CS 528 Mobile and Ubiquitous Computing
Lecture 4: AdapterViews, Intents, Fragments
Audio/Video, Camera

Emmanuel Agu
Layouts with More Interactivity & Data-Dependent
Container Control Classes

- LinearLayout, RelativeLayout, TableLayout, GridLayout useful for positioning UI elements
  - the layouts themselves are not interactive although the child Views may be

- Other available layouts have more interactivity between the user and the child Views
  - ListView, GridView, GalleryView
  - Tabs with TabHost, TabControl
  - ScrollView, HorizontalScrollView
Data Driven Containers

- Containers that display repetitive child View controls in a given way

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

- Containers that display repetitive child View controls in a given way

- GridView
  - specified number of rows and columns
Data Driven Containers

- Containers that display repetitive child View controls in a given way

- GalleryView
  - horizontal scrolling list, typically images
ListAdapter

- ListView, GridView, andGalleryView 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
  - **CursorAdapter** used to read from database
  - Use **ArrayAdapter** to read from resource, typically an XML file
Adapters

- When using an Adapter a layout is defined for each child element (View)
- The adapter
  - Creates Views using layout for each element in data source
  - Fills the containing View (List, Grid, Gallery) with the created Views
- Child Views can be as simple as a TextView or more complex layouts / controls
  - simple views can be declared in android.R.layout
Using ArrayAdapter

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

```
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/apk/res/android"
    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 AdapterArray

```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)findViewByPosition(R.id.selection);
  }

  @Override
  public void onListItemClick(ListView parent, View v, int position,
                               long id) {
    selection.setText(items[position]);
  }
}
```
Selection Events

- ListView, GridView, GalleryView
- Typically user can select one item of data
- Implement the OnItemClickListener class, set it as the listener
- This approach is used a lot:
  - create a class that implements some kind of listener
  - register it with a control
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 to show Answer

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

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

- First create layout for screen 2

```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"
    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.beginnerbranch.android.geoquiz.CheatActivity"
    android:label="@string/app_name"/>
</application>
```

Format using the layout you just created
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:

  ```java
  public Intent(Context packageContext, Class<?> 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 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
- Examples of Google applications:
**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 the component that receives the intent
  - 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

```
<?xml version="1.0"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.commonsware.android.skeleton"
  android:versionCode="1"
  android:versionName="1.0">

  <application>
    <activity
      android:name="Now"
      android:label="Now">
      <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.LAUNCHER"/>
      </intent-filter>
    </activity>
  </application>

</manifest>
```
### 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 - Data**

- URI (uniform resource identifier) of data to work with / on
  - for content on device a content provider and identifying information, for example an audio file or image or contact

- MIME (Multipurpose Internet Mail Extension, now internet media type) initially for email types, but extended to describe type information in general about data / content
  - image/png or audio/mpeg
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>Public Constructors</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Intent ()</strong></td>
</tr>
<tr>
<td>Create an empty intent.</td>
</tr>
<tr>
<td><strong>Intent (Intent o)</strong></td>
</tr>
<tr>
<td>Copy constructor.</td>
</tr>
<tr>
<td><strong>Intent (String action)</strong></td>
</tr>
<tr>
<td>Create an intent with a given action.</td>
</tr>
<tr>
<td><strong>Intent (String action, Uri uri)</strong></td>
</tr>
<tr>
<td>Create an intent with a given action and for a given data url.</td>
</tr>
<tr>
<td><strong>Intent (Context packageContext, Class&lt;? cls)</strong></td>
</tr>
<tr>
<td>Create an intent for a specific component.</td>
</tr>
<tr>
<td><strong>Intent (String action, Uri uri, Context packageContext, Class&lt;? cls)</strong></td>
</tr>
<tr>
<td>Create an intent for a specific component with a specified action and data.</td>
</tr>
</tbody>
</table>
Intent - *Extras*

- A *Bundle* (like a map / dictionary, key-value pairs) of additional information to be given to the component handling the Intent

- Some Action will have specified extras
  - `ACTION_TIMEZONE_CHANGED` will have an extra with key of "time-zone"
  - Intent method has put methods or put a whole Bundle
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
Recall: AndroidManifest.xml - Launcher Intent

```xml
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="scott.examples.lifeCycleTest"
    android:versionCode="1"
    android:versionName="1.0">

    <uses-sdk android:minSdkVersion="10" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name">
        <activity
            android:name=".LifeCycleTestActivity"
            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>
```

Declare this as Activity to start when app is started.


```java
/** Called when the user clicks the Send button */
public void sendMessage(View view) {
    Intent intent = new Intent(this, DisplayMessageActivity.class);
    EditText editText = (EditText) findViewById(R.id.edit_message);
    String message = editText.getText().toString();
    intent.putExtra(EXTRA_MESSAGE, message);
    startActivity(intent);
}

public final static String EXTRA_MESSAGE = "scottm.utexas.myfirstapp.MESSAGE";
```
Action Bar
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 = getSupportActionBar(); // Code to add action bar
    actionBar.setSubtitle("Bodies of Water");
}
```
Fragments
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 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.
Starting Criminal Intent

- Detail screen shown will be managed by a UI fragment called **CrimeFragment**
- An instance of **CrimeFragment** will be hosted by an activity called **CrimeActivity**
- **Hosted? CrimeActivity** provides a 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:**
  - Fragment’s lifecycle’s methods **called by hosting activity** NOT Android OS!
Hosting UI Fragment in an Activity

- 2 options. Can add fragment either
  - To Activity’s layout (layout fragment), or
  - In the 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

```xml
<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 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)
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 = getFragmentManager();
        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
- 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_width="match_parent"
- android:layout_height="wrap_content"
- 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"
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>
```
Lifecycle of a Layout Fragment

- Running
  - onPause()

- Paused
  - onResume()
  - (activity/fragment returns to foreground)

- Stopped
  - onStart()
  - (activity/fragment becomes visible again)
  - onStop()
  - onDestroyView()

- Created
  - onActivityCreated()
  - onAttach(), onCreate(...), oncreateView()
  - (all called in setContentView() for layout fragments)
  - Launch

- (activity shutdown)
  - onDestroy(), onDetach()
  - Death
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

- 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 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_weight="1"
            android:layout_width="0dp"
            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;
    }
}
```
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 GooglePlay, 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

... 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;
    }
}
```

Can also modify to see if the phone has a camera. See Android Nerd Ranch
Final Result

- Final Result after running App and clicking Camera Button

Can also hide status bar and action bar. See Android Nerd Ranch for details.
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
Update CrimeCameraFragment’s Layout

- Update **CrimeCameraFragment** to include progress indicator
- Progress indicator gives user feedback on picture taking process
- In **fragment_crime_camera.xml** add **FrameLayout** and **ProgressBar** widgets
Add FrameLayout and ProgressBar

- Make FrameLayout (and ProgressBar) invisible at first
- Become visible after user presses the Take! Button
- FrameLayout stacks child views in the order they are defined
- Consequently, child FrameLayout and ProgressBar completely covers sibling LinearLayout
- **Note:** User cannot interact with screen or press Take! Again

Becomes visible When Take! Button Is clicked
Wire up the FrameLayout containing ProgressBar

- After declaring **FrameLayout** and **ProgressBar**...
- In **CrimeCameraFragment.java**, get reference to **FrameLayout**
- Set **FrameLayout** to invisible

```java
public class CrimeCameraFragment extends Fragment {
    ...
    private View mProgressContainer;

    @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;
    }
    ...
}
```
Taking a Picture

- To take a picture, the **Camera** method **takePicture** is used.

```java
public final void takePicture(Camera.ShutterCallbackCallback shutter,
                              Camera.PictureCallbackCallback raw,
                              Camera.PictureCallbackCallback jpeg)
```

- Can implement these interfaces, pass them to **takePicture**()
- Pass null for any **takePicture** parameters not interested in.

---

- Occurs when camera captures picture but before image is processed.
- Occurs when raw image is available.
- Occurs when JPEG version of image is available.
Camera takePicture Interfaces

- In our app, we will implement shutter and JPEG callbacks

```java
public static interface Camera.ShutterCallback {
    public abstract void onShutter();
}

public static interface Camera.PictureCallback {
    public abstract void onPictureTaken (byte[] data, Camera camera);
}
```

- Interactions between objects are shown
Camera takePicture Interfaces

- In CrimeCameraFragment need to implement:
  1. Camera.ShutterCallback that makes ProgressBar visible
  2. Camera.PictureCallback implementation that handles naming and saving the JPEG file.
Add Shutter and Picture Callbacks in CrimeCameraFragment.java

...
Call takePicture from Take! Button Listener

```java
@Override
@SuppressWarnings("deprecation")
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;
}
```

Pass null for unimplemented callback
Setting the Picture Size

- Camera needs to know picture size to create
- Approach:
  - Get list of acceptable picture sizes by calling `getSupportedPictureSize()` method of `Camera.Parameters`
  - In `surfaceChanged()`, use `getBestSupportedSize()` to find picture size that will work with our `Surface`
  - Finally, set camera’s picture size

```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);
}
```
Passing Data Back to CrimeFragment

- **CrimeCameraFragment** can now take picture and save it
- Need to integrate photo with rest of **CriminalIntent app**
- Pass photo filename back to **CrimeFragment** and **CrimeCameraFragment**
Starting CrimeCameraActivity for a Result

- **CrimeFragment** currently just starts **CrimeCameraActivity**
- Need to modify **CrimeCameraActivity** to start for a result

```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);
            }
        });
    }
    ...
}
```
Camera taking Picture: More steps

- More steps are described in example in Android Nerd Ranch including:
  - Retrieving filename in CrimeFragment
  - Creating Photo object and setting photo properties
  - Adding scaled photo to an ImageView
  - Displaying larger image in a DialogFragment
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