. You will call it in `surfaceDestroyed()`.

Implementing SurfaceHolder.Callback

Implementing SurfaceHolder.Callback (CrimeCameraFragment.java)

```
...  
  
SurfaceHolder holder = mSurfaceView.getHolder();  
  
// setType() and SURFACE_TYPE_PUSH_BUFFERS are both deprecated,  
// but are required for Camera preview to work on pre-3.0 devices.  
holder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);  
holder.addCallback(new SurfaceHolder.Callback() {  
  
    public void surfaceCreated(SurfaceHolder holder) {  
  
        // Tell the camera to use this surface as its preview area  
        try {  
            if (mCamera != null) {  
                mCamera.setPreviewDisplay(holder);  
            }  
        } catch (IOException exception) {  
            Log.e(TAG, "Error setting up preview display", exception);  
        }  
    }  
  
    public void surfaceDestroyed(SurfaceHolder holder) {  
  
        // We can no longer display on this surface, so stop the preview.  
        if (mCamera != null) {  
            mCamera.stopPreview();  
        }  
    }  
}
```

```
    public void surfaceChanged(SurfaceHolder holder, int format, int w, int h)  
    {  
        if (mCamera == null) return;  
  
        // surface has changed size; update the camera preview size  
        Camera.Parameters parameters = mCamera.getParameters();  
        Size s = null; // To be reset in the next section  
        parameters.setPreviewSize(s.width, s.height);  
        mCamera.setParameters(parameters);  
        try {  
            mCamera.startPreview();  
        } catch (Exception e) {  
            Log.e(TAG, "Could not start preview", e);  
            mCamera.release();  
            mCamera = null;  
        }  
    }  
});  
return v;  
}
```

Determine the Camera Preview Size 1/2

The first step is to get a list of the camera's allowable preview sizes from the nested `Camera.Parameters` class. This class includes the following method:

```
public List<Camera.Size> getSupportedPreviewSizes()
```

This method returns a list of instances of the **android.hardware.Camera.Size** class, which wraps the width and height dimensions of an image.

You can then compare the sizes in the list with the width and height of the Surface passed into `surfaceChanged(...)` to find a preview size that will work with your Surface.

In **CrimeCameraFragment**, add the following method that accepts a list of sizes and then selects the size with the largest number of pixels. It is not beautiful code, but it will work fine for our purposes.

Determine the Camera Preview Size 2/2

Find the best supported size (CrimeCameraFragment.java)

```
/** A simple algorithm to get the largest size available. For a more
 * robust version, see CameraPreview.java in the ApiDemos
 *
 * sample app from Android. */

private Size getBestSupportedSize(List<Size> sizes, int width, int
height) {

    Size bestSize = sizes.get(0);

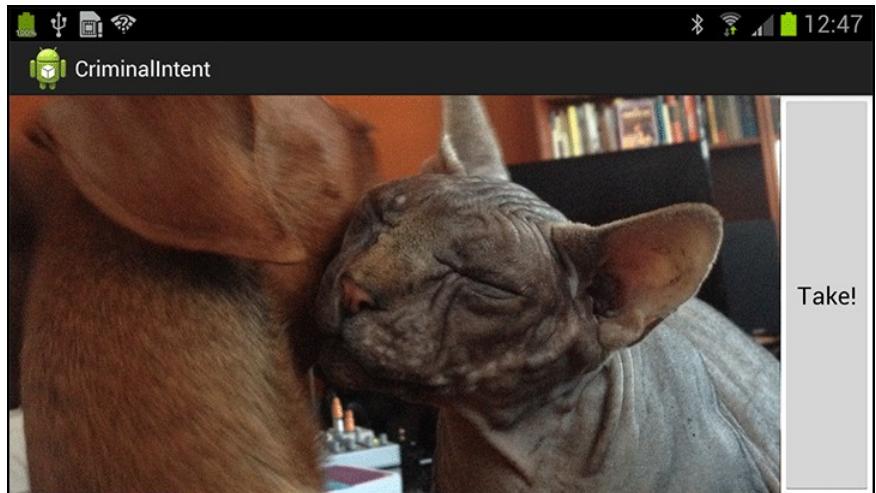
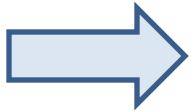
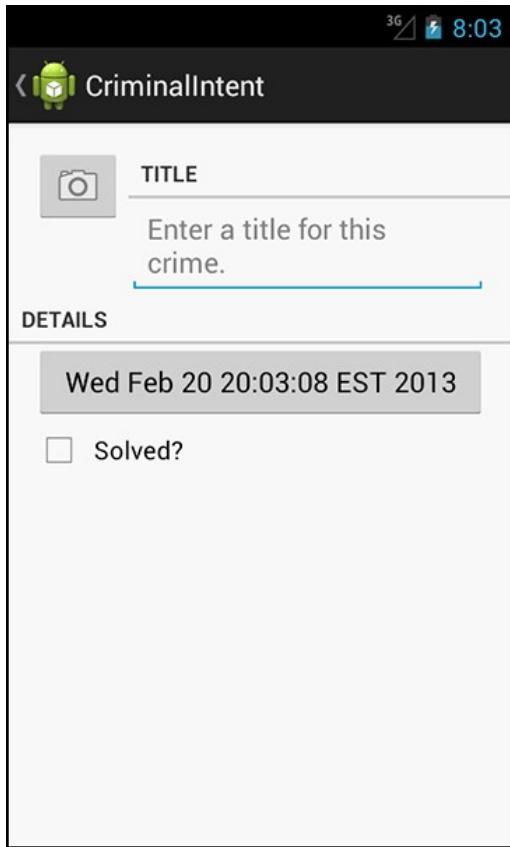
    int largestArea = bestSize.width * bestSize.height;
    for (Size s : sizes) {
        int area = s.width * s.height;
        if (area > largestArea) {
            bestSize = s;
            largestArea = area;
        }
    }
    return bestSize;
}
```

Now call this method to set the preview size in
surfaceChanged(...).

Calling getBestSupportedSize(...) (CrimeCameraFragment.java)

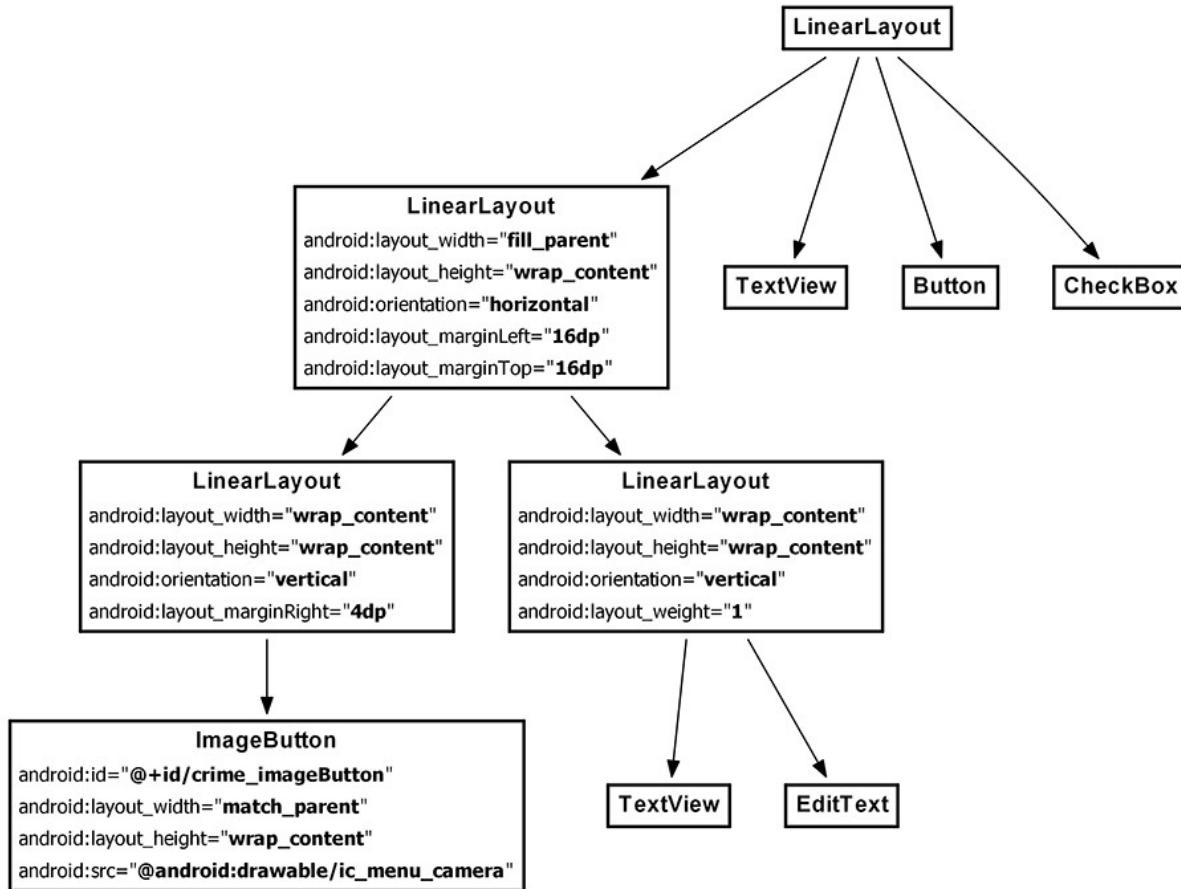
```
...
holder.addCallback(new SurfaceHolder.Callback() {
    ...
    public void surfaceChanged(SurfaceHolder holder, int format, int w, int
h) {
        // The surface has changed size; update the camera preview size
        Camera.Parameters parameters = mCamera.getParameters();
        Size s = null;
        Size s =
        getBestSupportedSize(parameters.getSupportedPreviewSizes(), w, h);
        parameters.setPreviewSize(s.width, s.height);
        mCamera.setParameters(parameters);
        try {
            mCamera.startPreview();
        } catch (Exception e) {
            Log.e(TAG, "Could not start preview", e);
            mCamera.release();
            mCamera = null;
        }
    }
});
```

Start CrimeCameraActivity from CrimeFragment



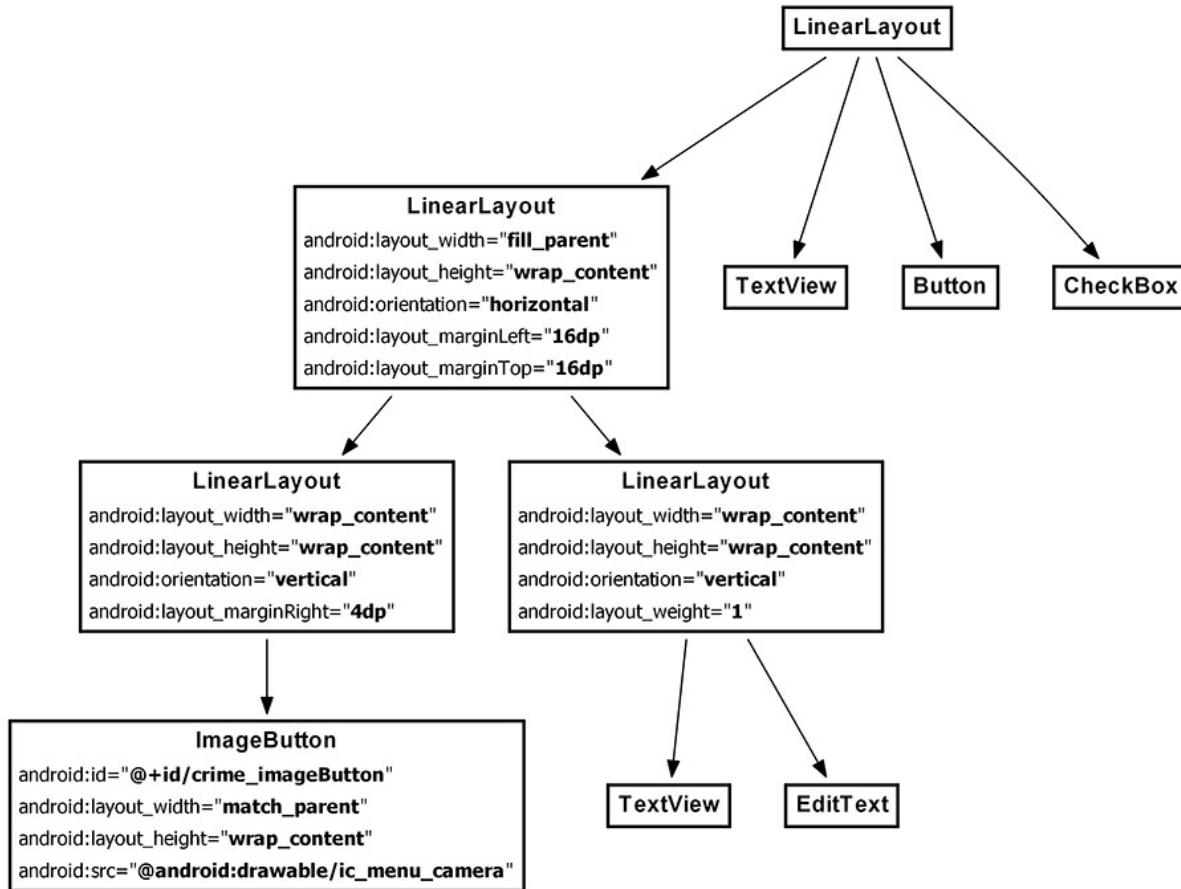
Add Camera Button in CrimeFragment 1/2

Adding camera button and more organizing (layout/fragment_crime.xml)



Add Camera Button in CrimeFragment 2/2

Adding camera button and more organizing (layout-land/fragment_crime.xml)



Starting CrimeCameraActivity

(CrimeFragment.java)

```
public class CrimeFragment extends Fragment {  
    ...  
    private ImageButton mPhotoButton;  
    ...  
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,  
        Bundle savedInstanceState) {  
        View v = inflater.inflate(R.layout.fragment_crime, parent, false);  
        ...  
        mPhotoButton = (ImageButton)v.findViewById(R.id.crime_imageButton);  
        mPhotoButton.setOnClickListener(new View.OnClickListener() {  
            @Override  
            public void onClick(View v) {  
                Intent i = new Intent(getActivity(), CrimeCameraActivity.class);  
                startActivity(i);  
            }  
        });  
  
        return v;  
    }
```

Checking for a Camera on the Device

(CrimeFragment.java)

Checking for a camera (CrimeFragment.java)

```
@Override  
public View onCreateView(LayoutInflater inflater, ViewGroup parent,  
    Bundle savedInstanceState) {  
    View v = inflater.inflate(R.layout.fragment_crime, parent, false);  
    ...  
    mPhotoButton = (ImageButton)v.findViewById(R.id.crime_imageButton);  
    mPhotoButton.setOnClickListener(new View.OnClickListener() {  
        ...  
    });  
    // If camera is not available, disable camera functionality  
    PackageManager pm = getActivity().getPackageManager();  
    boolean hasACamera =  
        pm.hasSystemFeature(PackageManager.FEATURE_CAMERA) ||  
        pm.hasSystemFeature(PackageManager.FEATURE_CAMERA_FRONT) ||  
        (Build.VERSION.SDK_INT >= Build.VERSION_CODES.GINGERBREAD &&  
        Camera.getNumberOfCameras() > 0);  
    if (!hasACamera) {  
        mPhotoButton.setEnabled(false);  
    }  
    ...  
}
```

Use `hasSystemFeature(String)`, passing constants for the device camera features you are interested in. If either of these checks returns true, you should have a camera available.

However, some emulators will return true even if there is no camera available.

Gingerbread introduces the `getNumberOfCameras()` API call. If this is greater than 0, you are guaranteed to have a camera available. If the device does not have a camera, you set the enabled property of the **ImageButton** to false.

Hiding the Status Bar and Action Bar 1/2

Users will spend only a brief time in **CrimeCameraActivity** and will be focused on the viewfinder. So, hiding the status and action bar is useful.

Interestingly, you cannot hide the status bar and action bar from **CrimeCameraFragment**; you must do this in **CrimeCameraActivity**.

The calls

to `requestWindowFeature(...)` and `addFlags(...)` must be made before the activity's view is created in `Activity.setContentView(...)`, which, in **CrimeCameraActivity**, is called in the superclass's implementation of `onCreate(Bundle)`.

A fragment cannot be added before its hosting activity's view is created. Thus, you have to call these methods from within your Activity.

Configuring activity (**CrimeCameraActivity.java**)

```
public class CrimeCameraActivity extends  
SingleFragmentActivity {  
  
    @Override  
  
    public void onCreate(Bundle savedInstanceState) {  
        // Hide the window title.  
        requestWindowFeature(Window.FEATURE_NO_TITLE);  
        // Hide the status bar and other OS-level chrome  
  
        getWindow().addFlags(WindowManager.LayoutParams.FLAG  
        _FULLSCREEN);  
        super.onCreate(savedInstanceState);  
    }  
  
    @Override  
    protected Fragment createFragment() {  
        return new CrimeCameraFragment();  
    }  
}
```

Hiding the Status Bar and Action Bar 2/2

CrimeCameraActivity without status bar or action bar



Android UI

Camer II: Take Picture and Handle Images

Javed Hasan
BJIT Limited

Lesson Content

CriminalIntent New Feature and Design Aspects

- Capture Image from Camera's Preview
- Display Image in CrimeFragment's View
- Option to Show the Larger Image in a DialogFragment

Taking a Picture from CrimeCameraActivity

- Update UI with a Progress Bar
- Implement Camera Callbacks
- Setting the Picture Size

Passing Data back to CrimeFragment

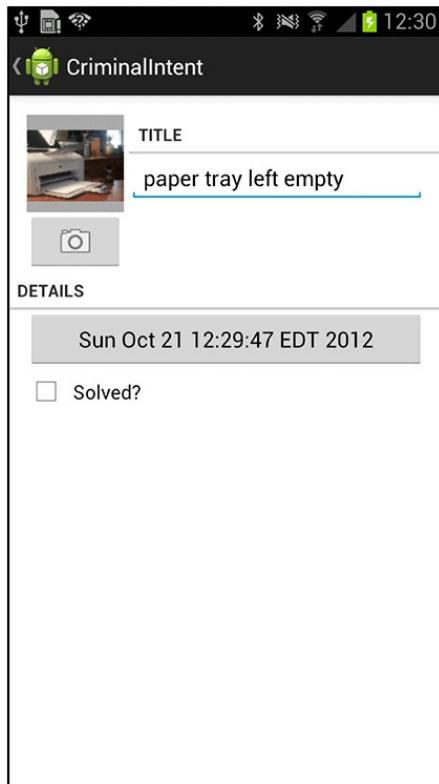
- Setting Result in CrimeCameraFragment
- Retrieving Result in CrimeFragment

Show Image in View

- Update Model Layer with Photo
- Show Photo in CrimeFragment's View and Larger Photo in a DialogFragment

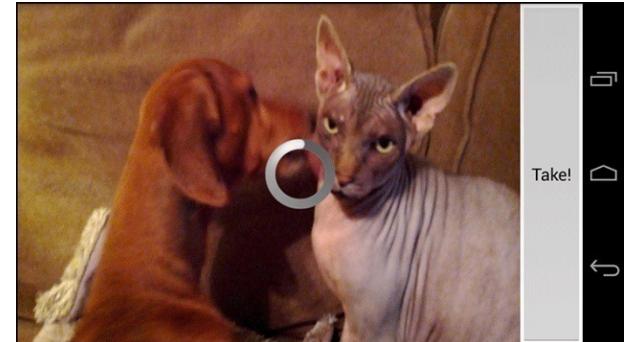
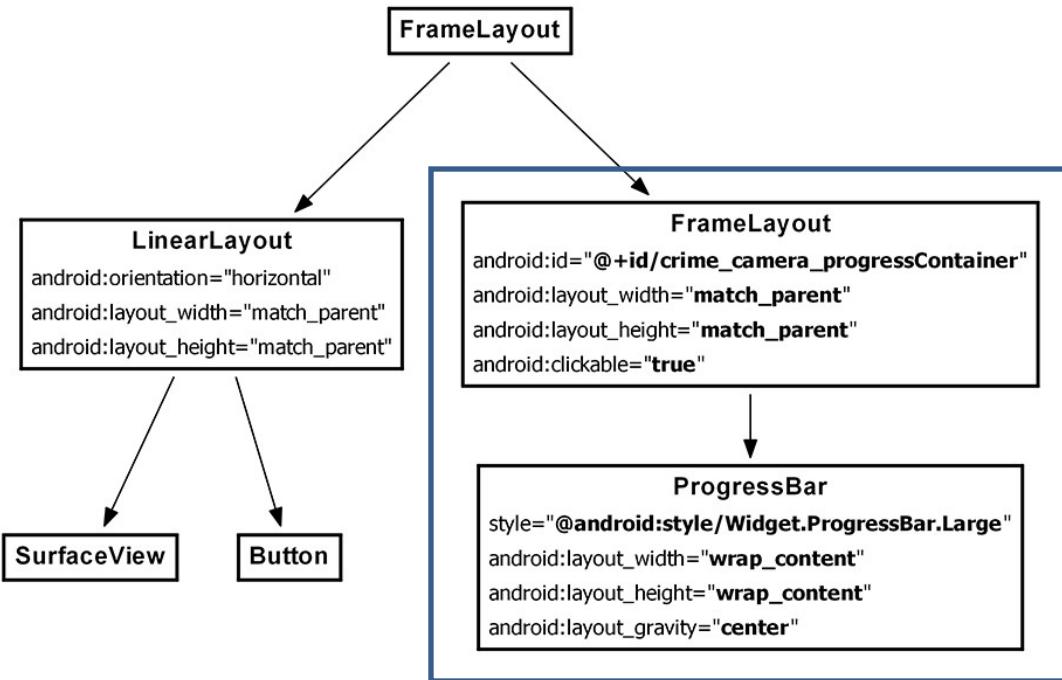
New Feature Overview

You will capture an image from the camera's preview and save it as a JPEG on the file system. Then you will associate that JPEG with a **Crime** and display it in **CrimeFragment's** view. You will also offer the user the option to view a larger version of the image in a **DialogFragment**.



Take Picture: Update UI with Progress Bar

Adding FrameLayout and ProgressBar widgets (fragment_crime_camera.xml)



Defining the **FrameLayout**'s height and width as `match_parent` and by setting `android:clickable="true"`, you ensure that the **FrameLayout** will intercept (and do nothing with) any touch events. This prevents the user from interacting with anything in the **LinearLayout** and, in particular, from pressing the Take! button again.

Take Picture: Wiring up FrameLayout

Add a member variable for the FrameLayout, get a reference to it, and mark it as invisible.

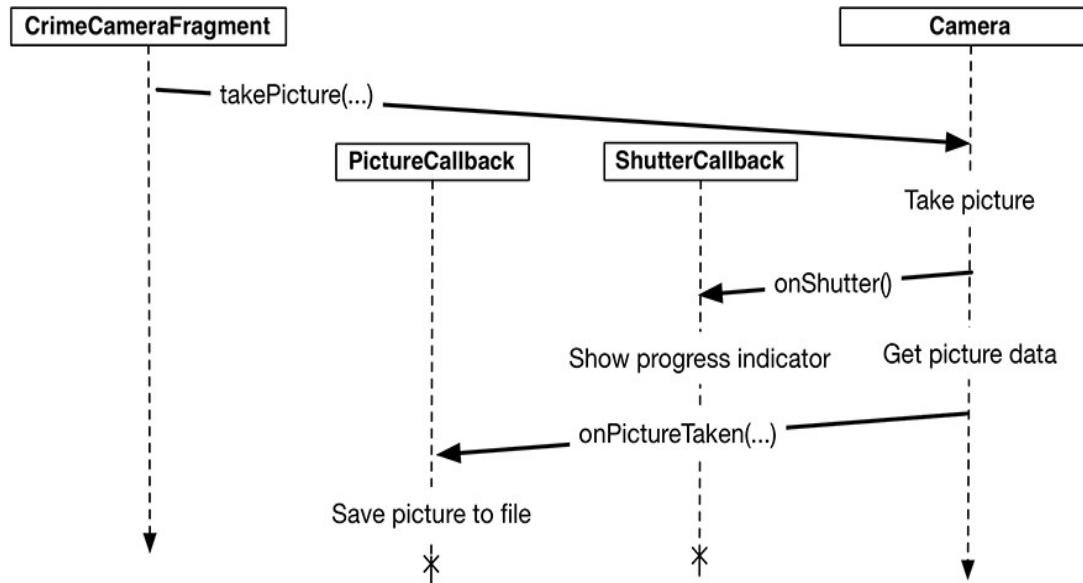
`@Override`

```
public View onCreateView(LayoutInflater inflater, ViewGroup parent, Bundle savedInstanceState) {  
    View v = inflater.inflate(R.layout.fragment_crime_camera, parent, false);  
    mProgressContainer = v.findViewById(R.id.crime_camera_progressContainer);  
    mProgressContainer.setVisibility(View.INVISIBLE);  
    ...  
    return v;  
}  
...  
}
```

Take Picture: Implement Camera Callbacks 1/3

To take a picture, you call the following aptly-named Camera method:

```
public final void takePicture(Camera.ShutterCallback shutter, Camera.PictureCallback raw, Camera.PictureCallback jpeg)
```



The `ShutterCallback` occurs when the camera captures the picture but before the image data is processed and available. The first `PictureCallback` occurs when the raw image data is available. This callback is typically used when processing the raw image before saving it. The second `PictureCallback` occurs when a JPEG version of the image is available.

Take Picture: Implement Camera Callbacks 2/3

In **CrimeCameraFragment.java**, add a `Camera.ShutterCallback` implementation that makes `mProgressContainer` visible and a `Camera.PictureCallback` implementation that handles naming and saving the JPEG file.

(**CrimeCameraFragment.java**)

```
...
private View mProgressContainer;
private Camera.ShutterCallback mShutterCallback = new
    Camera.ShutterCallback() {
    public void onShutter() {
        // Display the progress indicator
        mProgressContainer.setVisibility(View.VISIBLE);
    }
};

private Camera.PictureCallback mJpegCallback = new
    Camera.PictureCallback() {
    public void onPictureTaken(byte[] data, Camera camera) {
        // Create a filename
        String filename = UUID.randomUUID().toString() + ".jpg";
        // Save the jpeg data to disk
        FileOutputStream os = null;
        boolean success = true;
        try {
            os = getActivity().openFileOutput(filename, Context.MODE_PRIVATE);
            os.write(data);
        } catch (Exception e) {
            Log.e(TAG, "Error writing to file " + filename, e);
            success = false;
        } finally {
            try {
                if (os != null) os.close();
            } catch (Exception e) {
                Log.e(TAG, "Error closing file " + filename, e);
                success = false;
            }
        }
        if (success) {
            Log.i(TAG, "JPEG saved at " + filename);
        }
        getActivity().finish();
    };
}
...
}
```

Take Picture: Implement Camera Callbacks 3/3

Implementing `takePicture(...)` on button click (`CrimeCameraFragment.java`)

```
@Override  
@SuppressLint("NewApi")  
public View onCreateView(LayoutInflater inflater, ViewGroup parent,  
    Bundle savedInstanceState) {  
  
    ...  
  
    takePictureButton.setOnClickListener(new View.OnClickListener() {  
        public void onClick(View v) {  
            getActivity().finish();  
            if (mCamera != null) {  
                mCamera.takePicture(mShutterCallback, null, mJpegCallback);  
            }  
        }  
    });  
  
    ...  
    return v;  
}
```

Take Picture: Setting the Picture Size

Calling `getBestSupportedSize(...)` to set picture size
(CrimeCameraFragment.java)

(CrimeCameraFragment.java)

```
...
public void surfaceChanged(SurfaceHolder holder, int format, int w,
int h) {
    if (mCamera == null) return;
    // The surface has changed size; update the camera preview size
    Camera.Parameters parameters = mCamera.getParameters();
    Size s =
    getBestSupportedSize(parameters.getSupportedPreviewSizes(), w, h);
    parameters.setPreviewSize(s.width, s.height);
    s = getBestSupportedSize(parameters.getSupportedPictureSizes(),
w, h);
    parameters.setPictureSize(s.width, s.height);
    mCamera.setParameters(parameters);
...
}
});
```

The camera needs to know what size picture to create.

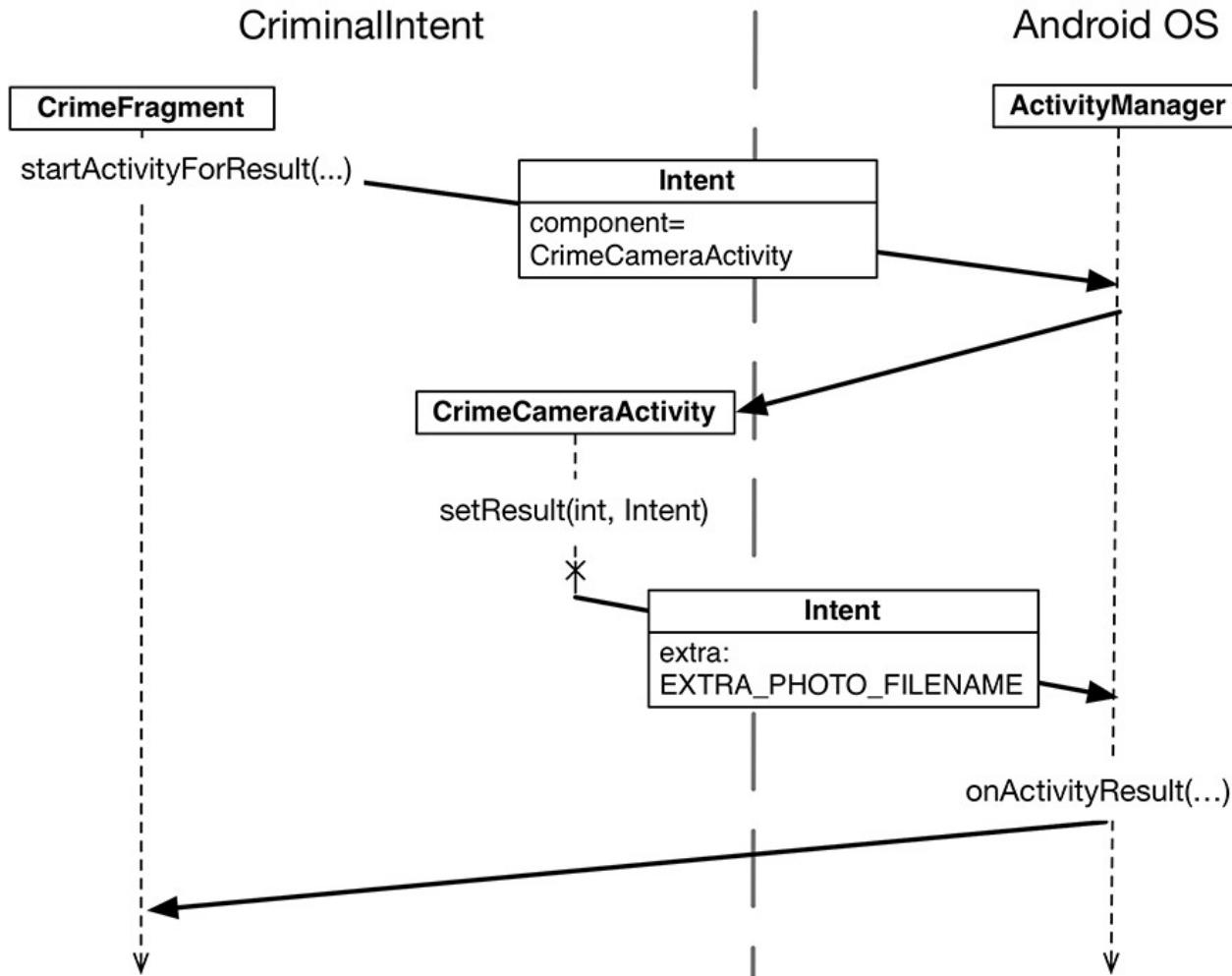
Setting the picture size works the same as setting the preview size. You can get a list of acceptable picture sizes by calling the following

`Camera.Parameters` method:

```
public List<Camera.Size> getSupportedPictureSizes()
```

Run `CriminalIntent` and press the `Take!` button. In `LogCat`, create a filter with a **CrimeCameraFragment tag** to see where your picture landed.

Passing Data Back to CrimeFragment



Starting CrimeCameraActivity for a Result

(CrimeFragment.java)

```
public class CrimeFragment extends Fragment {  
    ...  
  
    private static final int REQUEST_DATE = 0;  
    private static final int REQUEST_PHOTO = 1;  
  
    ...  
  
    @Override  
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,  
        Bundle savedInstanceState) {  
        ...  
  
        mPhotoButton.setOnClickListener(new View.OnClickListener() {  
            public void onClick(View v) {  
                // Launch the camera activity  
                Intent i = new Intent(getActivity(), CrimeCameraActivity.class);  
                startActivity(i);  
                startActivityForResult(i, REQUEST_PHOTO);  
            }  
        });  
        ...  
    }  
}
```

Setting a Result in CrimeCameraFragment

Adding an extra for photo filename
(CrimeCameraFragment.java)

```
public class CrimeCameraFragment extends Fragment {  
    private static final String TAG = "CrimeCameraFragment";  
    public static final String EXTRA_PHOTO_FILENAME =  
        "com.bignerdranch.android.criminalintent.photo_filename";  
    ...  
    private Camera.PictureCallback mJpegCallback = new  
    Camera.PictureCallback() {  
        public void onPictureTaken(byte[] data, Camera camera) {  
            ...  
            try {  
                ...  
            } catch (Exception e) {  
                ...  
            } finally {  
                ...  
            }  
        }  
    };  
}
```

```
Log.i(TAG, "JPEG saved at " + filename);  
// Set the photo filename on the result intent  
if (success) {  
    Intent i = new Intent();  
    i.putExtra(EXTRA_PHOTO_FILENAME, filename);  
    getActivity().setResult(Activity.RESULT_OK, i);  
} else {  
    getActivity().setResult(Activity.RESULT_CANCELED);  
}  
getActivity().finish();  
}  
};  
...  
}
```

Retrieving Filename in CrimeFragment

(CrimeFragment.java)

```
public class CrimeFragment extends Fragment {  
    private static final String TAG = "CrimeFragment"  
    public static final String EXTRA_CRIME_ID =  
        "com.bignerdranch.android.criminalintent.crime_id";  
    ...  
    @Override  
    public void onActivityResult(int requestCode, int resultCode, Intent data) {  
        if (resultCode != Activity.RESULT_OK) return;  
        if (requestCode == REQUEST_DATE) {  
            Date date = (Date)data  
                .getSerializableExtra(DatePickerFragment.EXTRA_DATE);  
            mCrime.setDate(date);  
            updateDate();  
        } else if (requestCode == REQUEST_PHOTO) {  
            // Create a new Photo object and attach it to the crime  
            String filename = data  
                .getStringExtra(CrimeCameraFragment.EXTRA_PHOTO_FILENAME);  
            if (filename != null) {  
                Log.i(TAG, "filename: " + filename);  
            }  
        }  
    }
```

Run **CriminalIntent**.

Take a picture in **CrimeCameraActivity**.

Check LogCat to confirm that
CrimeFragment retrieved a filename.

Show Image in a View

Now that **CrimeFragment** has the filename, there is a lot to be done with it:

- *Step1: Update the Model Layer*

To update the model layer, you will write a **Photo** class that wraps the filename of an image. You will also give **Crime** an `mPhoto` property of type **Photo**. **CrimeFragment** will use the filename to create a **Photo** object and set **Crime's** `mPhoto` property.

- *Step2: Update CrimeFragment's View*

To update **CrimeFragment's** view, you will add an **ImageView** to **CrimeFragment's** layout and display a thumbnail of the **Crime's** photo on this **ImageView**.

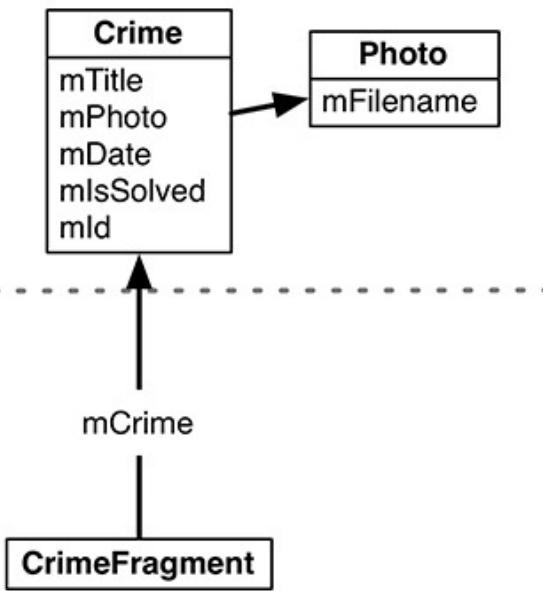
- *Step3: Present a Larger Version of the Image*

To present a larger version of the image, you will create a **DialogFragment** subclass named **ImageFragment** and pass it the path of the photo to display.

Step1: Updating the Model Layer

Model objects and CrimeFragment

Model



Controller

The Photo class (Photo.java)

...

```
public class Photo {  
    private static final String JSON_FILENAME = "filename";  
    private String mFilename;  
    /** Create a Photo representing an existing file on disk */  
    public Photo(String filename) {  
        mFilename = filename;  
    }  
    public Photo(JSONObject json) throws JSONException {  
        mFilename = json.getString(JSON_FILENAME);  
    }  
    public JSONObject toJSON() throws JSONException {  
        JSONObject json = new JSONObject();  
        json.put(JSON_FILENAME, mFilename);  
        return json;  
    }  
    public String getFilename() {  
        return mFilename;  
    }  
}
```

Step1: Giving Crime a Photo Property

A Photo for the Crime (Crime.java)

```
public class Crime {  
    ...  
    private static final String JSON_DATE = "date";  
    private static final String JSON_PHOTO = "photo";  
    ...  
    private Date mDate = new Date();  
    private Photo mPhoto;  
    ...  
    public Crime(JSONObject json) throws JSONException {  
        ...  
        mDate = new Date(json.getLong(JSON_DATE));  
        if (json.has(JSON_PHOTO))  
            mPhoto = new Photo(json.getJSONObject(JSON_PHOTO));  
    }  
    public JSONObject toJSON() throws JSONException {  
        JSONObject json = new JSONObject();  
        ...  
        json.put(JSON_DATE, mDate.getTime());  
        if (mPhoto != null)  
            json.put(JSON_PHOTO, mPhoto.toJSON());  
        return json;  
    }  
}
```

Step1: Setting the Photo Property

Handling a new Photo (CrimeFragment.java)

```
@Override  
public void onActivityResult(int requestCode, int resultCode, Intent data)  
{  
    if (resultCode != Activity.RESULT_OK) return;  
    if (requestCode == REQUEST_DATE) {  
        Date date = (Date)data  
            .getSerializableExtra(DatePickerFragment.EXTRA_DATE);  
        mCrime.setDate(date);  
        updateDate();  
    } else if (requestCode == REQUEST_PHOTO) {  
        // Create a new Photo object and attach it to the crime  
        String filename = data  
            .getStringExtra(CrimeCameraFragment.EXTRA_PHOTO_FILENAME);  
        if (filename != null) {  
            Log.i(TAG, "filename: " + filename);  
            Photo p = new Photo(filename);  
            mCrime.setPhoto(p);  
            Log.i(TAG, "Crime: " + mCrime.getTitle() + " has a photo");  
        }  
    }  
}
```

Run CriminalIntent and take a picture.
Check LogCat to confirm that the Crime
has a photo.

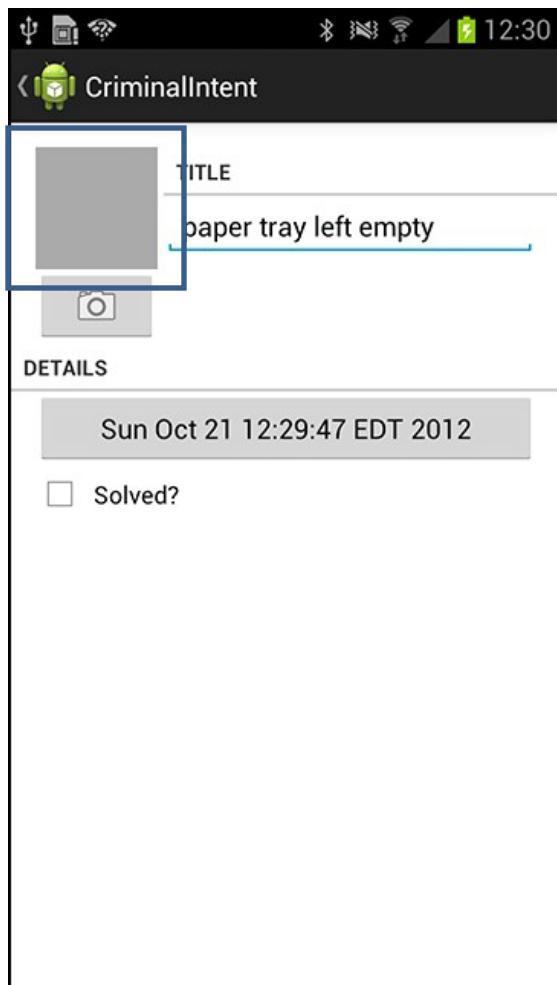
You may be wondering why you created
a **Photo** class instead of simply adding a
filename property to **Crime**.

The latter would work in this situation,
but there are other things you might
need a **Photo** to do, like display a
caption or handle a touch event.

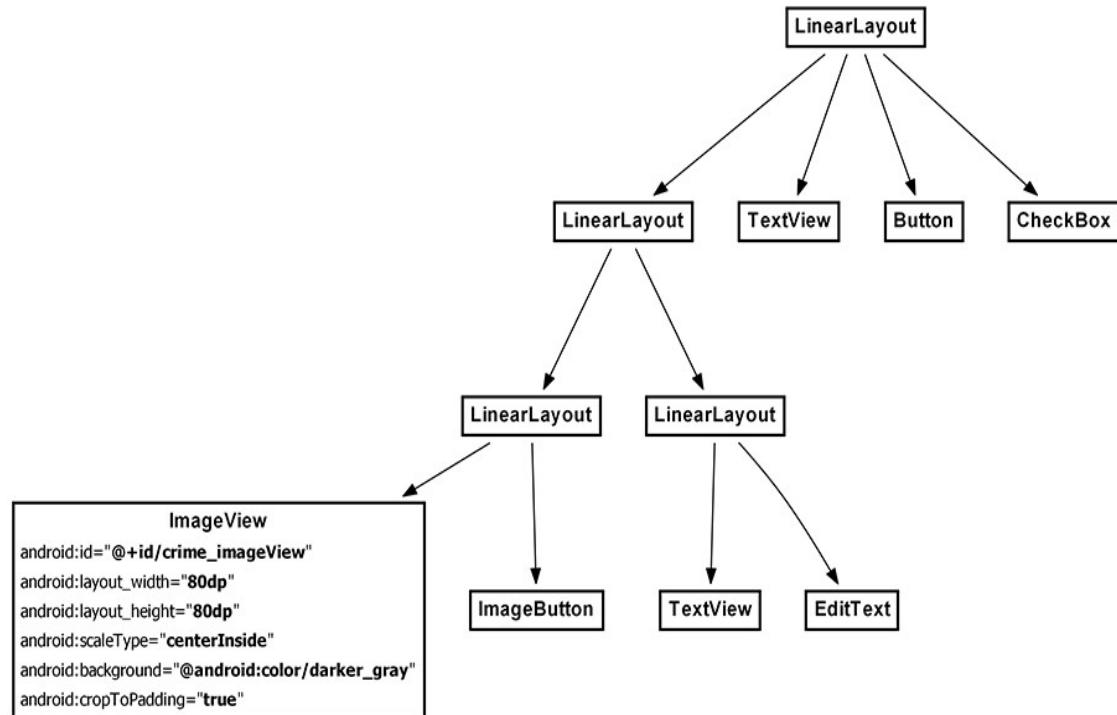
In that case, you would need a separate
class.

Step2: Updating CrimeFragment's View

CrimeFragment with new ImageView



CrimeFragment layout with ImageView
(layout/fragment_crime.xml). Add similar layout for (layout-land/fragment_crime.xml).



Step2: Configuring the Image Button

(CrimeFragment.java)

```
public class CrimeFragment extends Fragment {  
    ...  
    private ImageButton mPhotoButton;  
    private ImageView mPhotoView;  
    ...  
    @Override  
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,  
        Bundle savedInstanceState) {  
        ...  
        mPhotoButton.setOnClickListener(new View.OnClickListener() {  
            public void onClick(View v) {  
                // Launch the camera activity  
                Intent i = new Intent(getActivity(), CrimeCameraActivity.class);  
                startActivityForResult(i, REQUEST_PHOTO);  
            }  
        });  
        mPhotoView = (ImageView)v.findViewById(R.id.crime_imageView);  
        ...  
    }  
}
```

Step2: Adding Scaled Photo to the Image View

Displaying the image on the ImageView requires some image scaling first because the file that came from the camera could be exceptionally large. This may easily blow out your memory budget.

So you need some code to scale the image before loading it and some code to clean up when the image is no longer needed.

In **PictureUtils.java**, add the following method that scales an image to the size of the default display for the device.

Add PictureUtils class (PictureUtils.java)

```
public class PictureUtils {  
    /**  
     * Get a BitmapDrawable from a local file that is scaled down  
     * to fit the current Window size.  
     */  
    @SuppressWarnings("deprecation")  
    public static BitmapDrawable getScaledDrawable(Activity a, String path) {  
        Display display = a.getWindowManager().getDefaultDisplay();  
        float destWidth = display.getWidth();  
        float destHeight = display.getHeight();
```

Step2: Adding Scaled Photo to the Image View

```
// Read in the dimensions of the image on disk  
  
BitmapFactory.Options options = new BitmapFactory.Options();  
options.inJustDecodeBounds = true;  
BitmapFactory.decodeFile(path, options);  
float srcWidth = options.outWidth;  
float srcHeight = options.outHeight;  
  
int inSampleSize = 1;  
if (srcHeight > destHeight || srcWidth > destWidth) {  
    if (srcWidth > srcHeight) {  
        inSampleSize = Math.round(srcHeight / destHeight);  
    } else {  
        inSampleSize = Math.round(srcWidth / destWidth);  
    }  
}  
options = new BitmapFactory.Options();  
options.inSampleSize = inSampleSize;  
  
Bitmap bitmap = BitmapFactory.decodeFile(path, options);  
return new BitmapDrawable(a.getResources(), bitmap);  
}
```

in CrimeFragment, add a private method that sets a scaled image on the ImageView.

Adding **showPhoto()** (CrimeFragment.java)

```
@Override  
public View onCreateView(LayoutInflater inflater, ViewGroup parent,  
    Bundle savedInstanceState) {  
    ...  
}  
private void showPhoto() {  
    // (Re)set the image button's image based on our photo  
    Photo p = mCrime.getPhoto();  
    BitmapDrawable b = null;  
    if (p != null) {  
        String path = getActivity()  
            .getFileStreamPath(p.getFilename()).getAbsolutePath();  
        b = PictureUtils.getScaledDrawable(getActivity(), path);  
    }  
    mPhotoView.setImageDrawable(b);  
}
```

Step2: Loading the Image

In **CrimeFragment.java**, add an implementation of `onStart()` that calls `showPhoto()` to have the photo ready as soon as **CrimeFragment's** view becomes visible to the user.

In `CrimeFragment.onActivityResult(...)`, call `showPhoto()` to ensure that the image will be visible when the user returns from **CrimeCameraActivity**.

```
@Override  
public void onStart() {  
    super.onStart();  
    showPhoto();  
}
```

```
@Override  
public void onActivityResult(int requestCode, int resultCode, Intent data) {  
    if (resultCode != Activity.RESULT_OK) return;  
  
    if (requestCode == REQUEST_PHOTO) {  
        // Create a new Photo object and attach it to the crime  
        String filename = data  
.getStringExtra(CrimeCameraFragment.EXTRA_PHOTO_FILENAME);  
        if (filename != null) {  
            Photo p = new Photo(filename);  
            mCrime.setPhoto(p);  
            showPhoto();  
            Log.i(TAG, "Crime: " + mCrime.getTitle() + " has a photo");  
        }  
    }  
}
```

Step2: Unloading the Image

Cleanup work (PictureUtils.java)

```
public class PictureUtils {  
    ...  
    @SuppressLint("Deprecation")  
    public static BitmapDrawable  
    getScaledDrawable(Activity a, String path) {  
        ...  
    }  
    public static void cleanImageView(ImageView  
    imageView) {  
        if (!(imageView.getDrawable() instanceof  
        BitmapDrawable))  
            return;  
        // Clean up the view's image for the sake of  
        // memory  
        BitmapDrawable b =  
        (BitmapDrawable)imageView.getDrawable();  
        b.getBitmap().recycle();  
        imageView.setImageDrawable(null);  
    }  
}
```

Bitmap.recycle() frees the native storage for your bitmap. This is most of the meat of your bitmap object.

If you do not free this memory explicitly by calling recycle(), then the memory will still be cleaned up. However, it will be cleaned up at some future point in a finalizer, not when the bitmap itself is garbage-collected.

This means that there is a chance that you will run out of memory before the finalizer is called.

Unloading the image (CrimeFragment.java)

```
@Override  
public void onStop() {  
    super.onStop();  
    PictureUtils.cleanImageView(mPhotoView);  
}
```

Loading images in onStart() and unloading them in onStop() is a good practice. These methods mark the points where your activity can be seen by the user.

Step3: Showing Larger Image in a Dialog Fragment



Create ImageFragment (ImageFragment.java)

```
public class ImageFragment extends DialogFragment {  
    public static final String EXTRA_IMAGE_PATH =  
        "com.bignerdranch.android.criminalintent.image_path";  
    public static ImageFragment newInstance(String imagePath) {  
        Bundle args = new Bundle();  
        args.putSerializable(EXTRA_IMAGE_PATH, imagePath);  
        ImageFragment fragment = new ImageFragment();  
        fragment.setArguments(args);  
        fragment.setStyle(DialogFragment.STYLE_NO_TITLE, 0);  
        return fragment;  
    }
```

You set the fragment's style to `DialogFragment.STYLE_NO_TITLE` to achieve a minimalist look.

Step3: Showing Larger Image in a Dialog Fragment

Create ImageFragment (ImageFragment.java)

```
public class ImageFragment extends DialogFragment {  
    public static final String EXTRA_IMAGE_PATH =  
        "com.bignerdranch.android.criminalintent.image_path";  
    public static ImageFragment newInstance(String imagePath) {  
        ...  
    }  
    private ImageView mImageView;  
    @Override  
    public View onCreateView(LayoutInflater inflater,  
        ViewGroup parent, Bundle savedInstanceState) {  
        mImageView = new ImageView(getActivity());  
        String path =  
            (String) getArguments().getSerializable(EXTRA_IMAGE_PATH);  
        BitmapDrawable image =  
            PictureUtils.getScaledDrawable(getActivity(), path);  
        mImageView.setImageDrawable(image);  
        return mImageView;  
    }  
}
```

```
@Override  
public void onDestroyView() {  
    super.onDestroyView();  
    PictureUtils.cleanImageView(mImageView);  
}  
}}
```

In **ImageFragment.java**,
override `onCreateView(...)` to create
an **ImageView** from scratch and retrieve
the path from its arguments.

Then get a scaled version of the image and
set it on the **ImageView**.

Override `onDestroyView()` to free up memory once
the image is no longer needed.

Step3: Showing Larger Image in a Dialog Fragment

Show ImageFragment (CrimeFragment.java)

```
public class CrimeFragment extends Fragment {  
    ...  
    private static final String DIALOG_IMAGE = "image";  
    ...  
    @Override  
    @TargetApi(11)  
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,  
        Bundle savedInstanceState) {  
        ...  
        mPhotoView = (ImageView)v.findViewById(R.id.crime_imageView);  
        mPhotoView.setOnClickListener(new View.OnClickListener() {  
            public void onClick(View v) {  
                Photo p = mCrime.getPhoto();  
                if (p == null)  
                    return;  
                FragmentManager fm = getActivity().getSupportFragmentManager();
```

```
                String path = getActivity()  
                    .getFileStreamPath(p.getFilename()).getAbsolutePath();  
                ImageFragment.newInstance(path)  
                    .show(fm, DIALOG_IMAGE);  
            }  
        });  
        ...  
    }
```

In CrimeFragment.java, add a listener to mPhotoView. Within its implementation, create an instance of ImageFragment and add it to CrimePagerActivity's FragmentManager by calling show(...) on the ImageFragment.

Run **CriminalIntent**. Take a picture and confirm that you can view the crime scene photo in all its gory.

Challenge: Crime Image Orientation

Sometimes, the user will choose to take a picture in portrait orientation. Search the API documentation for a way to detect the orientation. Stash the correct orientation in your **Photo** and use it to rotate the picture appropriately in **CrimeFragment** and **ImageFragment**.

Challenge: Deleting Photos

Right now, you can replace a **Crime's** photo, but the old file will continue to take up space on disk. Add code to **CrimeFragment's** **onActivityResult(int, int, Intent)** method to check the crime for an existing photo and delete that file from disk.

For another challenge, give users the power to delete a photo without having to replace it. Implement a context menu and/or a contextual action mode in **CrimeFragment** that is triggered by long-pressing the thumbnail image. Offer a Delete Photo menu item that deletes the photo from the disk, the model, and the **ImageView**.

Android UI Implicit Intents

Javed Hasan
BJIT Limited

Lesson Content

CriminalIntent New Feature

- Opening a Contacts App from CriminalIntent
- Opening a Text Sending App from CriminalIntent

Update View Layer

- Change CrimeFragment Layout
 - Add Suspect Button
 - Add Send Crime Report Button
 - Create a Crime Report Using a Set of Format Resource Strings

Adding a Suspect to the Model Layer

- Save and Load Data Using JSON

Using Implicit Intents

- Sending a Crime Report
- Choosing a Suspect from Contact List

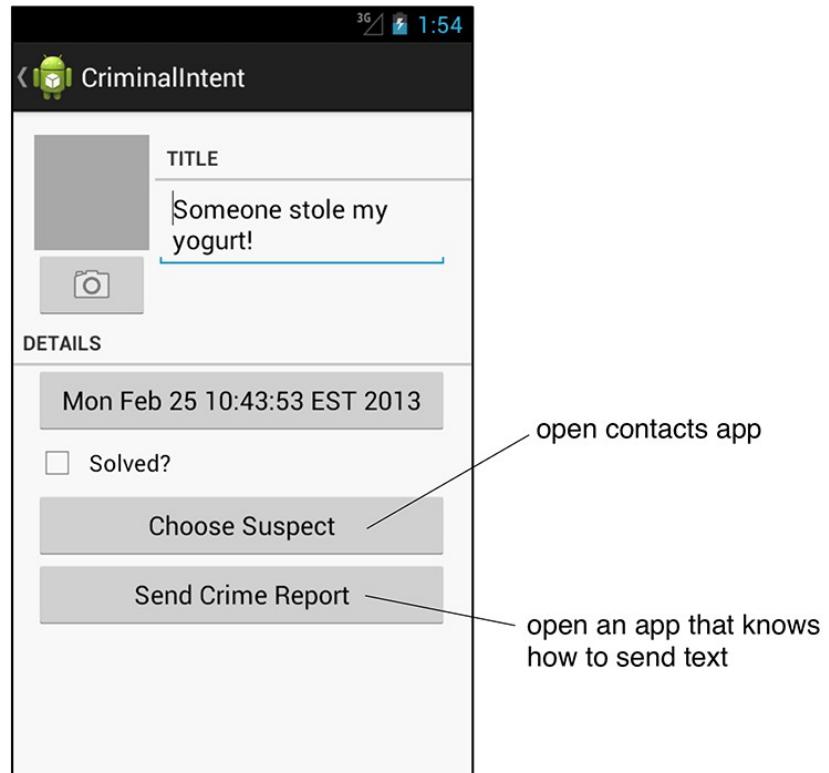
New Feature Overview

You can start an activity in another application on the device using an *implicit intent*.

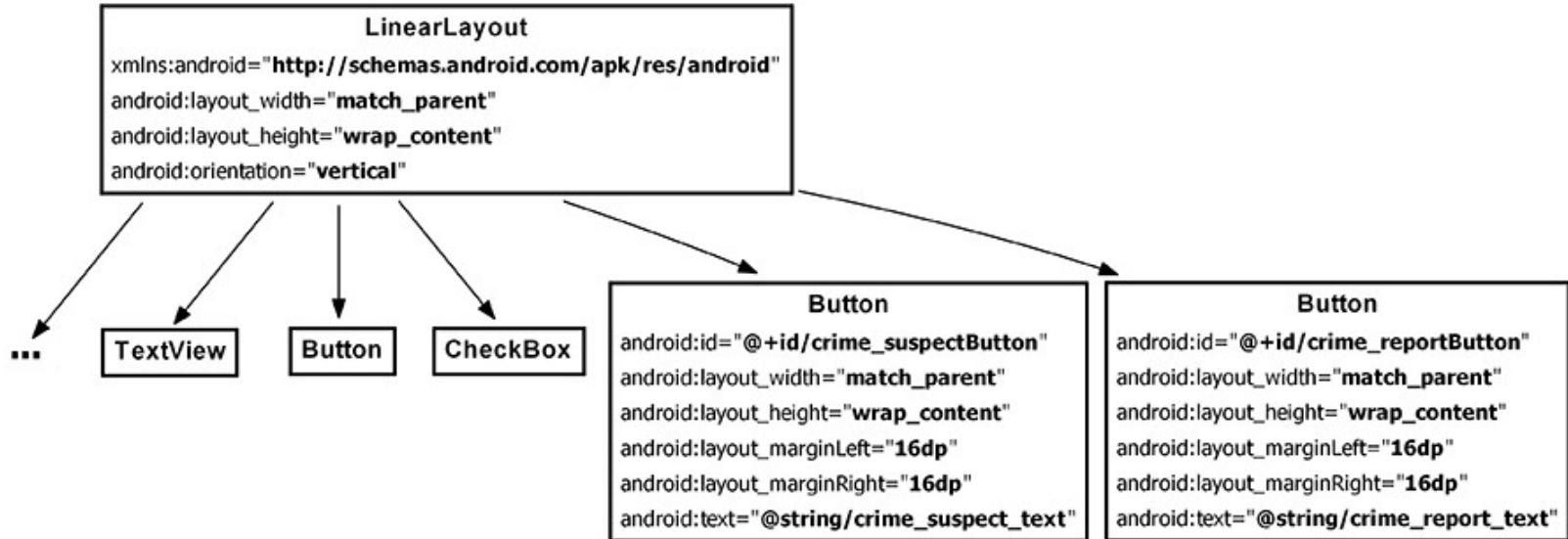
In CriminalIntent, you will use implicit intents to enable picking a suspect for a **Crime** from the user's list of contacts and sending a text-based report of a crime.

The user will choose a suspect from whatever contacts app is installed on the device and will be offered a choice of apps to send the crime report.

Using implicit intents to harness other applications is far easier than writing your own implementations for common tasks. Users also appreciate being able to use apps they already know and like in conjunction with your app.



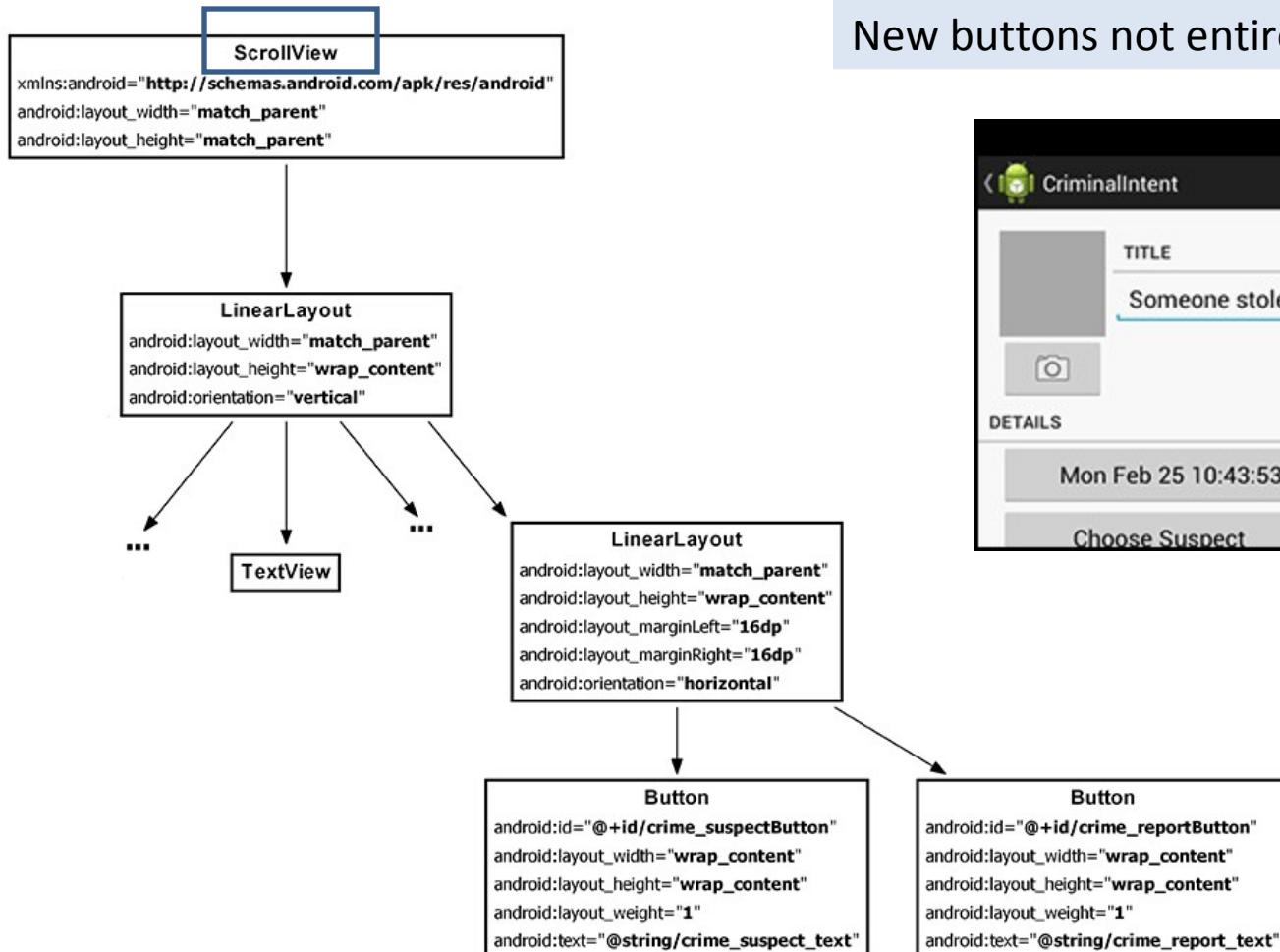
Add Buttons to CrimeFragment Layout



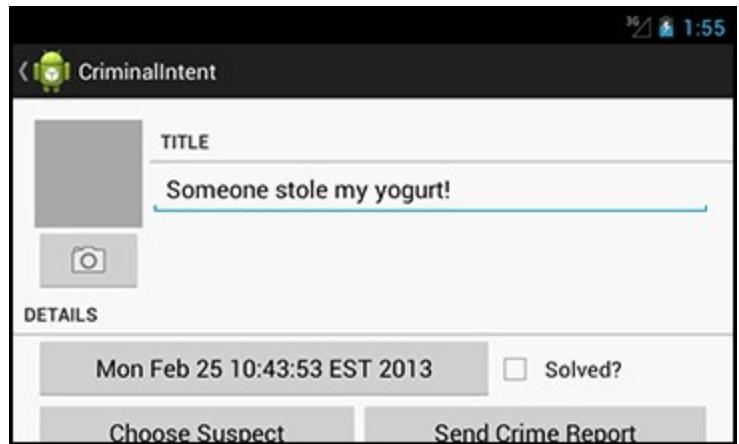
Adding button strings (strings.xml)

```
<string name="take">Take!</string>
<string name="crime_suspect_text">Choose Suspect</string>
<string name="crime_report_text">Send Crime Report</string>
</resources>
```

Add Buttons to Landscape Layout



New buttons not entirely visible in landscape



Add a Suspect to the Model Layer

Adding suspect field (Crime.java)

```
public class Crime {  
    ...  
  
    private static final String JSON_PHOTO = "photo";  
  
    private static final String JSON_SUSPECT = "suspect";  
  
    ...  
  
    private Photo mPhoto;  
  
    private String mSuspect;  
  
    public Crime(JSONObject json) throws JSONException {  
        mId = UUID.fromString(json.getString(JSON_ID));  
        ...  
        if (json.has(JSON_PHOTO))  
            mPhoto = new  
Photo(json.getJSONObject(JSON_PHOTO));  
        if (json.has(JSON_SUSPECT))  
            mSuspect = json.getString(JSON_SUSPECT);  
    }  
}
```

```
    public JSONObject toJSON() throws JSONException {  
        JSONObject json = new JSONObject();  
        ...  
        if (mPhoto != null)  
            json.put(JSON_PHOTO, mPhoto.toJSON());  
        if (mSuspect != null)  
            json.put(JSON_SUSPECT, mSuspect);  
        return json;  
    }  
  
    public void setPhoto(Photo p) {  
        mPhoto = p;  
    }  
    public String getSuspect() {  
        return mSuspect;  
    }  
    public void setSuspect(String suspect) {  
        mSuspect = suspect;  
    }  
}
```

Using a Format String 1/2

Add string resources (strings.xml)

```
<string name="crime_suspect_text">Choose Suspect</string>
<string name="crime_report_text">Send Crime Report</string>
<string name="crime_report">%1$s!
    The crime was discovered on %2$s. %3$s, and %4$s
</string>
<string name="crime_report_solved">The case is
solved</string>
<string name="crime_report_unsolved">The case is not
solved</string>
<string name="crime_report_no_suspect">There is no
suspect.</string>
<string name="crime_report_suspect">The suspect is
% s.</string>
<string name="crime_report_subject">CriminalIntent Crime
Report</string>
<string name="send_report">Send crime report via</string>
</resources>
```

We need to create a template crime report that can be configured with the specific crime's details.

Because you will not know a crime's details until runtime, you must use a format string with placeholders that can be replaced at runtime.

The %1\$s, %2\$s, etc. are placeholders that expect string arguments.

In code, you will call `getString(...)` and pass in the format string and four other strings in the order in which they should replace the placeholders.

In `CrimeFragment.java`, add a method that creates four strings and then pieces them together and returns a complete report.

Using a Format String 2/2

Add `getCrimeReport()` method (`CrimeFragment.java`)

```
private String getCrimeReport() {  
    String solvedString = null;  
    if (mCrime.isSolved()) {  
        solvedString = getString(R.string.crime_report_solved);  
    } else {  
        solvedString = getString(R.string.crime_report_unsolved);  
    }  
    String dateFormat = "EEE, MMM dd";  
    String dateString = DateFormat.format(dateFormat, mCrime.getDate()).toString();  
    String suspect = mCrime.getSuspect();  
    if (suspect == null) {  
        suspect = getString(R.string.crime_report_no_suspect);  
    } else {  
        suspect = getString(R.string.crime_report_suspect, suspect);  
    }  
    String report = getString(R.string.crime_report,  
        mCrime.getTitle(), dateString, solvedString, suspect);  
    return report;  
}
```

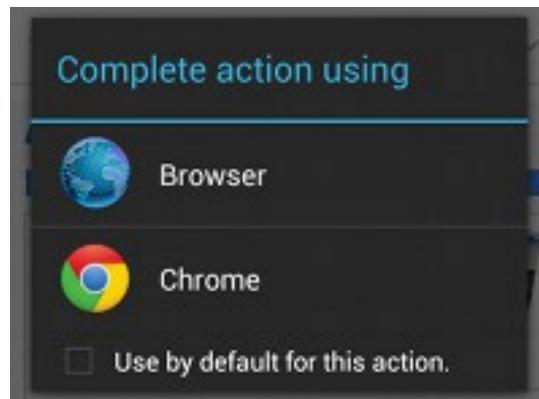
Implicit Intent Concepts 1/3

An Intent is an object that describes to the OS something that you want it to do.

With the *explicit intents*, you explicitly name the activity that you want the OS to start.

```
Intent i = new Intent(getActivity(), CrimeCameraActivity.class);  
startActivity(i);
```

With an *implicit intent*, you describe to the OS the job that you want done. The OS then starts the activity that has advertised itself as capable of doing that job. If the OS finds more than one capable activity, then the user is offered a choice.



Implicit Intent Concepts 2/3

Parts of an Implicit Intent

The *action* that you are trying to perform

These are typically constants from the Intent class. If you want to view a URL, you can use `Intent.ACTION_VIEW` for your action. To send something, you use `Intent.ACTION_SEND`.

The location of any *data*

This can be something outside the device, like the URL of a web page, but it can also be a URI to a file or a content URI pointing to a record in a **ContentProvider**.

The *type* of data that the action is for

This is a MIME type, like `text/html` or `audio/mpeg3`. If an intent includes a location for data, then the type can usually be inferred from that data.

The optional *categories*

If the action is used to describe what to do, the category usually describes where, when, or how you are trying to use an activity. Android uses the category `android.intent.category.LAUNCHER` to indicate that an activity should be displayed in the top-level app launcher.

Implicit Intent Concepts 3/3

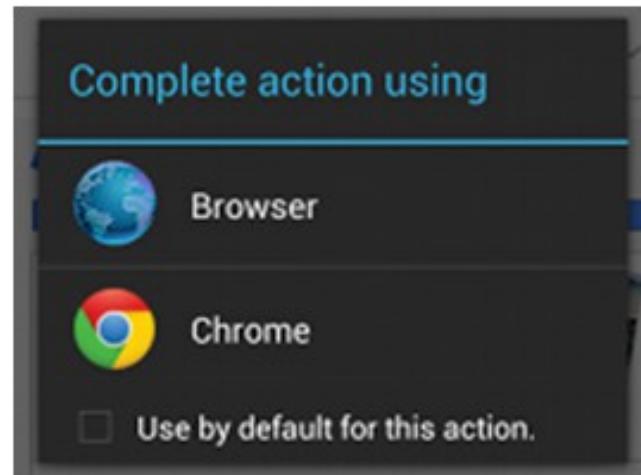
A simple implicit intent for viewing a website would include an action of `Intent.ACTION_VIEW` and a data Uri that is the URL of a website.

Based on this information, the OS will launch the appropriate activity of an appropriate application.

An activity would advertise itself as an appropriate activity for `ACTION_VIEW` via an intent filter in the manifest .

The action element in the intent filter tells the OS that the activity is capable of performing the job, and the `DEFAULT` category tells the OS that it wants to be considered for the job.

```
<activity  
    android:name=".BrowserActivity"  
    android:label="@string/app_name" >  
    <intent-filter>  
        <action android:name="android.intent.action.VIEW" />  
        <category android:name="android.intent.category.DEFAULT" />  
        <data android:scheme="http" android:host="www.bignerdranch.com" />  
    </intent-filter>  
</activity>
```

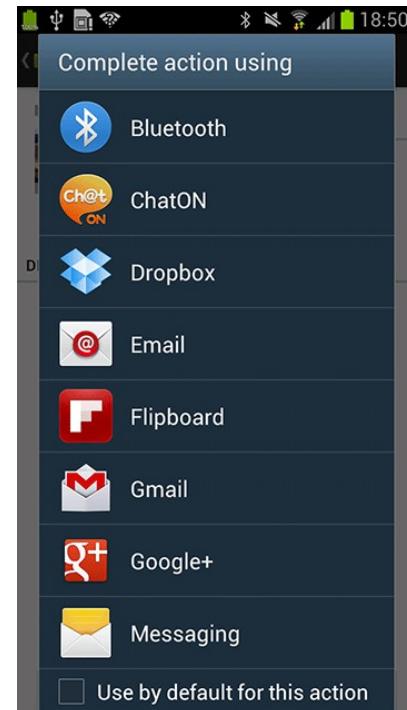


Sending a Crime Report 1/2

Sending a crime report (CrimeFragment.java)

```
...
Button reportButton =
(Button)v.findViewById(R.id.crime_reportButton);
reportButton.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        Intent i = new Intent(Intent.ACTION_SEND);
        i.setType("text/plain");
        i.putExtra(Intent.EXTRA_TEXT, getCrimeReport());
        i.putExtra(Intent.EXTRA_SUBJECT,
                  getString(R.string.crime_report_subject));
        startActivity(i);
    }
});
return v;
}
```

Run **CriminalIntent** and press the **Send Crime Report** button. Because this intent will likely match many activities on the device, you will probably see a list of activities presented in a chooser:



Sending a Crime Report 2/2

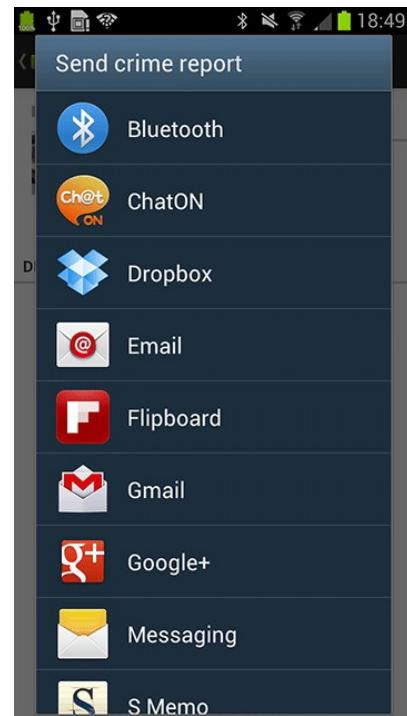
Sometimes you may not see the chooser. Because, either you may already set a default app for the implicit intent or your device has only one activity that can support that intent.

You can create a chooser to be shown every time an implicit intent is used to start an activity.

Use a chooser (CrimeFragment.java)

```
public void onClick(View v) {  
  
    Intent i = new Intent(Intent.ACTION_SEND);  
  
    i.setType("text/plain");  
  
    i.putExtra(Intent.EXTRA_TEXT, getCrimeReport());  
  
    i.putExtra(Intent.EXTRA_SUBJECT,  
        getString(R.string.crime_report_subject));  
  
    i = Intent.createChooser(i, getString(R.string.send_report));  
  
    startActivity(i);  
}
```

Run CriminalIntent and press the Send Crime Report button. If more than one activity that can handle your intent, you will be offered a list to choose from.



Asking Android for a Contact 1/3

Now you are going to create another implicit intent that enables users to choose a suspect from their contacts.

This implicit intent will have an action and a location where the relevant data can be found.

The action will be `Intent.ACTION_PICK`. The data for contacts is at

`ContactsContract.Contacts.CONTENT_URI`.

In short, you are asking Android to help pick an item in the Contacts database.

You expect a result back from the started activity, so you will pass the intent via `startActivityForResult(...)` along with a request code.

Run `CriminalIntent` and press the Choose Suspect button.

(`CrimeFragment.java`)

```
...
private static final int REQUEST_PHOTO = 1;
private static final int REQUEST_CONTACT = 2;
...
private ImageButton mPhotoButton;
private Button mSuspectButton;
```

Sending an implicit intent (`CrimeFragment.java`)

```
...
mSuspectButton = (Button)v.findViewById(R.id.crime_suspectButton);
mSuspectButton.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        Intent i = new Intent(Intent.ACTION_PICK,
            ContactsContract.Contacts.CONTENT_URI);
        startActivityForResult(i, REQUEST_CONTACT);
    }
});
if (mCrime.getSuspect() != null) {
    mSuspectButton.setText(mCrime.getSuspect());
}
return v;
}
```

Asking Android for a Contact 2/3

Contacts information is shared by many applications, so Android provides an in-depth API for working with contacts information through a **ContentProvider**.

Instances of this class wrap databases and make it available to other applications. You can access a **ContentProvider** through a **ContentResolver**.

Because you started the activity for a result with `ACTION_PICK`, you will receive an intent via `onActivityResult(...)`.

This intent includes a data URI. This URI is a locator that points at the single contact the user picked.

In `CrimeFragment.java`, add the following code to your `onActivityResult(...)` implementation in **CrimeFragment**:

Asking Android for a Contact 3/3

Pull contact name out (CrimeFragment.java)

```
@Override  
  
public void onActivityResult(int requestCode, int resultCode,  
Intent data) {  
  
    if (resultCode != Activity.RESULT_OK) return;  
  
    if (requestCode == REQUEST_DATE) {  
  
        ...  
  
    } else if (requestCode == REQUEST_PHOTO) {  
  
        ...  
  
    } else if (requestCode == REQUEST_CONTACT) {  
  
        Uri contactUri = data.getData();  
  
        // Specify which fields you want your query to return  
        // values for.  
  
        String[] queryFields = new String[] {  
            ContactsContract.Contacts.DISPLAY_NAME  
        };
```

```
        // Perform your query - the contactUri is like a "where"  
        // clause here  
  
        Cursor c = getActivity().getContentResolver()  
            .query(contactUri, queryFields, null, null, null);  
  
        // Double-check that you actually got results  
  
        if (c.getCount() == 0) {  
            c.close();  
            return;  
        }  
  
        // Pull out the first column of the first row of data -  
        // that is your suspect's name.  
  
        c.moveToFirst();  
  
        String suspect = c.getString(0);  
        mCrime.setSuspect(suspect);  
        mSuspectButton.setText(suspect);  
        c.close();  
    }  
}
```

Contacts Permissions

How are you getting permission to read from the contacts database?

The contacts app is extending its permissions to you. The contacts app has full permissions to the contacts database. When the contacts app returns a data URI in an Intent to the parent activity, it also adds the flag `Intent.FLAG_GRANT_READ_URI_PERMISSION`.

This flag signals to Android that the parent activity in `CriminalIntent` should be allowed to use this data one time. This works well because you do not really need access to the entire contacts database. You only need access to one contact inside that database.

Note: In some devices, it may not be sufficient. So, please add below permission in your `CriminalIntent` manifest file.

```
<uses-permission android:name="android.permission.READ_CONTACTS" />
```

Checking for Responding Activities

If the OS cannot find a matching activity for your implicit intent, then the app will crash. The answer is to check with part of the OS called the **PackageManager** first. The code looks like this:

```
PackageManager pm = getPackageManager();
List<ResolveInfo> activities = pm.queryIntentActivities(yourIntent, 0);
boolean isIntentSafe = activities.size() > 0;
```

You pass your intent into the `PackageManager's queryIntentActivities(...)` method. This method returns a list of objects containing metadata about activities that responded to the passed-in intent. You just need to confirm that there is at least one item in this list – at least one activity on the device that will respond to the intent.

You can run this check in `onCreateView(...)` to disable options that the device will not be able to respond to.

Challenge: Another Implicit Intent

Instead of sending a crime report, an angry user may prefer a phone confrontation with the suspect. Add a new button that calls the named suspect.

You will need the phone number out of the contacts database. Then you can create an implicit intent with a URI of the telephone:

```
Uri number = Uri.parse("tel:5551234");
```

The action can be `Intent.ACTION_DIAL` or `Intent.ACTION_CALL`. `ACTION_CALL` pulls up the phone app and immediately calls the number sent in the intent; `ACTION_DIAL` just dials the number and waits for the user to initiate the call.

We recommend using `ACTION_DIAL`. `ACTION_CALL` may be restricted and will definitely require a permission. Your user may also appreciate the chance to cool down before pressing the Call button.

Android UI

Two-Pane Master-Detail Interfaces

Javed Hasan
BJIT Limited

Lesson Content

CriminalIntent New Feature

- Two-pane Tablet Interface

Adding Layout Flexibility

- Create Layout with Two Fragment Containers for Tablets
- Using Alias Resources

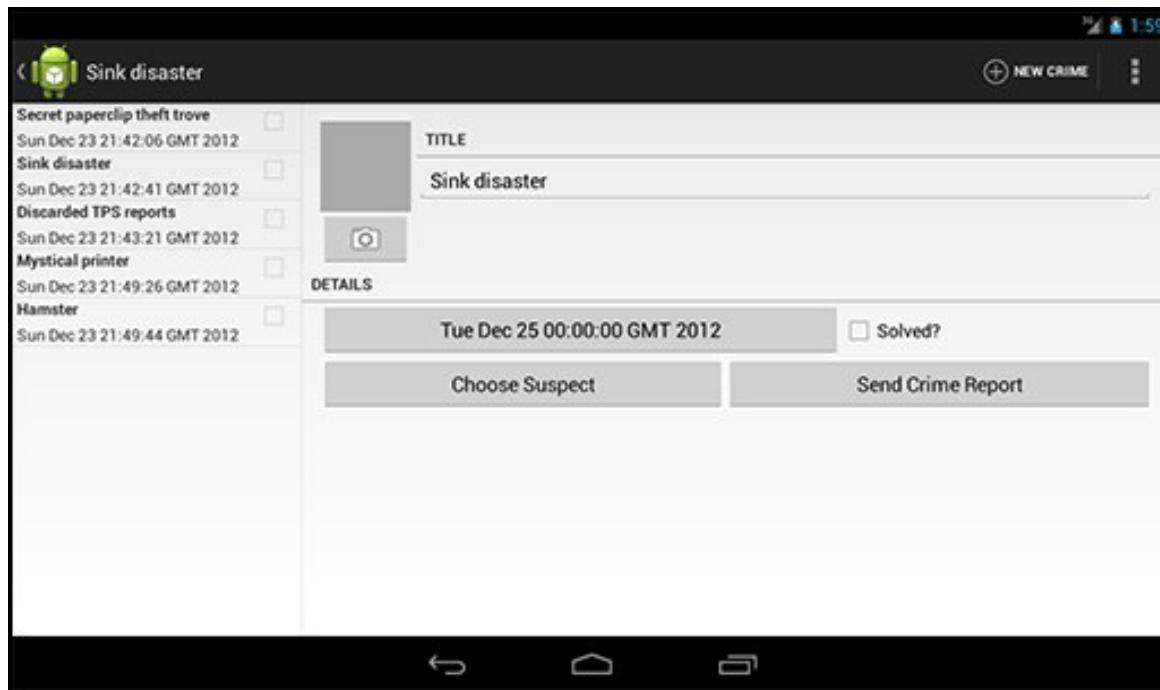
Fragment Callback Interfaces

- Implement Callbacks on Fragment's Activity
- Conditional CrimeFragment Startup based on Phone or Tablet

New Feature Overview

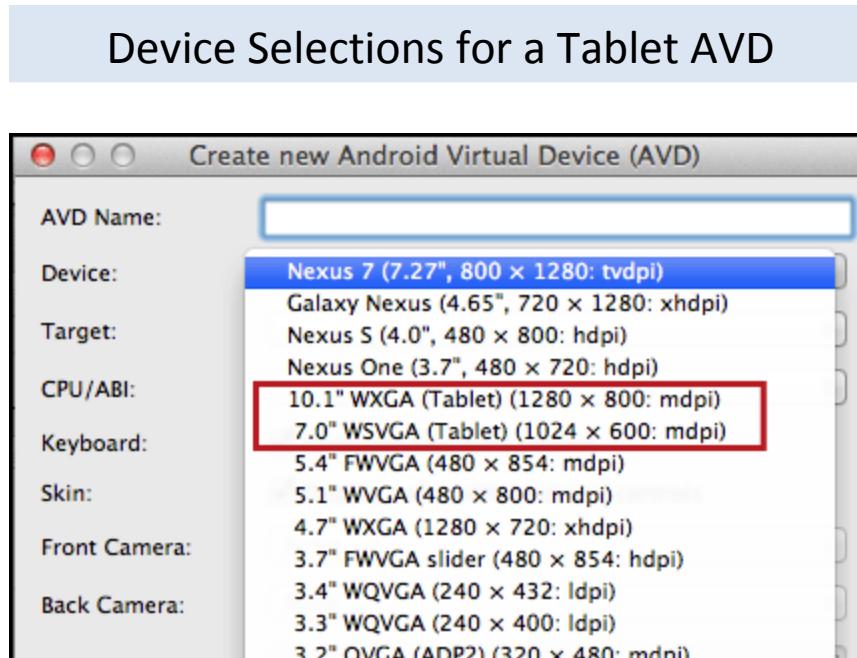
You will create a tablet interface for CriminalIntent that allows users to see and interact with the list of crimes and the detail of an individual crime at the same time.

Master and Detail Sharing the Spotlight

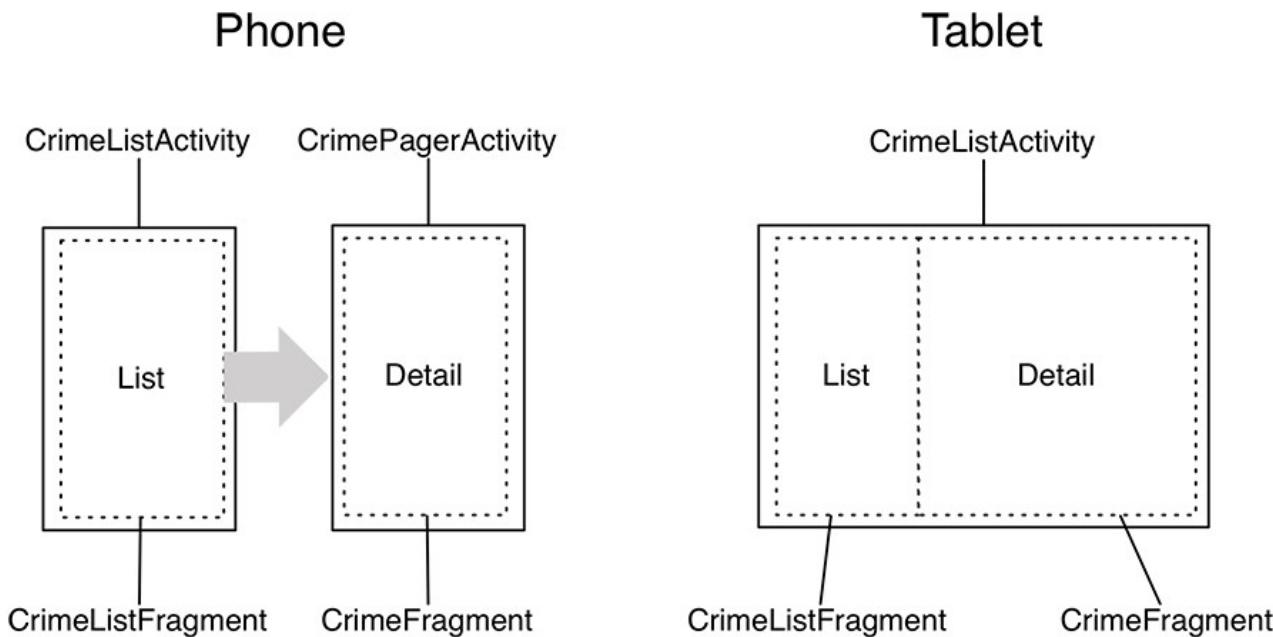


Tablet Device Selection in Emulator

You will need a tablet device or AVD for testing. To create a tablet AVD, select Window → Android Virtual Device Manager. Click New and set this AVD's Device to one of the two selections highlighted below. Then set the AVD's Target to API level 17.



Adding Layout Flexibility



To provide layout flexibility this happen, you are going to:

- Modify **SingleFragmentActivity** so that the layout that gets inflated is not hard-coded
- Create a new layout that consists of two fragment containers
- Modify **CrimeListActivity** so that it will inflate a single-container layout on phones and a two-container layout on tablets

Modifying SingleFragmentActivity

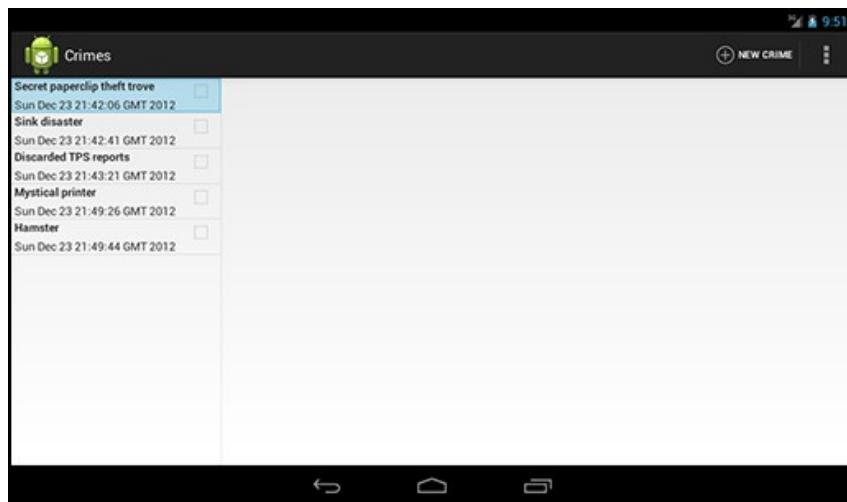
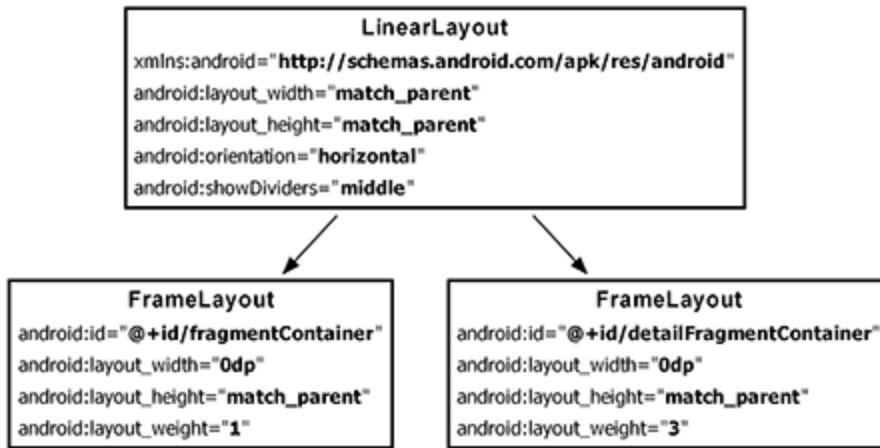
Making SingleFragmentActivity flexible
(SingleFragmentActivity.java)

```
public abstract class SingleFragmentActivity extends FragmentActivity {  
    protected abstract Fragment createFragment();  
  
    protected int getLayoutResId() {  
        return R.layout.activity_fragment;  
    }  
  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_fragment);  
        setContentView(getLayoutResId());  
  
        FragmentManager fm = getSupportFragmentManager();  
  
        Fragment fragment = fm.findFragmentById(R.id.fragmentContainer);  
  
        if (fragment == null) {  
            fragment = createFragment();  
            fm.beginTransaction()  
                .add(R.id.fragmentContainer, fragment)  
                .commit();  
        }  
    }  
}
```

CrimeListActivity is a subclass of **SingleFragmentActivity**. Currently, **SingleFragmentActivity** is set up to always inflate `activity_fragment.xml`.

To make **SingleFragmentActivity** more flexible, you are going to enable a subclass to provide its own resource ID for the layout instead.

Creating a Layout with Two Fragment Containers



Change to two-pane layout file
(`CrimeListActivity.java`)

```
public class CrimeListActivity extends
SingleFragmentActivity {

    @Override
    protected Fragment createFragment() {
        return new CrimeListFragment();
    }

    @Override
    protected int getLayoutResId() {
        return R.layout.activity_twopane;
    }
}
```

Run **CriminalIntent** on a tablet device and confirm that you have two panes

Using an Alias Resource

In last slide, **CrimeListActivity** will also inflate the two-pane interface when running on a phone. You will fix this using an alias resources.

An alias resource is a resource that points to another resource. Alias resources live in `res/values/` and, by convention, are defined in a `refs.xml` file.

Create a default alias resource value (`res/values/refs.xml`)

```
<resources>
    <item name="activity_masterdetail" type="layout">@layout/activity_fragment</item>
</resources>
```

This resource's value is a reference to the single-pane layout. You can now use this resource ID in place of `R.layout.activity_fragment`.

Switch layout again (`CrimeListActivity.java`)

```
@Override
protected int getLayoutResId(){
    return R.layout.activity_twopane;
    return R.layout.activity_masterdetail;
}
```

Creating Tablet Alternative Alias Resources

Alternative alias for larger devices (res/values-sw600dp/refs.xml)

```
<resources>
<item name="activity_masterdetail" type="layout">@layout/activity_fragment</item>
<item name="activity_masterdetail" type="layout">@layout/activity_twopane</item>
</resources>
```

With a `-sw600dp` qualifier, you are saying, “Use this resource on any device whose smallest dimension is 600dp or greater.” This is a good rule of thumb for a tablet-sized screen.

The smallest width configuration qualifier was introduced in Android 3.2. This means that a tablet-sized device running Android 3.0 and 3.1 will not recognize it. To fix this use deprecated screen size qualifier `-xlarge`.

Alternative alias for pre-3.2 larger devices (res/values-xlarge/refs.xml)

```
<resources>
<item name="activity_masterdetail" type="layout">@layout/activity_twopane</item>
</resources>
```

The `-xlarge` qualifier contains resources to use on devices with a minimum size of 720x960dp. This qualifier will only be used for devices running versions earlier than Android 3.2.

Activity: Fragment Boss

One typical implementation of showing Crime Fragment in the tablet may be below

```
CrimeListFragment.onListItemClick(...)
```

```
public void onListItemClick(ListView l, View v, int position,
long id) {
    // Get the Crime from the adapter
    Crime crime =
    ((CrimeAdapter)getListAdapter()).getItem(position);
    // Stick a new CrimeFragment in the activity's layout
    Fragment fragment =
    CrimeFragment.newInstance(crime.getId());
    FragmentManager fm =
    getActivity().getSupportFragmentManager();
    fm.beginTransaction()
        .add(R.id.detailFragmentContainer, fragment)
        .commit();
}
```

This works, but it is not how stylish Android programmers do things.

Fragments are intended to be standalone, composable units. If you write a fragment that adds fragments to the activity's FragmentManager, then that fragment is making assumptions about how the hosting activity works, and your fragment is no longer a standalone, composable unit.

To maintain the independence of your fragments, you will delegate work back to the hosting activity by defining callback interfaces in your fragments.

The hosting activities will implement these interfaces to perform fragment-bossing duties and layout-dependent behavior.

Implementing CrimeListFragment.Callbacks

Add callback interface (CrimeListFragment.java)

```
public class CrimeListFragment extends ListFragment {  
    private ArrayList<Crime> mCrimes;  
    private boolean mSubtitleVisible;  
    private Callbacks mCallbacks;  
  
    /**  
     * Required interface for hosting activities.  
     */  
    public interface Callbacks {  
  
        void onCrimeSelected(Crime crime);  
    }  
    @Override  
    public void onAttach(Activity activity) {  
        super.onAttach(activity);  
        mCallbacks = (Callbacks)activity;  
    }  
    @Override  
    public void onDetach() {  
        super.onDetach();  
        mCallbacks = null;  
    }  
}
```

Implement callbacks (CrimeListActivity.java)

```
public class CrimeListActivity extends SingleFragmentActivity  
    implements CrimeListFragment.Callbacks {  
    @Override  
    protected Fragment createFragment() {  
        return new CrimeListFragment();  
    }  
    @Override  
    protected int getLayoutResId() {  
        return R.layout.activity_masterdetail;  
    }  
    public void onCrimeSelected(Crime crime) {  
    }  
}
```

Eventually, CrimeListFragment will call this method in `onListItemClick(...)` and also when the user chooses to create a new crime.

Implementing CrimeListFragment.Callbacks

When `onCrimeSelected(Crime)` is called, **CrimeListActivity** needs to do one of two things:

- if using the phone interface, start a new **CrimePagerActivity**
- if using the tablet interface, put a **CrimeFragment** in `detailFragmentContainer`

Conditional CrimeFragment startup (CrimeListActivity.java)

```
public void onCrimeSelected(Crime crime) {  
    if (findViewById(R.id.detailFragmentContainer) == null) {  
        // Start an instance of CrimePagerActivity  
        Intent i = new Intent(this, CrimePagerActivity.class);  
        i.putExtra(CrimeFragment.EXTRA_CRIME_ID, crime.getId());  
        startActivity(i);  
    } else {  
        FragmentManager fm = getSupportFragmentManager();  
        FragmentTransaction ft = fm.beginTransaction();
```

```
        Fragment oldDetail =  
            fm.findFragmentById(R.id.detailFragmentContainer);  
        Fragment newDetail =  
            CrimeFragment.newInstance(crime.getId());  
  
        if (oldDetail != null) {  
            ft.remove(oldDetail);  
        }  
  
        ft.add(R.id.detailFragmentContainer, newDetail);  
        ft.commit();  
    }  
}
```

Finally, in **CrimeListFragment**, you are going to call `onCrimeSelected(Crime)` in the places where you currently start a new **CrimePagerActivity**.

Calling All Callbacks!

Calling all callbacks! (CrimeListFragment.java)

```
public void onListItemClick(ListView l, View v, int position, long id) {  
    // Get the Crime from the adapter  
    Crime c = ((CrimeAdapter)getListAdapter()).getItem(position);  
    // Start an instance of CrimePagerActivity  
    Intent i = new Intent(getActivity(), CrimePagerActivity.class);  
    i.putExtra(CrimeFragment.EXTRA_CRIME_ID, c.getId());  
    startActivityForResult(i);  
    mCallbacks.onCrimeSelected(c);  
}
```

...

```
@TargetApi(11)  
@Override  
public boolean onOptionsItemSelected(MenuItem item) {  
    switch (item.getItemId()) {  
        case R.id.menu_item_new_crime:  
            Crime crime = new Crime();  
            CrimeLab.get(getActivity()).addCrime(crime);  
            Intent i = new Intent(getActivity(), CrimePagerActivity.class);  
            i.putExtra(CrimeFragment.EXTRA_CRIME_ID, crime.getId());  
            startActivityForResult(i);  
            ((CrimeAdapter)getListAdapter()).notifyDataSetChanged();  
            mCallbacks.onCrimeSelected(crime);  
            return true;  
        ...  
    }  
}
```

Run CriminalIntent on a tablet. Create a new crime, and a **CrimeFragment** will be added and shown in the detailFragmentContainer. Then view an old crime to see the **CrimeFragment** being swapped out for a new one.

Master and Detail View Now Wired Up



One problem, if you make changes to a crime, the list will not update to reflect them. Right now, you only reload the list immediately after adding a crime and in `CrimeListFragment.onResume()`.

But on a tablet, **CrimeListFragment** will stay visible alongside the **CrimeFragment**. The **CrimeListFragment** is not paused when the **CrimeFragment** appears, so it is never resumed. Thus, the list is not reloaded.

You can fix this problem with another callback interface – this one in **CrimeFragment**.

Implementing CrimeFragment.Callbacks 1/4

CrimeFragment will define the following interface:

```
public interface Callbacks {  
    void onCrimeUpdated(Crime crime);  
}
```

CrimeFragment will call `onCrimeUpdated(Crime)` on its hosting activity whenever changes are saved to the **Crime**. **CrimeListActivity** will implement `onCrimeUpdated(Crime)` to reload **CrimeListFragment**'s list.

Before you start with **CrimeFragment**'s interface, add a method to **CrimeListFragment** that can be called to reload **CrimeListFragment**'s list.

Add `updateUI()` method (`CrimeListFragment.java`)

```
public class CrimeListFragment extends ListFragment {  
    ...  
    public void updateUI() {  
        ((CrimeAdapter)getListAdapter()).notifyDataSetChanged();  
    }  
}
```

Implementing CrimeFragment.Callbacks 2/4

Add CrimeFragment callbacks
(CrimeFragment.java)

```
...
private ImageView mPhotoView;
private Button mSuspectButton;
private Callbacks mCallbacks;
// Required interface for hosting activities.
public interface Callbacks {
    void onCrimeUpdated(Crime crime);
}
@Override
public void onAttach(Activity activity) {
    super.onAttach(activity);
    mCallbacks = (Callbacks)activity;
}
@Override
public void onDetach() {
    super.onDetach();
    mCallbacks = null;
}
public static CrimeFragment newInstance(UUID crimeId) {
...
}
```

Then implement CrimeFragment.Callbacks in CrimeListActivity to reload the list in onCrimeUpdated(Crime).

Refresh crime list (CrimeListActivity.java)

```
public class CrimeListActivity extends SingleFragmentActivity
    implements CrimeListFragment.Callbacks, CrimeFragment.Callbacks {
...
public void onCrimeUpdated(Crime crime) {
    FragmentManager fm = getSupportFragmentManager();
    CrimeListFragment listFragment = (CrimeListFragment)
        fm.findFragmentById(R.id.fragmentContainer);
    listFragment.updateUI();
}
}
```

Implementing CrimeFragment.Callbacks 3/4

In **CrimeFragment.java**, add calls to `onCrimeUpdated(Crime)` when a Crime's title or solved status has changed.

```
@Override  
@TargetApi(11)  
public View onCreateView(LayoutInflater inflater, ViewGroup  
parent,  
    Bundle savedInstanceState) {  
    View v = inflater.inflate(R.layout.fragment_crime, parent, false);  
  
    ...  
  
    mTitleField = (EditText)v.findViewById(R.id.crime_title);  
    mTitleField.setText(mCrime.getTitle());  
    mTitleField.addTextChangedListener(new TextWatcher() {  
        public void onTextChanged(CharSequence c, int start, int  
before, int count) {  
            mCrime.setTitle(c.toString());  
            mCallbacks.onCrimeUpdated(mCrime);  
            getActivity().setTitle(mCrime.getTitle());  
        }  
  
        ...  
    });
```

```
mSolvedCheckBox = (CheckBox)v.findViewById(R.id.crime_solved);  
mSolvedCheckBox.setChecked(mCrime.isSolved());  
mSolvedCheckBox.setOnCheckedChangeListener(new  
OnCheckedChangeListener() {  
    public void onCheckedChanged(CompoundButton buttonView,  
boolean isChecked) {  
        // Set the crime's solved property  
        mCrime.setSolved(isChecked);  
        mCallbacks.onCrimeUpdated(mCrime);  
    }  
});  
...  
return v;  
}
```

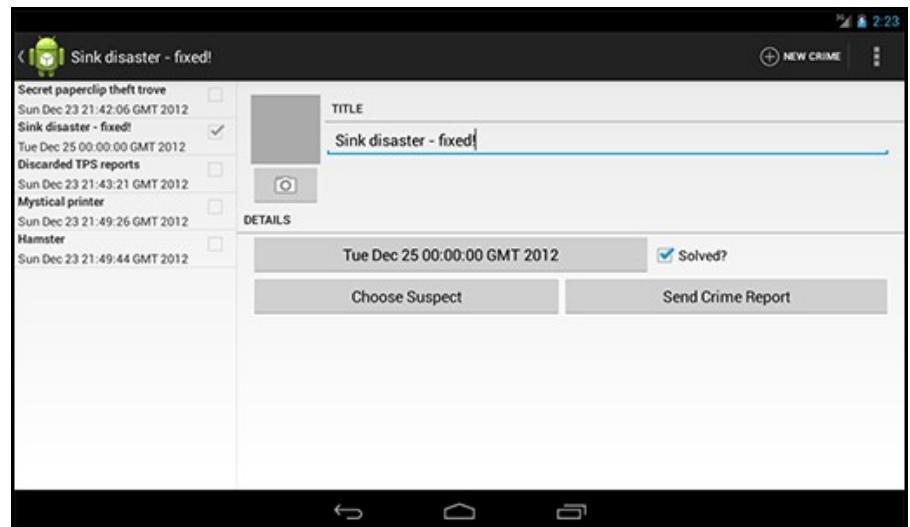
Implementing CrimeFragment.Callbacks 4/4

One problem, if you run CriminalIntent on a phone, it will *crash*. Remember, any activity that hosts **CrimeFragment** must implement CrimeFragment.Callbacks. You need to implement CrimeFragment.Callbacks in **CrimePagerActivity**.

For **CrimePagerActivity**, all you need is an empty implementation where onCrimeUpdated(Crime) does nothing. When **CrimePagerActivity** is hosting **CrimeFragment**, the only reloading of the list that is necessary is already done in onResume().

```
public class CrimePagerActivity extends FragmentActivity  
    implements CrimeFragment.Callbacks {  
  
    ...  
  
    public void onCrimeUpdated(Crime crime) {  
    }  
}
```

List reflects changes made in detail



Android UI Styles and Includes

Javed Hasan

BJIT Limited

Lesson Content

Remote Control App Features

- Make a Television Remote Control

Setting Up the Controller Components

- Setting Up RemoteControlActivity
- Setting Up RemoteControlFragment
 - Table Layout

Cleaning Up and Refactor Using Style

- Defining and Applying Styles in Layout

Wiring Up Views with Controller

- Buttons and Table Row Wiring Up

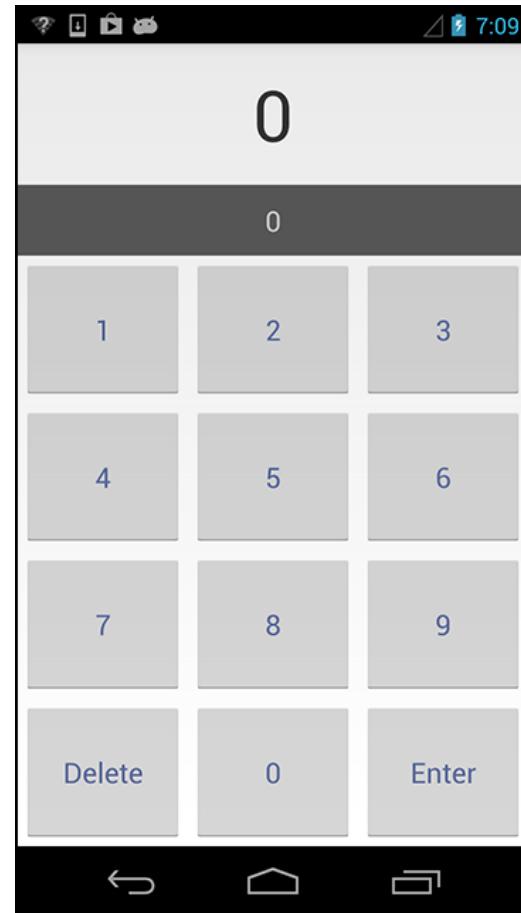
New Feature Overview

You will create a television remote control app. However, it will not actually control anything; it is just a playground for practicing using design tools.

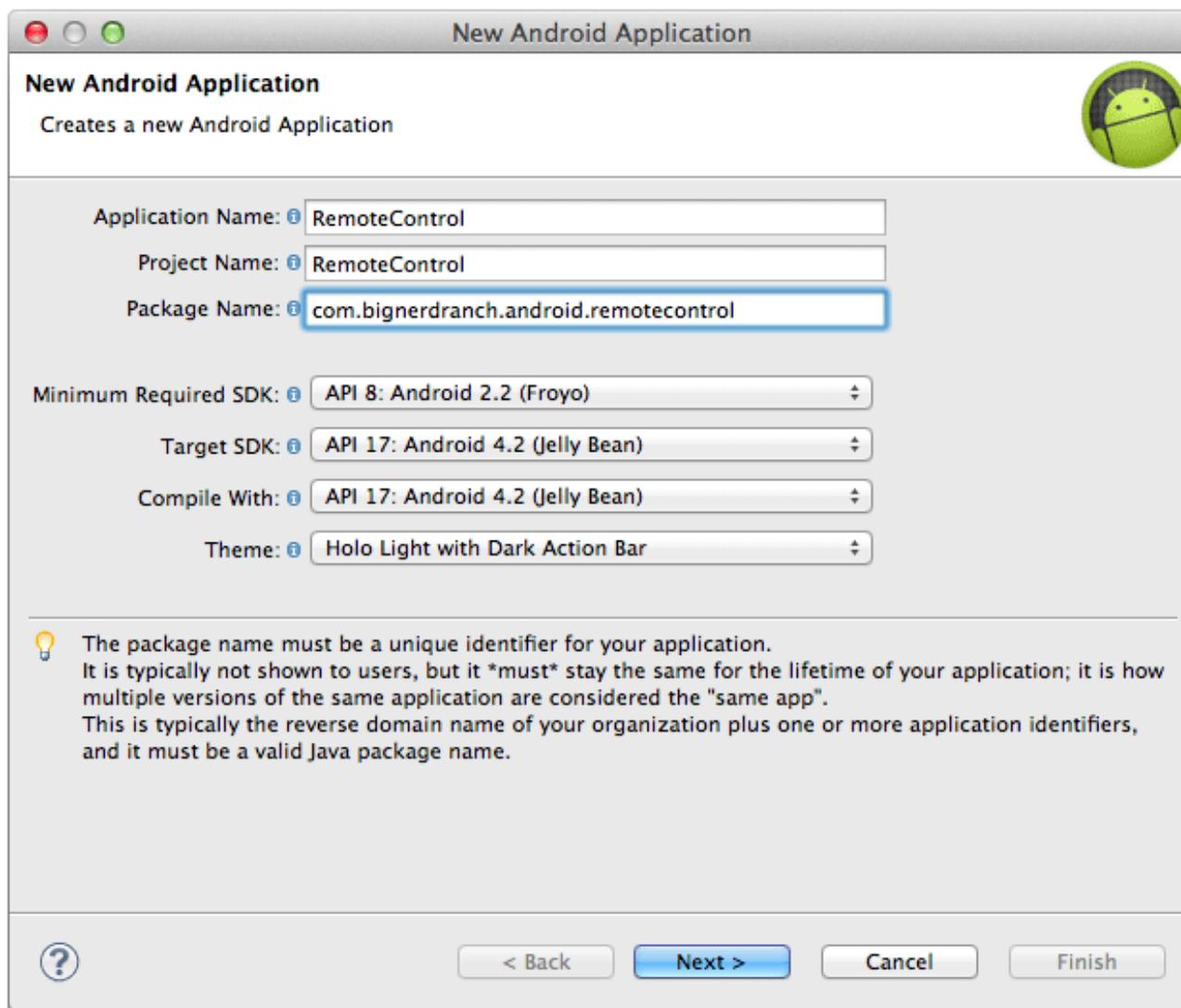
RemoteControl will have the single activity. The area at the top will display the current channel. The area underneath will display a new channel as it is being entered.

Pressing `Delete` will clear out the channel entry display.

Pressing `Enter` will change the channel, updating the current channel display and clearing the channel entry display.



Setting Up the RemoteControl Project



Setting up RemoteControlActivity

RemoteControlActivity set-up (RemoteControlActivity.java)

```
public class RemoteControlActivity extends Activity SingleFragmentActivity {  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_remote_control);  
    }  
    @Override  
    public boolean onCreateOptionsMenu(Menu menu) {  
        getMenuInflater().inflate(R.menu.activity_remote_control, menu);  
        return true;  
    }  
    @Override  
    protected Fragment createFragment() {  
        return new RemoteControlFragment();  
    }  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        requestWindowFeature(Window.FEATURE_NO_TITLE);  
        super.onCreate(savedInstanceState);  
    }  
}
```

Lock to portrait orientation (AndroidManifest.xml)

```
<activity  
    android:name="com.bignerdranch.android.remotecontrol.Remote  
    ControlActivity"  
    android:label="@string/app_name"  
    android:screenOrientation="portrait">  
    <intent-filter>  
        <action android:name="android.intent.action.MAIN" />  
        <category  
            android:name="android.intent.category.LAUNCHER" />  
    </intent-filter>  
</activity>
```

Setting up RemoteControlFragment Layout

```
<TableLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/fragment_remote_control_tableLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:stretchColumns="*" >

    <TextView
        android:id="@+id/fragment_remote_control_selectedTextView"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="2"
        android:gravity="center"
        android:text="0"
        android:textSize="50dp" />

    <TextView
        android:id="@+id/fragment_remote_control_workingTextView"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:layout_margin="15dp" />

    android:background="#555555"
    android:gravity="center"
    android:text="0"
    android:textColor="#cccccc"
    android:textSize="20dp" />

    <TableRow android:layout_weight="1" >
        <Button
            android:id="@+id/fragment_remote_control_zeroButton"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:text="0" />
        <Button
            android:id="@+id/fragment_remote_control_oneButton"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:text="1" />
        <Button
            android:id="@+id/fragment_remote_control_enterButton"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:text="Enter" />
    </TableRow>
</TableLayout>
```

Wiring Up Buttons in RemoteControlFragment 1/2

```
public class RemoteControlFragment extends Fragment {  
    private TextView mSelectedTextView;  
    private TextView mWorkingTextView;  
    @Override  
    public View onCreateView(LayoutInflater inflater, ViewGroup parent, Bundle savedInstanceState) {  
        View v = inflater.inflate(R.layout.fragment_remote_control, parent, false);  
        mSelectedTextView = (TextView)v.findViewById(R.id.fragment_remote_control_selectedTextView);  
        mWorkingTextView = (TextView)v.findViewById(R.id.fragment_remote_control_workingTextView);  
        View.OnClickListener numberButtonListener = new View.OnClickListener() {  
            public void onClick(View v) {  
                TextView textView = (TextView)v;  
                String working = mWorkingTextView.getText().toString();  
                String text = textView.getText().toString();  
                if (working.equals("0")) {  
                    mWorkingTextView.setText(text);  
                } else {  
                    mWorkingTextView.setText(working + text);  
                }  
            }  
        };  
    }  
};
```

Wiring Up Buttons in RemoteControlFragment 2/2

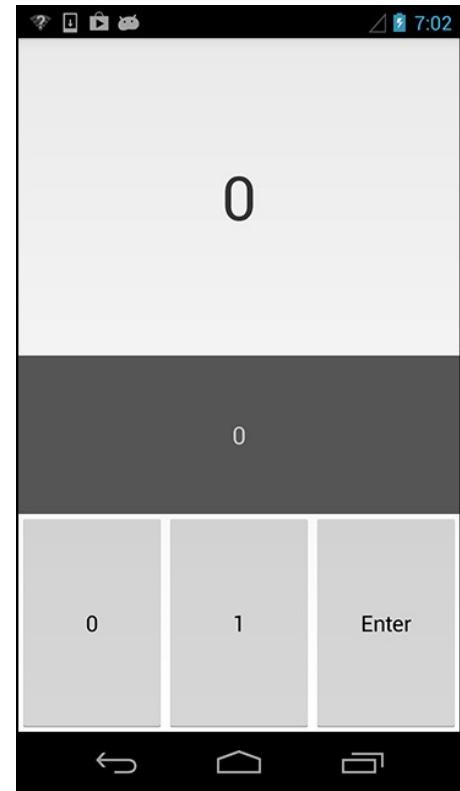
```
Button zeroButton = (Button)v.findViewById(R.id.fragment_remote_control_zeroButton);
zeroButton.setOnClickListener(numberButtonListener);

Button oneButton = (Button)v.findViewById(R.id.fragment_remote_control_oneButton);
oneButton.setOnClickListener(numberButtonListener);

Button enterButton = (Button) v.findViewById(R.id.fragment_remote_control_enterButton);
enterButton.setOnClickListener(new View.OnClickListener() {

    public void onClick(View v) {
        CharSequence working = mWorkingTextView.getText();
        if (working.length() > 0)
            mSelectedTextView.setText(working);
        mWorkingTextView.setText("0");
    }
});

return v;
}
```



While you have three buttons here, you only need two click listeners. That is because you can use the same click listener for both number buttons. Run RemoteControl. You should see a simple three-button remote control app with a binary interface.

Defining Android Styles

In layout XML, each button is identical. That is fine for now, but if you want to add an attribute to a button, you will have to repeat your work three times.

Android has UI styles, which are meant to eliminate that repetition. Each style defines a set of XML attribute-value pairs. Styles can be organized into a hierarchy: a child has the same attributes and values as its parent, but may override them or add additional values.

The Android project wizard has created a `styles.xml` file. You will move the attributes that are common to all your buttons out of the individual buttons and into a new `RemoteButton` style.

Initial RemoteControl styles
(`values/styles.xml`)

```
<resources>
    <style name="AppTheme" parent="android:Theme.Light" />
    <style name="RemoteButton">
        <item name="android:layout_width">0dp</item>
        <item name="android:layout_height">match_parent</item>
    </style>
</resources>
```

Applying the Styles

Applying styles

(layout/fragment_remote_control.xml)

```
<TableLayout  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    ...  
    >  
    ...  
  
<TableRow android:layout_weight="1">  
    <Button  
        android:id="@+id/fragment_remote_control_zeroButton"  
        android:layout_width="0dp"  
        android:layout_height="match_parent"  
        style="@style/RemoteButton"  
        android:text="0" />  
    <Button  
        android:id="@+id/fragment_remote_control_oneButton"  
        android:layout_width="0dp"  
        android:layout_height="match_parent"  
        style="@style/RemoteButton"  
        android:text="1" />
```

```
<Button  
    android:id="@+id/fragment_remote_control_enterButton"  
    android:layout_width="0dp"  
    android:layout_height="match_parent"  
    style="@style/RemoteButton"  
    android:text="Enter" />  
</TableRow>  
</TableLayout>
```

Run RemoteControl. Your remote looks exactly the same as it did before. But your layout looks cleaner and you are repeating yourself less.

More on Applying Style on Layout 1/2

Let's flesh out the remote control interface to a full, luxurious 12-key version, complete with ten deluxe decimal digits.

You need to create a 3 x 4 array of nearly identical buttons.

You can create one row of three buttons and then use it four times.

A row of three buttons (layout/button_row.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<TableRow xmlns:android="http://schemas.android.com/apk/res/android" >
    <Button style="@style/RemoteButton" />
    <Button style="@style/RemoteButton" />
    <Button style="@style/RemoteButton" />
</TableRow>
```

More on Applying Style on Layout 2/2

Including the button rows in the main layout

(layout/fragment_remote_control.xml)

```
<TableRow android:layout_weight="1">
    <Button
        android:id="@+id/fragment_remote_control_zeroButton"
        style="@style/RemoteButton"
        android:text="0" />
    <Button
        android:id="@+id/fragment_remote_control_oneButton"
        style="@style/RemoteButton"
        android:text="1" />
    <Button
        android:id="@+id/fragment_remote_control_enterButton"
        style="@style/RemoteButton"
        android:text="Enter" />
</TableRow>
```

```
<include
    android:layout_weight="1"
    layout="@layout/button_row" />
<include
    android:layout_weight="1"
    layout="@layout/button_row" />
<include
    android:layout_weight="1"
    layout="@layout/button_row" />
<include
    android:layout_weight="1"
    layout="@layout/button_row" />
```

You may have noticed that the three buttons you defined in layout/button_row.xml did not have ids. Since you included this button row four times, if you had given the buttons ids, then they would not be unique. Similarly, the buttons do not have any text defined. It's okay, you can handle it all in code.

Wiring Up the Number Buttons

Wiring up the number buttons
(RemoteControlFragment.java)

```
View.OnClickListener numberButtonListener = new View.OnClickListener() {  
    ...  
};  
  
Button zeroButton =  
(Button)v.findViewById(R.id.fragment_remote_control_zeroButton);  
zeroButton.setOnClickListener(numberButtonListener);  
  
Button oneButton =  
(Button)v.findViewById(R.id.fragment_remote_control_oneButton);  
oneButton.setOnClickListener(numberButtonListener);  
  
TableLayout tableLayout = (TableLayout)v  
    .findViewById(R.id.fragment_remote_control_tableLayout);  
  
int number = 1;  
  
for (int i = 2; i < tableLayout.getChildCount() - 1; i++) {  
    TableRow row = (TableRow)tableLayout.getChildAt(i);  
  
    for (int j = 0; j < row.getChildCount(); j++) {  
        Button button = (Button)row.getChildAt(j);  
  
        button.setText("" + number);  
  
        button.setOnClickListener(numberButtonListener);  
  
        number++;  
    }  
}
```

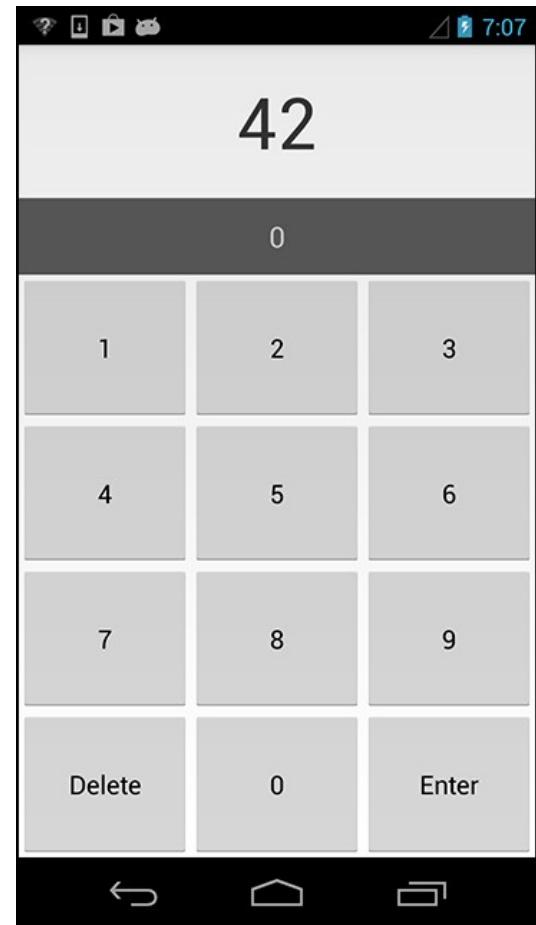
The for loop starts at index 2 to skip the two text views and then steps through each button in the first three rows.

For each button, it sets the same numberButtonListener you created earlier and sets the text to a string of the appropriate number.

Wiring Up the Last Row

The last row is trickier: you have to deal with the special cases of the **Delete** and **Enter** buttons.

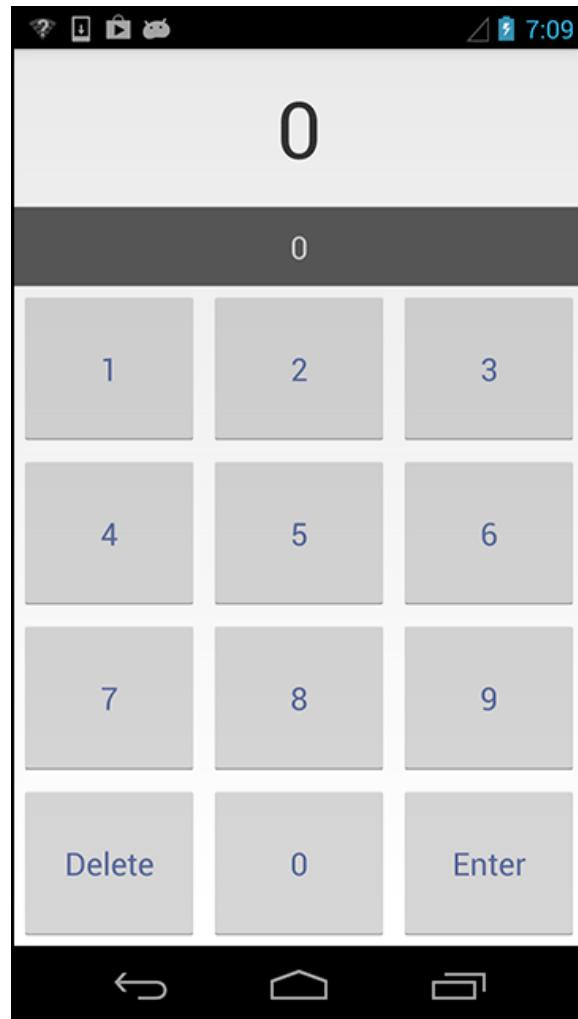
```
for (int i = 2; i < tableLayout.getChildCount() - 1; i++) {  
    ...  
}  
  
TableRow bottomRow = (TableRow)tableLayout.getChildAt(tableLayout.getChildCount() - 1);  
Button deleteButton = (Button)bottomRow.getChildAt(0);  
deleteButton.setText("Delete");  
deleteButton.setOnClickListener(new View.OnClickListener() {  
    public void onClick(View v) {  
        mWorkingTextView.setText("0");  
    }  
});  
  
Button zeroButton = (Button)bottomRow.getChildAt(1);  
zeroButton.setText("0");  
zeroButton.setOnClickListener(numberButtonListener);  
  
Button enterButton = (Button)v.findViewById(R.id.fragment_remote_control_enterButton);  
Button enterButton = (Button)bottomRow.getChildAt(2);  
enterButton.setText("Enter");  
enterButton.setOnClickListener(new View.OnClickListener() {  
    ...  
});
```



Little More on Styles

Tweaking your style a little bit
(values/styles.xml)

```
<style name="RemoteButton">  
    <item name="android:layout_width">0dp</item>  
    <item name="android:layout_height">match_parent</item>  
    <item name="android:textColor">#556699</item>  
    <item name="android:textSize">20dp</item>  
    <item name="android:layout_margin">3dp</item>  
</style>
```



Challenge: Style Inheritance

Your `Delete` and `Enter` buttons are sitting modestly at the bottom of the screen with the same style as all the number buttons. These “action” buttons need a bolder style.

For this small challenge, make those buttons look special. Create a new button style that inherits from your `RemoteButton` style and sets a bold text style attribute. Then make your bottom row use your new style for the first and third buttons. Creating a style that inherits from another is simple.

There are two ways to do it. One is to set the `parent` attribute of your style to the name of the style you want to inherit from. The easier way is to just prefix your style name with your parent’s style, then a dot – e.g., `ParentStyleName.MyStyleName`.

Android UI

XML Drawables and 9-Patches

Javed Hasan
BJIT Limited

Lesson Content

Remote Control App New Features

- Unique Look and Feel in the UI

XML Drawables

- Shape Drawables
- State List Drawables
- Layer List and Inset Drawables

Using 9-patch Images

- What is 9-patch image, Draw 9-patch image

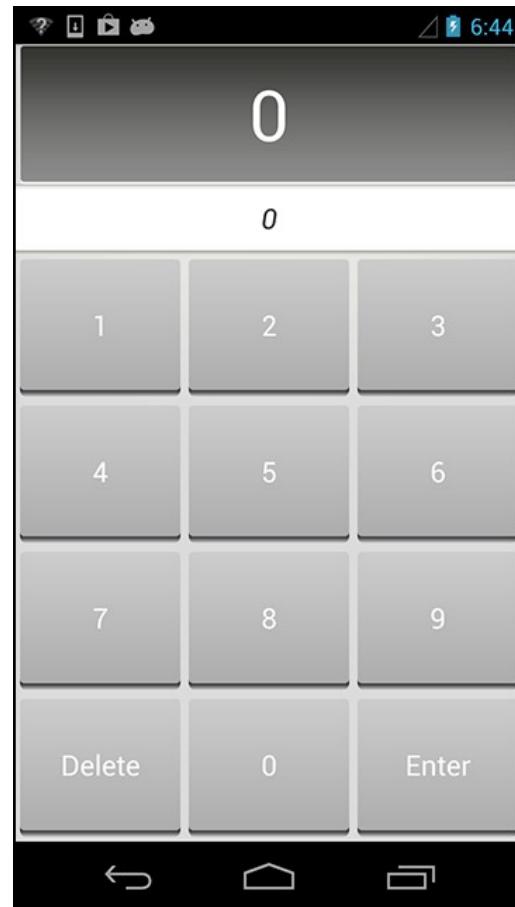
New Feature Overview

You built a TV remote control interface using advanced layout tips and tricks. It uses the stock Android look and feel for all of its buttons. Here, you will use two new tools to give it a completely unique look and feel.

Both of these tools are drawable. Android calls anything that is intended to be drawn to the screen a drawable, whether it is an abstract shape, a clever bit of code that subclasses the **Drawable** class, or a bitmap image.

You will see several kinds of drawables: state list drawables, shape drawables, layer list drawables, and nine patch drawables. First 3 drawables are defined in XML file. So, they are grouped in XML Drawables.

RemoteControl Makeover

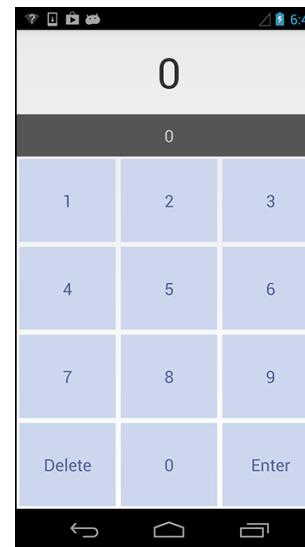


XML Drawables

An attempt at changing the button's color
(values/styles.xml)

```
<style name="RemoteButton">  
    <item name="android:layout_width">0dp</item>  
    <item name="android:layout_height">match_parent</item>  
    <item name="android:textColor">#556699</item>  
    <item name="android:textSize">20dp</item>  
    <item name="android:layout_margin">3dp</item>  
    <item name="android:background">#ccd7ee</item>  
</style>
```

What Happened?



The 3-dimensional look of the buttons is gone. If you press one of them, you will find that it no longer changes its appearance when you press it, either.

Why did all that happen from one innocent attribute change? The **Button** class is not much more than a **View** with a default style applied to it. The style comes from whatever theme you have selected, and sets a **Drawable** as the background. That background drawable was responsible for the 3-D look and the appearance change.

Making a Shape Drawable Button

(drawable/button_shape_normal.xml)

```
<?xml version="1.0" encoding="utf-8"?>

<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle" >
    <corners android:radius="3dp" />
    <gradient
        android:angle="90"
        android:endColor="#cccccc"
        android:startColor="#acacac" />
</shape>
```

Update button style (values/styles.xml)

```
<style name="RemoteButton">
    <item name="android:layout_width">0dp</item>
    <item name="android:layout_height">match_parent</item>
    <item name="android:textColor">#556699</item>
    <item name="android:textSize">20dp</item>
    <item name="android:layout_margin">3dp</item>
    <item name="android:background">#ced7ee</item>
    <item name="android:background">@drawable/button_shape_normal</item>
</style>
```

Now with rounded-corner buttons



State List Drawables 1/3

State list drawables allow you to display a different drawable for each state its associated View is in.

There are a lot of different states, but here you are only interested in whether the button is being pressed or not.

Creating the shape for a pressed button
(drawable/button_shape_pressed.xml)

```
<?xml version="1.0" encoding="utf-8"?>  
  
<shape  
    xmlns:android="http://schemas.android.com/apk/res/android"  
        android:shape="rectangle">  
            <corners android:radius="3dp"/>  
            <gradient  
                android:angle="90"  
                android:angle="270"  
                android:endColor="#cccccc"  
                android:startColor="#acacac"/>  
        </shape>
```

State List Drawables 2/3

Creating an interactive button shape (drawable/button_shape.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:drawable="@drawable/button_shape_normal"
        android:state_pressed="false"/>
    <item android:drawable="@drawable/button_shape_pressed"
        android:state_pressed="true"/>
</selector>
```

State-sensitive button text color (drawable/button_text_color.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_pressed="false" android:color="#ffffffff"/>
    <item android:state_pressed="true" android:color="#556699"/>
</selector>
```

State List Drawables 3/3

Updating the button style (values/styles.xml)

```
<style name="RemoteButton">  
    <item name="android:layout_width">0dp</item>  
    <item name="android:layout_height">match_parent</item>  
    <item name="android:textColor">#556699</item>  
    <item name="android:textSize">20dp</item>  
    <item name="android:layout_margin">3dp</item>  
    <item  
        name="android:background">@drawable/button_shape_normal</item>  
    <item name="android:background">@drawable/button_shape</item>  
    <item name="android:textColor">@drawable/button_text_color</item>  
</style>
```

A Pressed Button



Run RemoteControl. Check out the new look of your pressed button.

Shadow Affect: Layer List and Inset Drawables 1/4

First, you will create a shadow with the same shape as your current button drawable. Then you will combine it with the current button using layer-list, and offset the bottom edge of the button a bit using inset so that the shadow becomes visible underneath.

The layer list contains multiple Drawables, ordered from back to front as they will be drawn on the screen.

The second drawable in your list is an inset drawable. All it does is inset the bottom of the drawable you already created by 5dp, so that it sits above the shadow you created.

Normal button shadows
(drawable/button_shape_shadowed.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<layer-list xmlns:android="http://schemas.android.com/apk/res/android" >
    <item>
        <shape android:shape="rectangle" >
            <corners android:radius="5dp" />
            <gradient
                android:angle="90"
                android:centerColor="#303339"
                android:centerY="0.05"
                android:endColor="#000000"
                android:startColor="#00000000" />
        </shape>
    </item>
    <item>
        <inset
            android:drawable="@drawable/button_shape"
            android:insetBottom="5dp" />
    </item>
</layer-list>
```

Shadow Affect: Layer List and Inset Drawables 2/4

Change button style one last time (values/styles.xml)

```
<style name="RemoteButton">  
    <item name="android:layout_width">0dp</item>  
    <item name="android:layout_height">match_parent</item>  
    <item name="android:textSize">20dp</item>  
    <item name="android:layout_margin">3dp</item>  
    <item name="android:background">@drawable/button_shape</item>  
    <item name="android:background">@drawable/button_shape_shadowed</item>  
    <item name="android:textColor">@drawable/button_text_color</item>  
</style>
```

Shadow Affect: Layer List and Inset Drawables 3/4

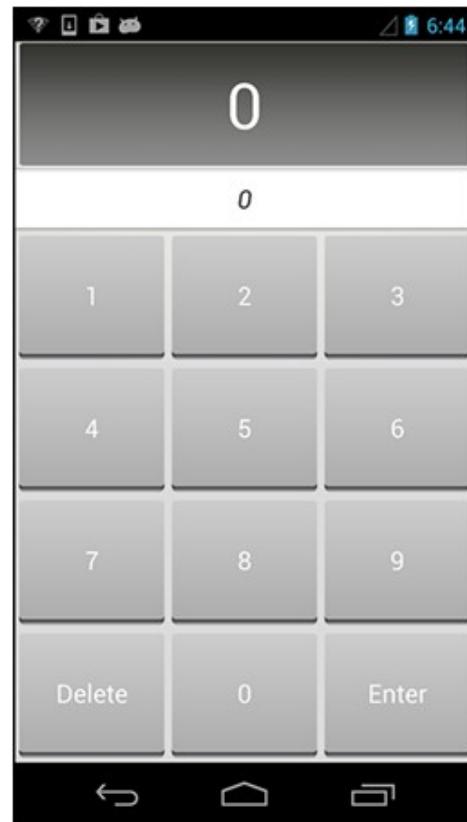
One last change: create a gradient drawable for the entire main view, for a subtle lighting effect. Create a new drawable file named `remote_background.xml` with a shape root element.

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android" >
    <gradient
        android:centerY="0.05"
        android:endColor="#dbdbdb"
        android:gradientRadius="500"
        android:startColor="#f4f4e9"
        android:type="radial" />
</shape>
```

Shadow Affect: Layer List and Inset Drawables 4/4

Applying the drawable to the layout
(layout/fragment_remote_control.xml)

```
<TableLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/fragment_remote_control_tableLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/remote_background"
    android:stretchColumns="*" >
```



Using 9-patch Images

For stretchable UI elements like button backgrounds, Android provides a tool called the 9-patch.

Your next task is to fix up the two **TextViews** at the top of the screen. You will keep your button layout the same, but use stretchable drawables for their backgrounds.

Both of them are much narrower than your **TextViews**, so that they do not take up much space.

Channel Display Area



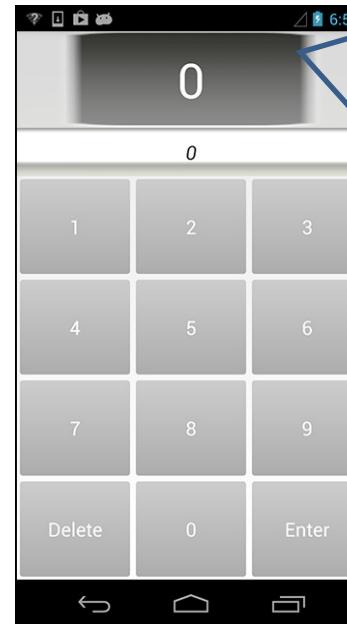
Channel Entry Area



Add the Drawables without 9-patch

```
(layout/fragment_remote_control.xml)
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
... >
<TextView
    android:id="@+id/fragment_remote_control_selectedTextView"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="2"
    android:background="@drawable/window"
    android:gravity="center"
    android:text="0"
    android:textColor="#ffffffff"
    android:textSize="50dp" />
<TextView
    android:id="@+id/fragment_remote_control_workingTextView"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_margin="15dp"
    android:layout_weight="1"
    android:background="#555555"
    android:background="@drawable/bar"
    android:gravity="center"
    android:text="0"
    android:textColor="#cccccc"
    android:textStyle="italic"
    android:textSize="20dp" />
...
</TableLayout>
```

Shattered Dreams



Each image was evenly stretched in all dimensions to fill the view.

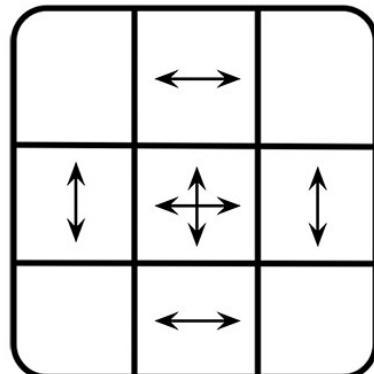
Your images are all fuzzed out, and your bottom **TextView's** digits are not properly centered.

Add the Drawables with 9-patch 1/3

A 9-patch image file is specially formatted so that Android knows which portions can and cannot be scaled. Done properly, this ensures that the edges and corners of your background remain consistent with the image as it was created.

Why are they called 9-patches? A 9-patch breaks your image into a 3×3 grid – a grid with 9 sections, or patches. The corners of the grid remain unscaled, the sides are only scaled in one dimension, and the center is scaled in both dimensions.

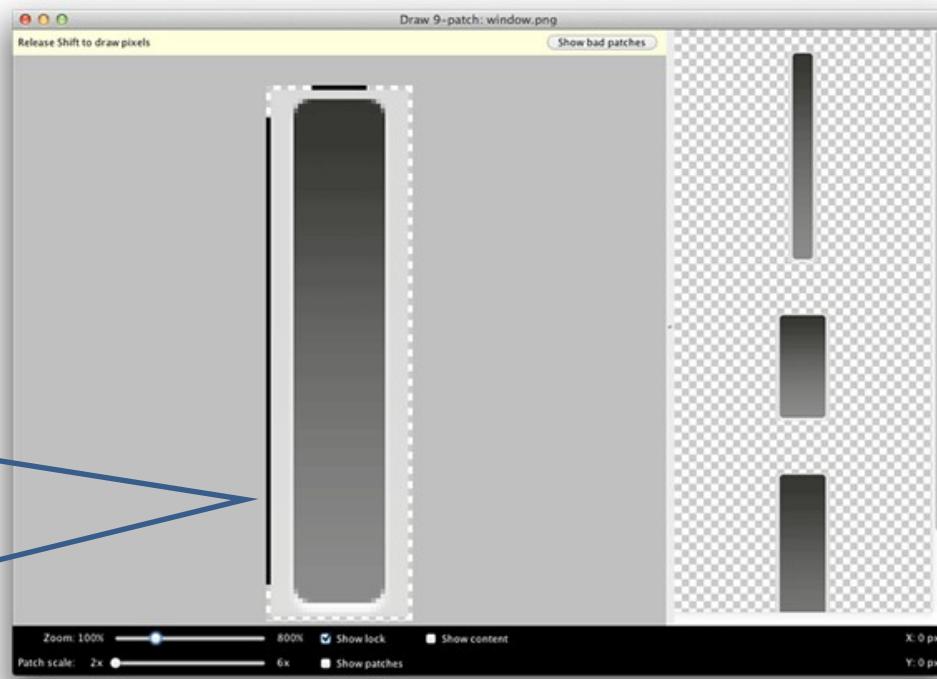
What a 9-patch does



Add the Drawables with 9-patch 2/3

A 9-patch image is like a regular png image in everything except two aspects: its filename ends with .9.png, and it has an additional one pixel border around the edge. This border is used to specify the location of the center square of the nine patch. Border pixels are drawn black to indicate the center and transparent to indicate the edges.

Channel display window 9-patch



You can create a 9-patch using `draw9patch` tool provided with Android SDK.

Fill in black pixels on the top and left borders to mark the stretchable regions of the 9-patch image for `window.png`. Save as `window_patch.9.png`.

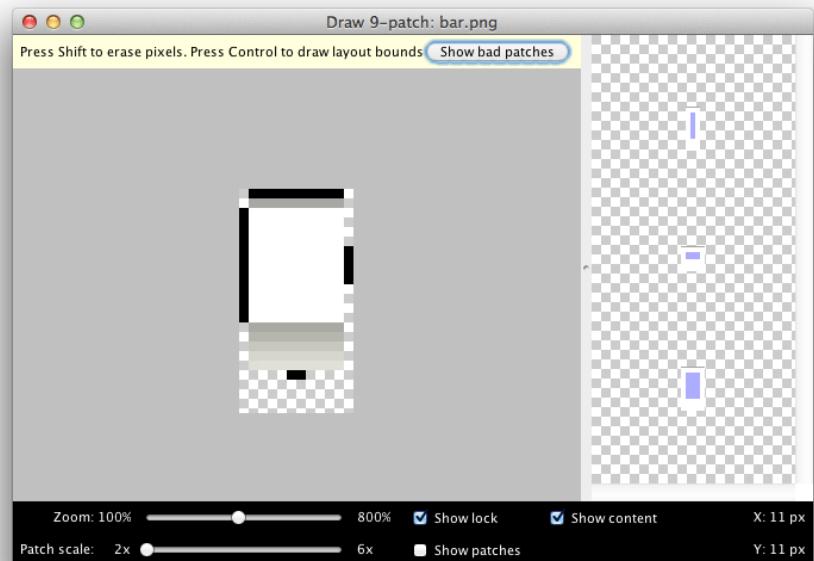
Add the Drawables with 9-patch 3/3

The top and left borders mark the stretchable region of the image.

Bottom and Right borders define an optional drawable region for the 9-patch image. The drawable region is the area where content (usually text) should be rendered. If you do not include a drawable region, it defaults to be the same as your stretchable region.

Next, edit a 9-patch for bar.png. Use the drawable region here to center your text properly vertically and horizontally, as well as provide for a 4 pixel margin on all sides. When you finish, save out to bar_patch.9.png.

Channel entry area 9-patch



Switch to Use 9-patches

(layout/fragment_remote_control.xml)

```
<TableLayout  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    ... >  
  
<TextView  
    android:id="@+id/fragment_remote_control_selectedTextView"  
    android:layout_width="match_parent"  
    android:layout_height="0dp"  
    android:layout_weight="2"  
    android:background="@drawable/window"  
    android:background="@drawable/window_patch"  
    android:gravity="center"  
    android:text="0"  
    android:textSize="50dp" />  
  
<TextView  
    android:id="@+id/fragment_remote_control_workingTextView"  
    android:layout_width="match_parent"  
    android:layout_height="0dp"  
    android:layout_margin="15dp"  
    android:layout_weight="1"  
    android:background="@drawable/bar"  
    android:background="@drawable/bar_patch"
```

```
        android:gravity="center"  
        android:text="0"  
        android:textColor="#cccccc"  
        android:textSize="20dp" />  
        ...  
</TableLayout>
```

Remote Control Makeover

