CS 403X Mobile and Ubiquitous Computing
Lecture 4: AdapterViews, Intents, Fragments
Audio/Video, Camera

Emmanuel Agu
Android UI Components: Controls
Checkbox

- Has 2 states: **checked** and **unchecked**
- Clicking on checkbox toggles between these 2 states
- Used to indicate a choice (e.g. Add rush delivery)
- Since Checkbox widget inherits from TextView, its properties (e.g. **android:textColor**) can be used to format checkbox
- XML code to create Checkbox:

```xml
<CheckBox xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@id/check"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/unchecked"/>
```
Checkbox Example Java Code

```java
package com.commonsware.android.checkbox;

import android.app.Activity;
import android.os.Bundle;
import android.widget.CheckBox;
import android.widget.CompoundButton;

public class CheckBoxDemo extends Activity implements
    CompoundButton.OnCheckedChangeListener {
    CheckBox cb;

    @Override
    public void onCreate(Bundle icicle) {
        super.onCreate(icicle);
        setContentView(R.layout.main);

        cb=(CheckBox)findViewById(R.id.check);
        cb.setOnCheckedChangeListener(this);
    }

    public void onCheckedChanged(CompoundButton buttonView,
                                  boolean isChecked) {
        if (isChecked) {
            cb.setText(R.string.checked);
        } else {
            cb.setText(R.string.unchecked);
        }
    }
}
```

- **Checkbox inherits from CompoundButton**
- **Register listener OnCheckedChangeListener to be notified when checkbox state changes**
- **Callback, called When checkbox state changes**
Checkbox Example Result

- This checkbox is: unchecked
- This checkbox is: checked
Other Android Controls

- **ToggleButton and Switches**
  - Like CheckBox has 2 states
  - However, visually shows states on and off text

- **XML code to create ToggleButton**

  ```xml
  <?xml version="1.0" encoding="utf-8"?>
  <ToggleButton xmlns:android="http://schemas.android.com/apk/res/android"
                android:id="@+id/toggle"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content" />
  ```
RadioButton and RadioGroup

- Select only 1 option from a set

- set onClick method for each button
  - generally same method

- Inherits from CompoundButton which inherits from TextView
  - Format using TextView properties (font, style, color, etc)
SeekBar

- a slider
- Subclass of progress bar
- implement a `SeekBar.OnSeekBarChangeListener` to respond to changes in setting
Auto Complete Options

- Depending on EditText inputType suggestions can be displayed
  - works on actual devices

- Other options for exist for auto complete from list
  - AutoCompleteTextView
    - choose one option
  - MultiAutoCompleteTextView
    - choose multiple options (examples tags, colors)
Spinner Controls

- User **must** select from a set of choices
Indicators

- Variety of built in indicators in addition to TextView
- ProgressBar
- RatingBar
- Chronometer
- DigitalClock
- AnalogClock
Dynamic and Data-Driven Layouts
Data Driven Containers

- Sometimes want to read in data (e.g. from file) => organize, display

- Dynamic Layout in which child views are generated from data

- ListView
  - vertical scroll, horizontal row entries, pick item
Data Driven Containers

- **GridView**
  - specified number of rows and columns

- **GalleryView**
  - horizontal scrolling list, typically images
AdapterView

- ListView, GridView, and GalleryView are all sub classes of AdapterView
- Adapter generates child Views from some data source and populates the larger View.
  - E.g. Data is adapted into cells of GridView

- Most common Adapters (sources)
  - CursorAdapter: read data from database
  - ArrayAdapter: read data from resource, typically an XML file

- The adapter
  - Creates Views (widgets) each element in data source
  - Fills layout (List, Grid, Gallery) with the created Views
Using ArrayAdapter

- Wraps adapter around a Java array of menu items or `java.util.List` instance

```java
String[] items={"this", "is", "a", "really", "silly", "list"};
new ArrayAdapter<String>(this,
    android.R.layout.simple_list_item_1,
    items);
```

- In example, `android.R.layout.simple_list_item_1` turns strings into TextView objects
- TextView widgets shown in list using this ArrayAdapter
Example: Creating ListView using AdapterArray

- See project from textbook: theSelection/List sample

- Want to create the following ListView from the following strings

```java
```
Example: Creating ListView using AdapterArray

- First create LinearLayout

```xml
<LinearLayout
    xmlns:android="http://schemas.android.com/"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:id="@+id/selection"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"/>
    <ListView
        android:id="@android:id/list"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>
</LinearLayout>
```
Example: Creating ListView using ArrayAdapter

```java
package com.commonsware.android.list;

import android.app.ListActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.TextView;

public class ListViewDemo extends ListActivity {
    private TextView selection;
    private static final String[] items=
        {"lorem", "ipsum", "dolor", "sit", "amet",
        "consectetuer", "adipiscing", "elit", "morbi", "vel",
        "ligula", "vitae", "arcu", "aliquet", "mollis",
        "etiam", "vel", "erat", "placerat", "ante",
        "porttitor", "sodales", "pellentesque", "augue", "purus"};

    @Override
    public void onCreate(Bundle icicle) {
        super.onCreate(icicle);
        setContentView(R.layout.main);
        setListAdapter(new ArrayAdapter<String>(this,
            android.R.layout.simple_list_item_1, items));

        selection=(TextView)findViewById(R.id.selection);
    }

    @Override
    public void onListItemClick(ListView parent, View v, int position, long id) {
        selection.setText(items[position]);
    }
}
```
Starting Activity 2 from Activity 1
Why would we want to do this?
Ref: Android Nerd Ranch pg 89

- May want to allow user to cheat by getting answer to quiz
- Second screen pops up, displays “Are you sure?, show Answer”

Activity 1

Click here to cheat if you don’t know the answer

Activity 2

Click here to cheat if you don’t know the answer
Layout for Screen 2

- First create layout for screen 2

```
<?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"
    android:gravity="center">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="24dp"
        android:text="@string/warning_text" />

    <TextView
        android:id="@+id/answerTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="24dp" />

    <Button
        android:id="@+id/showAnswerButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/show_answer_button" />

</LinearLayout>
```
Declare New Activity in AndroidManifest.xml

- Create new activity in Android Studio, override onCreate()

```java
public class CheatActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_cheat);
    }
}
```

- Then declare new Activity in AndroidManifest

```xml
<activity
    android:name="com.bignerdranch.android.geoquiz.CheatActivity"
    android:label="@string/app_name"
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```
Starting Activity 2 from Activity 1

- Activity 1 starts activity 2 **through** the Android OS
- Activity 1 starts activity 2 by calling `startActivity(Intent)`
- Passes Intent (object for communicating with Android OS)

- Intent specifies which Activity OS ActivityManager should start
Starting Activity 2 from Activity 1

- Intents have many different constructors. We will use form:

\[
\text{public Intent(Context packageContext, Class<? } \text{ cls)}
\]

- Actual code looks like this

```java
... 
mCheatButton = (Button) findViewById(R.id.cheat_button);
mCheatButton.setOnClickListener(new View.OnClickListener()
{
    @Override
    public void onClick(View v) {
        Intent i = new Intent(QuizActivity.this, CheatActivity.class);
        startActivity(i);
    }
}); 
updateQuestion();
```
Final Words on Intents

- Previous example is called an **explicit intent** because Activity 1 and activity 2 are in the same app.
- If Activity 2 were in another app, an **implicit intent** would have to be created instead.
- Can also pass data between Activities 1 or 2:
  - E.g. New Activity 2 can tell activity 1 if user checked answer.

See Android Nerd Ranch for more details
Intents
Intents

- Allows apps to use Android applications and components
  - start activities
  - start services
  - deliver broadcasts
- Also allows other apps to use components of our apps

- More details at:
Intents

- "An intent is an abstract description of an operation to be performed"

- Intents consist of:
  - **Action** (what to do, example visit a web page)
  - **Data** (to perform operation on, example web page url)

- Commands related with Intents: `startActivity`, `startActivityForResult`, `startService`, `bindService`
Intent Object Info

- data for component that receives the intent (e.g. Activity 2)
  - action to take
  - data to act on

- data for the Android system
  - category of component to handle intent (activity, service, broadcast receiver)
  - instructions on how to launch component if necessary
Recall: Inside AndroidManifest.xml

Your package name

Android version

List of activities (screens) in your app

Action of intent

Category of intent
## Intent Action

<table>
<thead>
<tr>
<th>Constant</th>
<th>Target component</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTION_CALL</td>
<td>activity</td>
<td>Initiate a phone call.</td>
</tr>
<tr>
<td>ACTION_EDIT</td>
<td>activity</td>
<td>Display data for the user to edit.</td>
</tr>
<tr>
<td>ACTION_MAIN</td>
<td>activity</td>
<td>Start up as the initial activity of a task, with no data input and no returned output</td>
</tr>
<tr>
<td>ACTION_SYNC</td>
<td>activity</td>
<td>Synchronize data on a server with data on the mobile device.</td>
</tr>
<tr>
<td>ACTION_BATTERY_LOW</td>
<td>broadcast receiver</td>
<td>A warning that the battery is low.</td>
</tr>
<tr>
<td>ACTION_HEADSET_PLUG</td>
<td>broadcast receiver</td>
<td>A headset has been plugged into the device, or unplugged from it.</td>
</tr>
<tr>
<td>ACTION_SCREEN_ON</td>
<td>broadcast receiver</td>
<td>The screen has been turned on.</td>
</tr>
<tr>
<td>ACTION_TIMEZONE_CHANGED</td>
<td>broadcast receiver</td>
<td>The setting for the time zone has changed.</td>
</tr>
</tbody>
</table>
### Intent Info - *Category*

- String with more information on what kind of component should handle Intent

<table>
<thead>
<tr>
<th>Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CATEGORY_BROWSABLE</td>
<td>The target activity can be safely invoked by the browser to display data referenced by a link — for example, an image or an e-mail message.</td>
</tr>
<tr>
<td>CATEGORY_GADGET</td>
<td>The activity can be embedded inside of another activity that hosts gadgets.</td>
</tr>
<tr>
<td>CATEGORY_HOME</td>
<td>The activity displays the home screen, the first screen the user sees when the device is turned on or when the <em>Home</em> button is pressed.</td>
</tr>
<tr>
<td>CATEGORY_LAUNCHER</td>
<td>The activity can be the initial activity of a task and is listed in the top-level application launcher.</td>
</tr>
<tr>
<td>CATEGORY_PREFERENCE</td>
<td>The target activity is a preference panel.</td>
</tr>
</tbody>
</table>
## Intent Constructors

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Intent()</code></td>
<td>Create an empty intent.</td>
</tr>
<tr>
<td><code>Intent(Intent o)</code></td>
<td>Copy constructor.</td>
</tr>
<tr>
<td><code>Intent(String action)</code></td>
<td>Create an intent with a given action.</td>
</tr>
<tr>
<td><code>Intent(String action, Uri uri)</code></td>
<td>Create an intent with a given action and for a given data url.</td>
</tr>
<tr>
<td><code>Intent(Context packageContext, Class&lt;? extends Activity&gt; cls)</code></td>
<td>Create an intent for a specific component.</td>
</tr>
<tr>
<td><code>Intent(String action, Uri uri, Context packageContext, Class&lt;? extends Activity&gt; cls)</code></td>
<td>Create an intent for a specific component with a specified action and data.</td>
</tr>
</tbody>
</table>

We used this previously.
Intent Info - *Data*

- How is data passed to newly created component (e.g. Activity 2)

- URI (uniform resource identifier) of data to work with / on
  - for content on device E.g. an audio file or image or contact

- MIME (Multipurpose Internet Mail Extension),
  - Initially for email types, now used to generally describe data/content type
  - E.g. image/png or audio/mpeg
**Intent - *Extras***

- *A Bundle* (key-value pairs) of additional information to be passed to component handling the Intent (e.g. Activity 2)

- Some Action will have specified extras
  - ACTION_TIMEZONE_CHANGED will have an extra with key of "time-zone"
  - Example of use of Intents extras to create alarm

```java
public void createAlarm(String message, int hour, int minutes) {
    Intent intent = new Intent(AlarmClock.ACTION_SET_ALARM)
        .putExtra(AlarmClock.EXTRA_MESSAGE, message)
        .putExtra(AlarmClock.EXTRA_HOUR, hour)
        .putExtra(AlarmClock.EXTRA_MINUTES, minutes);
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivity(intent);
    }
}
```
AndroidManifest.xml

- describes app components:
  - activities, services, broadcast receivers, content providers
- **Intents:** Also describes *intent messages each component can handle*
- **Permissions:** declares permissions requested by app
- **Libraries:** libraries application to link to
Action Bar
Can add Action bar to the `onCreate()` method of GeoQuiz to indicate what part of the app we are in.

```java
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d(TAG, "onCreate() called");
    setContentView(R.layout.activity_quiz);

    ActionBar actionBar = getActionBar();
    actionBar.setSubtitle("Bodies of Water");

    // Code to add action bar
}
```
Fragments
To illustrate fragments, we create new app **CriminalIntent**

Used to record “office crimes” e.g. leaving plates in sink, etc

Record includes:
- Title, date, photo

**List-detail app + Fragments**

**Tablet:** show list + detail

**Phone:** swipe to show next crime
Fragments

- Activities can contain multiple fragments
- Fragment’s views are inflated from a XML layout file
- Can rearrange fragments as desired on an activity
Starting Criminal Intent

- So, we will start by developing the detail view of CriminalIntent using Fragments.

Final Look of CriminalIntent

Start by Developing detail view using Fragments
Starting Criminal Intent

- Detail screen shown will be managed by a UI fragment called **CrimeFragment**
- Activity called **CrimeActivity** will host instance of fragment **CrimeFragment**
- **Hosted? CrimeActivity** provides a space/spot for **CrimeFragment** in its view hierarchy
Starting Criminal Intent

- **Crime**: holds single office crime. Has
  - **Title** e.g. “Someone stole my yogurt!”
  - **ID**: uniquely identifies crime
- **CrimeFragment** has member variable **mCrime** to hold crimes
- **CrimeActivity** has a FrameLayout with position of **CrimeFragment** defined
Hosting a UI Fragment

- To host a UI fragment, an activity must
  - Define a spot in its layout for the fragment’s view
  - Manage the lifecycle of the fragment instance
- Fragment’s lifecycle somewhat similar to activity lifecycle
  
- Has states **running**, **paused** and **stopped**
- Also has some similar activity lifecycle methods (e.g. `onPause()`, `onStop()`, etc)
- **Key difference:**
  - Activity’s lifecycle methods called by OS
  - Fragment’s lifecycle’s methods **called by hosting activity NOT Android OS!**
Hosting UI Fragment in an Activity

- 2 options. Can add fragment either
  - XML: To Activity’s layout (layout fragment), or
  - Java: In activity’s code (more complex but more flexible)
- We will add fragment to activity’s code now
- First, create a spot for the fragment’s view in CrimeActivity’s layout
Creating a UI Fragment

- Creating Fragment is similar to creating activity
  1. Define widgets in a layout file
  2. Create class and specify its view as layout above
  3. Wire up widget inflated from layout in code

- Defining layout file for CrimeFragment *(fragment_crime.xml)*

```xml
<?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>
```
Implementing Fragment Lifecycle Methods

- **CrimeFragment** presents details of a specific crime + updates
- Override CrimeFragment’s **onCreate()** function

```java
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;
    }
}
```

- **Note**: Fragment’s view not inflated in **Fragment.onCreate()**
- Fragment’s view created and configured in another fragment lifecycle method (**onCreateView**)
**Fragment LifeCycle**

- **Note:** Fragment’s view not inflated in `Fragment.onCreate()`.
- Fragment’s view created and configured in another fragment lifecycle method (onCreateView).
Wiring up the EditText Widget

```java
public class CrimeFragment extends Fragment {
    private Crime mCrime;
    private EditText mTitleField;

    ...  

    @Override
    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;
    }
}
```
Adding UI Fragment to FragmentManager

- Finally, we add fragment just created to FragmentManager.

FragmentManager
- Manages fragments
- Adds their views to activity’s view
- Handles
  - List of fragment
  - Back stack of fragment transactions

```java
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();
        }
    }
```
Examining Fragment’s Lifecycle

- FragmentManager calls fragment lifecycle methods
- `onAttach()`, `onCreate()` and `onCreateView()` called when a fragment is added to FragmentManager
- `onActivityCreated()` called after hosting activity’s `onCreate()` method is executed
- If fragment is added to already running Activity then `onAttach()`, `onCreate()`, `onCreateView()`, `onActivityCreated()`, `onStart()` and then `onResume()` called
Playing Audio File using MediaPlayer
Example taken from Android Nerd Ranch
Chapter 13

- Example creates **HelloMoon app** that uses **MediaPlayer** to play audio file
- **MediaPlayer**: Android Class for audio and video playback
- **Source**: Can play local files, or streamed over Internet
- **Supported formats**: WAV, MP3, Ogg, Vorbis, MPEG-4, 3GPP, etc
HelloMood App

- Put image `armstrong_on_moon.jpg` in `res/drawable-mdpi/` folder
- Place audio file to be played back (`one_small_step.wav`) in `res/raw` folder
- Can also copy mpeg file and play it back
- Create `strings.xml` file for app

```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>
```
HelloMoon App

- HelloMoon app will have:
  - 1 activity (**HelloMoonActivity**) that hosts **HelloMoonFragment**

- **AudioPlayer** class will be created to encapsulate **MediaPlayer**

- First set up the rest of the app by
  1. Define a layout for the fragment
  2. Create the fragment class
  3. Modify the activity and its layout to host the fragment
Defining the Layout for HelloMoonFragment

```
<TableLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
>
    <ImageView
        android:src="@drawable/armstrong_on_moon"
        android:contentDescription="@string/hellomoon_description"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:scaleType="centerInside"
        android:layout_weight="1"
    />

    <TableRow
        android:gravity="center|bottom"
        android:layout_weight="0"
    >
        <Button
            android:id="@+id/hellomoon_playButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/hellomoon_play"
        />
        <Button
            android:id="@+id/hellomoon_stopButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/hellomoon_stop"
        >
    </TableRow>
</TableLayout>
```
Creating a Layout Fragment

- Previously added Fragments to activity’s code
- Layout fragment enables fragment views to be inflated from XML file
- We will use a layout fragment instead
- Create layout fragment `activity_hello_moon.xml`

```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>
```
Set up 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;
    }
}
```
Create AudioPlayer Class to Wrap MediaPlayer

```java
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();
    }

}
Hook up Play and Stop Buttons

```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;
    }
}
```
Taking Pictures with the Smartphone’s Camera
Camera Example
Ref: Android Nerd Ranch Ch 19 & 20 (pg 299)

- Simple way: Send intent with `MediaStore.ACTION_IMAGE_CAPTURE` to Android camera app
  - Buggy on many phones
- Alternate way: Use `SurfaceView` class, display live video preview from camera
  - We will try second (alternate) way here
Overview of Camera App for CriminalIntent

- **Camera** provides hardware-level access to device’s camera(s)
- A **SurfaceView** is a view (widget) that allows content to be directly rendered unto the screen
- App will use instance of **SurfaceView** as **ViewFinder**
- Create in following order:
  - Layout for **CrimeCameraFragment**’s view
  - **CrimeCameraFragment** class
  - **CrimeCameraActivity** class
  - Use camera API in **CrimeCameraFragment**
- Finally enable instance of **CrimeCameraActivity**
Creating Layout for CrimeCameraFragment

- **Steps**
  - Layout for *CrimeCameraFragment’s view*
  - *CrimeCameraFragment* class
  - *CrimeCameraActivity* class
  - Use camera API in *CrimeCameraFragment*

```xml
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <SurfaceView
            android:id="@+id/crime_camera_surfaceView"
            android:layout_width="match_parent"
            android:layout_height="match_parent" />

        <Button
            android:id="@+id/crime_camera_takePictureButton"
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:text="@string/take" />
    </LinearLayout>
</FrameLayout>

Add “Take!” for Camera button to strings.xml

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

- **Steps**
  - Layout for *CrimeCameraFragment’s* view
  - *CrimeCameraFragment* class
  - *CrimeCameraActivity* class
  - Use camera API in *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;
    }
}
```

- Inflate camera view
- Get handle to “Take” button
- Handle “Take” Button click
- Get handle to Surface View
Creating Layout for CrimeCameraFragment

- Steps
  - Layout for CrimeCameraFragment’s view
  - CrimeCameraFragment class
  - CrimeCameraActivity class
  - Use camera API in CrimeCameraFragment

- Create new SingleFragmentActivity subclass named CrimeCameraActivity

```java
public class CrimeCameraActivity extends SingleFragmentActivity {
    @Override
    protected Fragment createFragment() {
        return new CrimeCameraFragment();
    }
}
```
Modify AndroidManifest.xml

- Steps
  - Layout for CrimeCameraFragment’s view
  - CrimeCameraFragment class
  - CrimeCameraActivity class
  - Use camera API in CrimeCameraFragment

- Add permissions and camera activity to AndroidManifest.xml

- Permissions?
  - Ask phone owner to use phone’s camera when app is installed

- uses-feature means that Google Play, app offered only to phones with camera
Use Camera API: Opening and Releasing Camera

- Camera is system-wide resource, needs to be obtained when needed and released afterwards
- Have camera handle in **CrimeCameraFragment**
- Camera Methods:
  ```java
  public static Camera open(int cameraId)
  public static Camera open()
  public final void release()
  ```
- Open Camera in onResume(), release it in onPause()
Use Camera API: Opening and Releasing Camera

- Release camera in onPause() method if it is going offscreen.
- Releasing camera if you don’t have it causes crash (e.g. running on a virtual device or another activity has it) so check first.

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

@Override
public void onPause() {
    super.onPause();

    if (mCamera != null) {
        mCamera.release();
        mCamera = null;
    }
}
```

Check that you have camera.
Use Camera API: Opening and Releasing Camera

- Camera image will be displayed on a **Surface**
- A **Surface** is a buffer of raw pixel data
- In **CrimeCameraFragment**, get **SurfView’s SurfaceHolder**

```java
@Override
@SuppressWarnings("deprecation")
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;
}
```
Camera API: Attaching Camera to Surface

- A **Surface** has a lifecycle
  - Created when **SurfaceView** appears on screen
  - Destroyed when **SurfaceView** no longer visible
- Ensure nothing is drawn to **Surface** when it no longer exists
- **SurfaceView** allows other clients to draw to its buffer
Camera API: Attaching Camera to Surface

- Would like **Camera** to attach to **SurfaceHolder** when **Surface** is created, detach when it is destroyed
- **SurfaceHolder.Callback** is another interface of **Surface**
Camera API: Using Surface

```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) {
        // 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;
        }
    }
});

return v;
```

- **SurfaceHolder.Callback** has 3 methods:
  1. **SurfaceCreated**: Called when view hierarchy containing **SurfaceView** is created
  2. **SurfaceChanged**: Called when surface is first displayed
  3. **SurfaceDestroyed**: Called when **SurfaceView** is destroyed
Adding Camera Start Button in CriminalIntent

- Would like to add Button in CriminalIntent to launch camera
Adding Camera Start Button in CriminalIntent

- Add 3 linearlayouts, rearrange widgets
Update CrimeFragment

1. Add member variable for image button
2. Get reference to it
3. Set onClickListener that starts CrimeCameraActivity

```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;
}
```
Final Result

- Final Result after running App and clicking Camera Button
Camera II: Taking Pictures and Handling Images
Camera II: Taking Pictures and Handling Images (Ref: Chapter 20 Android Nerd Ranch)

- **Goal:** Write program to:
  - Capture image from camera’s preview
  - Save image as JPEG as part of a **Crime**
  - Display image in **CrimeFragment’s view**
  - Offer user option to view larger version of image in **DialogFragment**

**IMPORTANT:** Read Android Nerd Ranch Chapter 20!!!
References

- Busy Coder’s guide to Android version 4.4
- CS 65/165 slides, Dartmouth College, Spring 2014
- CS 371M slides, U of Texas Austin, Spring 2014