CSC 472 / 372 Mobile Application Development for Android

Prof. Xiaoping Jia
School of Computing, CDM
DePaul University
xjia@cdm.depaul.edu
@DePaulSWEng

Outline

- Handling different screen sizes
- Fragments
- Fragment lifecycle
- Using the *Master/Detail Flow* template
- Coordination among fragments

Support Different Screen Sizes

- Android runs on devices with a wide range of different screen sizes.
- Optimize UI design for tablets and handsets
- Ensure layout can be appropriately sized to fit the screen
- Provide appropriate UI layout according to screen configuration
- Ensure the appropriate layout is applied to the correct screen

Single and Multi-Pane Layouts

- Smaller screens are generally suited for a single vertical pane of content
  - To present a hierarchy of information (category-list-details), screens generally map one-to-one to levels
- Larger screens have much more space
  - Single vertical panes generally works poorly
  - Capable of presenting multiple panes, with multiple levels of information on a single screen

Poor Design: Single Pane Layouts on Large Screens

Better Design: Multi-Pane Layouts in Landscape

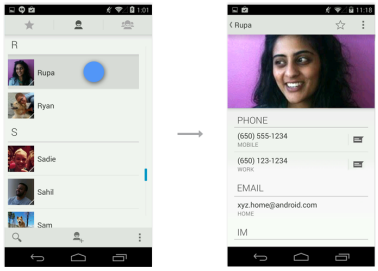
UI Pattern: Multi-Pane Layout

- An app should accommodate both small and large screen sizes
- On smaller screens
  - Each screen contains a single pane
  - Each screen maps to a single level of information
- On larger screens
  - Each screen can be divided into several vertical panes
  - Each pane contains a level of information
  - Each screen maps to multiple levels of information

But, we don't want to rewrite an app for every different screen size!

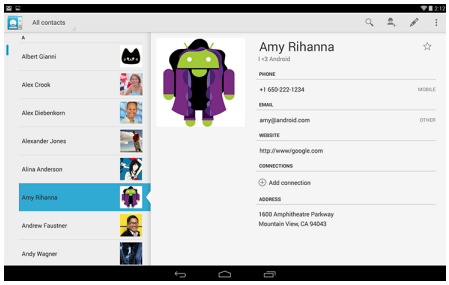
DePaul University 8

UI Pattern: Small Screen Layout



DePaul University 9

UI Pattern: Large Screen Layout

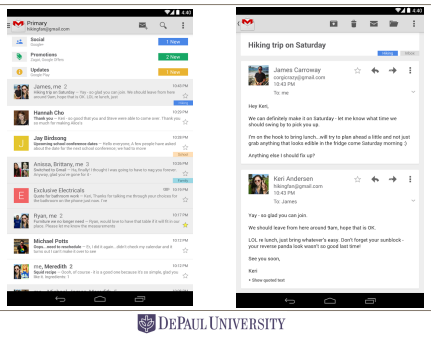


DePaul University 10

Example: Netflix App – Handset and Tablet Layout

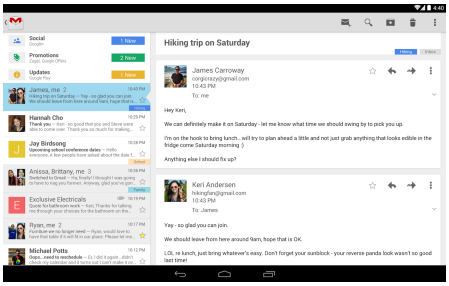
DePaul University 11

Example: Gmail App – Handset Layout



DePaul University 12

Example Gmail App – Tablet Layout



DePaul University 13

Support Multiple Screen Sizes

- Use *alternative layout resources* for different screen sizes
  - Use *configuration qualifiers* to classify devices into different categories
  - Provide alternative resources for specific categories
  - All layout resources are packed into the application package
  - Android system selects the best match of resources at run-time

Support Multiple Screen Sizes

- Use *Fragment API* (complements *Activities*)
  - Decouple app logic from screen layouts
  - Support more dynamic and flexible UI designs on large screens
  - Separate behavioral components for multi-pane layout
  - Maximize code reuse, and avoid redundant code

Providing Alternative Layout Resources

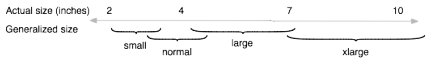
Configuration Qualifiers

- Developers may provide alternative resources for different configurations
- A *configuration qualifier* is a string that can be appended to a resource directory.
  - Specify the configuration for which the resources are design
  - Control how the system selects alternative resources
- Some configuration qualifiers
  - Screen density: `ldpi mdpi hdpi xhdpi` *etc.*,
  - Screen size: `small normal large xlarge`
  - Screen orientation: `land port`

Screen "Size Bucket" Qualifiers

- `xlarge` – at least 960dp x 720dp.
  - Can comfortably fit multi-pane UI, even in portrait mode
- `large` – at least 640dp x 480dp.
  - Can fit multi-pane UI in landscape mode.
- `normal` – at least 470dp x 320dp.
  - Generally fits single pane of information
- `small` – at least 426dp x 320dp.
  - Android does not support screens smaller

Notice that the sizes are specified in **dp**, not in px



Screen Orientations and Rotation

- Configuration qualifiers for screen orientations
  - `land` – landscape orientation, wide aspect ratio
  - `port` – portrait orientation, tall aspect ratio
- Screen rotation is a *run-time configuration change*
  - Default behavior – Android restarts the running *Activity*
    - `onDestroy()` is called, followed by `onCreate()`
    - Allows reloading of alternative resources for the *Activity* to adapt to the new configuration
    - Need to properly handle saving and restoring the instance state
  - You may retain stateful objects through configuration change
  - You may handle configuration change yourself

Use Configuration Qualifiers

- For any resource directory:
  res/*resourceName*
  you may create a new resource directory for alternative resources by appending a configuration qualifier
  res/*resourceName-qualifier*
- The configuration-specific resource files must be named exactly the same as the default resource file
- You may use multiple qualifiers, separated by '-'
  - The order of qualifiers implies the order of precedence

DePaul University 21

Provide Layout Resources for Different Screen Sizes

- **res/layout/my_layout.xml**
  - layout for normal screen size (the "default")
- **res/layout-large/my_layout.xml**
  - layout for large screen size
- **res/layout-xlarge/my_layout.xml**
  - layout for extra-large screen size
- **res/layout-xlarge-land/my_layout.xml**
  - layout for extra-large in landscape orientation

The resource file names are identical in all directories

Directory with two qualifiers

DePaul University 26

Issues with "Size Bucket" Selector

- The limited set of screen-size buckets is inadequate in adapting to the increasing variety of Android-device shapes and sizes
- The boundaries are artificial. May not correspond to actual devices or application needs.
- The "large" screen size has been challenging for developers
  - Encompass a wide range of devices
  - May need different design in landscape vs. portrait mode

DePaul University 27

Numeric Size Selectors

- Smallest-width: (in "dp")
  - The smallest width available for application layout, i.e. the smallest width in any rotation of the display
  - *Does not change* when the screen switches orientation
- Available width: (in "dp")
  - The current width available for application layout
  - *May change* when the screen switches orientation
- Available height: (in "dp"):
  - The current height available for application layout
  - *May change* when the screen switches orientation.

Most useful. Can provide a continuous range of sizes.

DePaul University 29

Using Numeric Size Selectors

- **res/layout/main_activity.xml**
  - For phones
- **res/layout-sw600dp/main_activity.xml**
  - For 7" tablets or larger
- **res/layout-sw720dp/main_activity.xml**
  - For 10" tablets or larger
- **res/layout-w600dp/main_activity.xml**
  - Multi-pane when enough width
- **res/layout-sw600dp-port/main_activity.xml**
  - Tablets when in portrait

DePaul University 30

An Issue: Proliferation of Resource Files

- Suppose you have the following resources for **main**
  - **res/layout/main.xml** single-pane layout
  - **res/layout-large/main.xml** multi-pane layout
  - **res/layout-sw600dp/main.xml** multi-pane layout
- The multi-pane layout files are the same

Can we avoid the duplications?

DePaul University 32

Layout Aliases

- Rearrange the resources
  - **res/layout/main.xml** single-pane layout
  - **res/layout/main-twopanes.xml** multi-pane layout
- Add two (small) files that define *aliases* to layouts
  - **res/values-large/layout.xml**

```
<resources>
  <item name="main" type="layout">@layout/main_twopanes</item>
</resources>
```

  - **res/values-sw600dp/layout.xml**

```
<resources>
  <item name="main" type="layout">@layout/main_twopanes</item>
</resources>
```


Fragments

Fragments

- A *Fragment* represents the behavior associated with *a portion* of user interface in an activity
  - A *Controller*, in MVC,
  - Similar purpose and form to *Activity*, but at a finer granularity than *Activity*
- A modular section of an activity, i.e., a "fragment"
  - Has its own layout
  - Has its own lifecycle
  - Can be added or removed independently, while the activity is running

Hot swap

Fragments

- A *Fragment* must be embedded in an *Activity*
- Can be added to the activity's layout file using the `<fragment>` element
  - Not removable if added in layout resource
- Can be added and removed dynamically at run-time to a *View Group*
- Can also be UI-less, (but added to an *Activity*)
  - Not part of the *Activity*'s layout
  - An invisible worker for the *Activity*

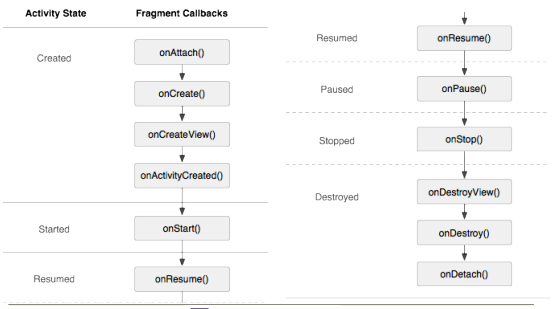
Fragment Lifecycle States

- *Resumed,* or *Running,* or *Active*
  - The fragment is visible in the running activity.
- *Paused*
  - Another activity is in the foreground and has focus, but the host activity is still visible
- *Stopped*
  - The fragment is not visible.
    - Either the host activity has been stopped or the fragment has been removed from the activity
  - A stopped fragment is still alive (all state is retained).
  - Will be killed if the host activity is killed.

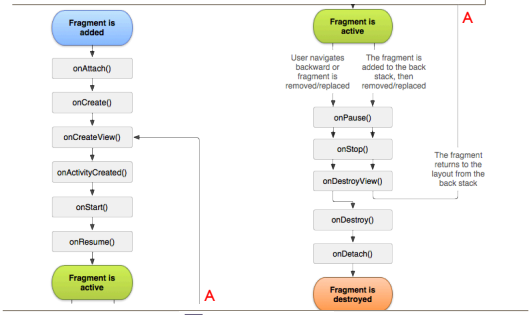
Fragment Lifecycle

- The fragment's lifecycle is directly affected by the host activity's lifecycle.
- When the activity is *paused*, so are all the fragments hosted in the activity.
- When the activity is *destroyed*, so are all the fragments hosted in the activity.
- While an activity is *running*, each fragment can be added and removed independently.

Fragment Lifecycle States



Fragment Lifecycle – While the Host Activity is Running



Fragment Callback Methods

- Similar to the callbacks of *Activity*.
  - To facilitate conversion from *Activities* to *Fragments*.
  - Code can be moved from *Activity* callback methods to the respective callback methods of *Fragment*.
- Usually the following three methods are needed for every *Fragment* class.
  - `onCreate()`
  - `onCreateView()`
  - `onPause()`

Fragment Callback Methods

- `onCreate()`
  - Called when the fragment is created.
  - Initialize essential components of the fragment.
- `onCreateView()`
  - Called before the fragment become visible.
  - Return a *View* object for the UI of the fragment
  - Return null if the fragment has no UI.
- `onPause()`
  - Called when the user is leaving the fragment.
  - Save any changes that should be persisted.

Fragment Callback Methods

- `onAttach()`
  - Called when the fragment has been associated with the activity
- `onDestroyView()`
  - Called when the view hierarchy associated with the fragment is being removed
- `onDetach()`
  - Called when the fragment is being disassociated from the activity

Fragment Back Stack

- Fragments can also be added to a *back stack* managed by its hosting activity
  - Not the same back stack that contains activities
- Each entry is a record of the fragment transaction.
- Allows the user to reverse a fragment transaction (navigate backwards), by pressing the *Back* button.

Multi-Pane Design Using Fragments

Multi-Pane Design Techniques

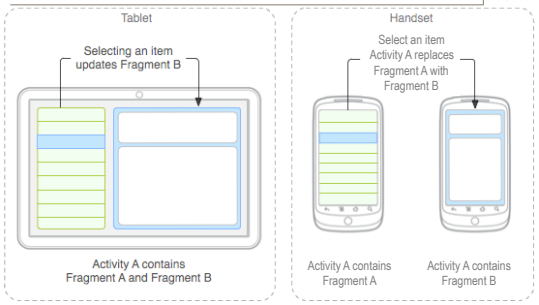
DePaul University 47

Multi-Pane Design Techniques Approach A

- **Multiple Fragments, one Activity**
- **Implement behavior in *Fragments***
- Use one activity regardless of the screen size
- Decide at runtime whether to combine fragments in the layout
  - For tablets:
    - Combine fragments, to create a multiple-pane design
  - For handsets:
    - Swap fragments, to create a single-pane design

DePaul University 48

Multi-Pane Design Technique Approach A



DePaul University 49

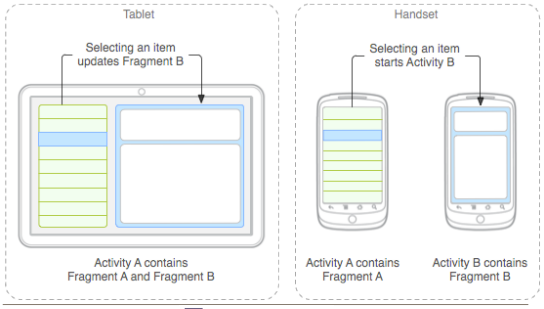
Multi-Pane Design Techniques Approach B

- **Multiple fragments, multiple activities**
- **Implement behavior in *Fragments***
- Use multiple activities for handsets
  - For tablets:
    - Use a multi-pane layout
    - Place multiple fragments in one activity
  - For handsets:
    - Use a single-pane layout
    - Each activity hosts one fragment

The *Master/Detail Flow* template uses this option

DePaul University 51

Multi-Pane Design Technique Approach B



DePaul University 52

Wine List App – With Fragments

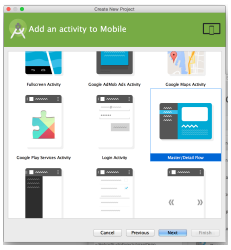
Using the Master/Detail Flow Template

Wine List App with Fragments

- Another variation of the *Wine List* app
- Support tablet screen size
  - A single pane layout for handsets
  - A two-pane layout for tablets
  - Using *Fragments*
  - Using the Master/Detail template in Android Studio
- Two versions
  - Simple and minimalistic with *Fragments*
  - Customized *Fragments* with ratings

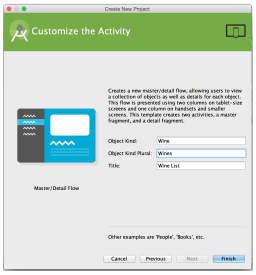
Using Master/Detail Template

- A template to create *Master/Detail* app with *Fragments*
  - Two levels
  - Single pane design for handsets
  - Double pane design for tablets
  - Simple, minimum UI
  - Contains dummy data
- First version of *Wine List* app
  - Replace dummy data with real data
  - No UI customization



Create a Master/Detail App

- New Project
- *Activity* dialog
  - choose "Master/Detail Flow" template
- *Choose options* dialog
  - **Object Kind**: *Wine*
  - **Object Kind Plural**: *Wines*
  - **Title**: *Wine List*



Understand the Components of the Master/Detail Template

- The generated Java classes:

`WineListFragment` – The *Fragment* class for the master list pane

`WineDetailFragment` – The *Fragment* class for the detail view pane

`WineListActivity` – The main *Activity* class
  - Hosts the list fragment in single-pane layout
  - Hosts both the list and detail fragments in two-pane layout

`WineDetailActivity` – The detail *Activity* class
  - Hosts the detail fragment in single-pane layout
  - Unused in two-pane layout

`dummy.DummyContent` – The dummy data.

Understand the Components of the Master/Detail Template

- The generated layout resource files:

layout/activity_wine_list.xml (2 configurations)

The main *Activity* loads a resource in this name.
  - *(default)* The single-pane layout for the main *Activity*.
  - **(sw600dp)** The two-pane layout for the main *Activity*.

layout/activity_wine_detail.xml
  - The single-pane layout for the detail *Activity*

layout/fragment_wine_detail.xml
  - The layout for the detail *Fragment*

Implement a Fragment Class

- Extend the *Fragment* class, and override lifecycle methods
  - The default (no-arg) constructor is **mandatory**
- Must use the `onCreateView()` callback to define the layout.
  - The only callback needed to get a fragment running.
- Adding a *Fragment* to a layout resource

```
<fragment android:name="fully-qualified-class-name"
          android:id="@+id/fragment-name"
          … other attributes … />
```

DePaul University 59

The Fragment Manager

- Manage the fragments in an *Activity*.
- From the host *Activity*

```
getFragmentManager()
```

- Things you can do with *Fragment Manager*
  - `findFragmentById()`
    - Get fragments in the activity
  - `popBackStack()`
    - Pop fragments off the back stack (simulating a *Back* command by the user)

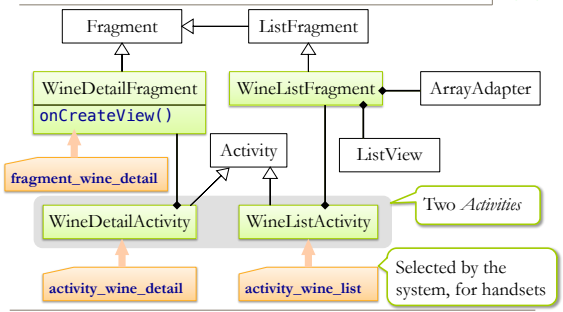
DePaul University 60

The Frame Layout

- One of the simplest layout
  - Usually contains a single child
  - If it has more than one children, they are overlaid on top of one another
- Designed to block out an area on the screen to display a single item
- Commonly used as a container to hold the UI of a *Fragment*

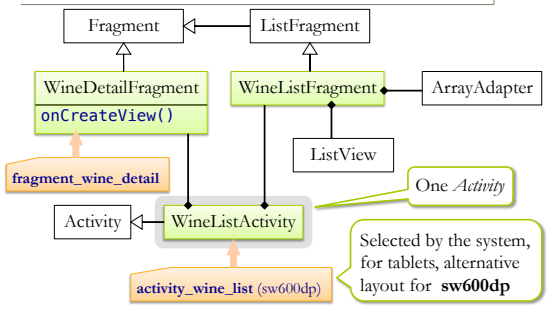
DePaul University 61

Master/Detail Architecture – Single Pane Mode



DePaul University 64

Master/Detail Architecture – Two-Pane Mode



DePaul University 67

The Single Pane Layout layout/activity_wine_list.xml

The main *Activity* contains a *Fragment*. Part of the layout. Fixed.

The fully qualified name of the *Fragment*

```
<fragment xmlns:android="http://…"
  xmlns:tools="http://schemas.android.com/tools"
  android:id="@+id/wine_list"
  android:name="edu.depaul.csc472.winelist.WineListFragment"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:layout_marginLeft="16dp"
  android:layout_marginRight="16dp"
  tools:context=".WineListActivity"
  tools:layout="@android:layout/list_content" />
```

DePaul University 70

The Two Pane Layout
layout-sw600dp/activity_wine_list.xml

```
<LinearLayout … >
  <fragment android:id="@+id/wine_list"
    android:name="edu.depaul.csc472.winelist.WineListFragment"
    android:layout_width="0dp"
    android:layout_height="match_parent"
    android:layout_weight="1"
    tools:layout="@android:layout/list_content" />
  <FrameLayout
    android:id="@+id/wine_detail_container"
    android:layout_width="0dp"
    android:layout_height="match_parent"
    android:layout_weight="3" />
</LinearLayout>
```

Same as single pane.

The container to hold the detail *Fragment*. A placeholder only.

DePaul University 73

The Main Activity
WineListActivity.java

```
public class WineListActivity extends Activity
        implements … {
  private boolean mTwoPane;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_wine_list);
    if (findViewById(R.id.wine_detail_container) != null) {
      mTwoPane = true;
      ((WineListFragment) getFragmentManager()
          .findFragmentById(R.id.wine_list))
          .setActivateOnItemClick(true);
    }
  }
  …
}
```

A flag for two-pane mode

The layout contains a *Fragment*

If the layout also contains the *Frame Layout*, it must be the two-pane layout

Code to activate the detail *Fragment*

DePaul University 77

List Fragment

- A subclass of *Fragment*
- Displays a list of items that are managed by an adapter
- Similar to *List Activity*
- The default implementation returns a *List View* from `onCreateView()`

DePaul University 78

The List Fragment

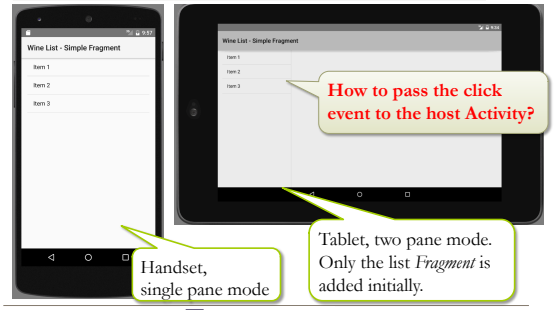
```
public class WineListFragment extends ListFragment {
  public WineListFragment() {}
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setListAdapter(
      new ArrayAdapter<DummyContent.DummyItem>(
            getActivity(),
            android.R.layout.simple_list_item_activated_1,
            android.R.id.text1,
            DummyContent.ITEMS));
  }
  …
}
```

`onCreateView()` not needed

Similar to *List Activity*. Use dummy data

DePaul University 81

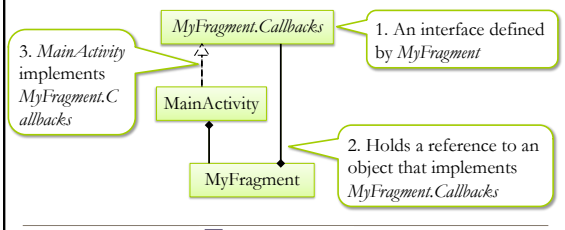
The Skeletal App



DePaul University 84

Communicate Between a Fragment and Its Hosting Activity

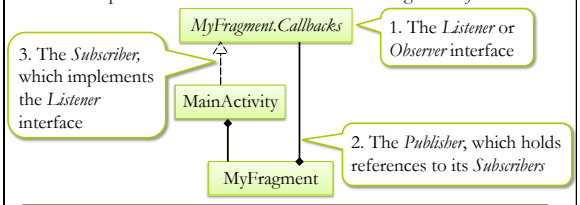
- Define a callback (listener) interface in the *Fragment*
- Require that the host *Activity* implement it



DePaul University 88

The Publisher-Subscriber Design Pattern

- A.k.a. Observer, Listener
- A common design pattern to facilitate communication among objects through event notifications.
- Decouple the *Publisher* from the *Subscribers* using an *interface*



The List Fragment The Callback Interface

```
public class WineListFragment extends ListFragment {
  public interface Callbacks {
    public void onItemSelected(String id);
  }

  private Callbacks mCallbacks = sDummyCallbacks;

  private static Callbacks sDummyCallbacks =
    new Callbacks() {
      @Override
      public void onItemSelected(String id) { }
    };
  …
}
```

The interface

The callback listener

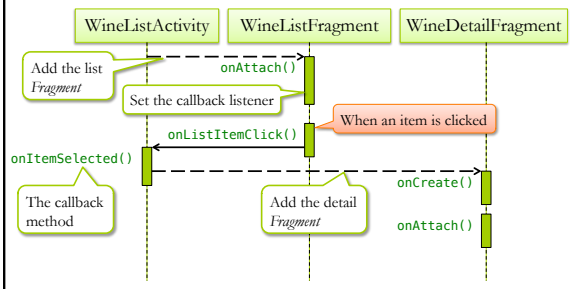
A dummy listener

The Main Activity Implement the Callbacks Interface

```
public class WineListActivity extends Activity
        implements WineListFragment.Callbacks {
  …
  @Override
  public void onItemSelected(String id) {
    …
  }
  …
}
```

Will show the implementation later

Communicate Between a Fragment and Its Hosting Activity



The List Fragment onAttach()

```
public class WineListFragment extends ListFragment {
  private Callbacks mCallbacks = sDummyCallbacks;
  …
  @Override
  public void onAttach(Activity activity) {
    super.onAttach(activity);
    if (!(activity instanceof Callbacks)) {
      throw new IllegalStateException(
          "Activity must implement fragment's callbacks.");
    }
    mCallbacks = (Callbacks) activity;
  }
  …
}
```

API changes in 6.0. Activity → Context

The host *Activity* becomes the callback listener.

The List Fragment onListItemClick()

```
public class WineListFragment extends ListFragment {
  private Callbacks mCallbacks = sDummyCallbacks;
  …
  @Override
  public void onListItemClick(ListView listView,
              View view, int position, long id) {
    super.onListItemClick(listView, view, position, id);
    mCallbacks.onItemSelected(
      DummyContent.ITEMS.get(position).id);
  }
  …
}
```

Pass the event to the *Activity*. Pass the id of the selected item.

The Main Activity onItemSelected()

```
@Override public void onItemSelected(String id) {
  if (mTwoPane) {
    Bundle arguments = new Bundle();
    arguments.putString(WineDetailFragment.ARG_ITEM_ID, id);
    WineDetailFragment fragment = new WineDetailFragment();
    fragment.setArguments(arguments);
    getFragmentManager().beginTransaction()
      .replace(R.id.wine_detail_container, fragment).commit();
  } else {
    Intent detailIntent = new Intent(this,
                              WineDetailActivity.class);
    detailIntent.putExtra(WineDetailFragment.ARG_ITEM_ID, id);
    startActivity(detailIntent);
  }
}
```

Replace the detail *Fragment* in two-pane mode

Activate the detail *Activity* in single pane mode

The Detail Fragment

```
public class WineDetailFragment extends Fragment {
  public static final String ARG_ITEM_ID = "item_id";
  private DummyContent.DummyItem mItem;
  public WineDetailFragment() {}
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    if (getArguments().containsKey(ARG_ITEM_ID)) {
      mItem = DummyContent.ITEM_MAP.get(getArguments()
                .getString(ARG_ITEM_ID));
    }
  }
  …
}
```

Retrieve the selected item using the id received as an argument.

The Detail Fragment

```
public class WineDetailFragment extends Fragment {
  …
  @Override
  public View onCreateView(LayoutInflater inflater,
          ViewGroup container, Bundle savedInstanceState) {
    View rootView = inflater.inflate(
        R.layout.fragment_wine_detail, container, false);
    if (mItem != null) {
      ((TextView) rootView.findViewById(R.id.wine_detail))
        .setText(mItem.content);
    }
    return rootView;
  }
  …
```

1. Inflate the layout
2. Set the content to the selected item.

The Layout for Detail Fragment layout/fragment_wine_detail.xml

```
<TextView xmlns:android="http://…"
  xmlns:tools="http://schemas.android.com/tools"
  android:id="@+id/wine_detail"
  style="?android:attr/textAppearanceLarge"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:padding="16dp"
  android:textIsSelectable="true"
  tools:context=".WineDetailFragment" />
```


The Detail Activity – onCreate()

```
public class WineDetailActivity extends Activity {
  @Override protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_wine_detail);
    getActionBar().setDisplayHomeAsUpEnabled(true);
    if (savedInstanceState == null) {
      Bundle arguments = new Bundle();
      arguments.putString(WineDetailFragment.ARG_ITEM_ID,
      getIntent().getStringExtra(WineDetailFragment.ARG_ITEM_ID));
      WineDetailFragment fragment = new WineDetailFragment();
      fragment.setArguments(arguments);
      getFragmentManager().beginTransaction()
            .add(R.id.wine_detail_container, fragment).commit();
    }
  }
…
```

1. Create the detail *Fragment*
2. Add it to the *Frame Layout*

The Detail Activity Layout layout/activity_wine_detail.xml

```
<FrameLayout xmlns:android="http://…"
  xmlns:tools="http://schemas.android.com/tools"
  android:id="@+id/wine_detail_container"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context=".WineDetailActivity"
  tools:ignore="MergeRootFrame" />
```

The container to hold the detail *Fragment*. Only used in the single pane mode

The Detail Activity – Handling the Home Button

```
public class WineDetailActivity extends Activity {
  @Override protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_wine_detail);
    getActionBar().setDisplayHomeAsUpEnabled(true);
    …
  }
  @Override
  public boolean onOptionsItemSelected(MenuItem item) {
    int id = item.getItemId();
    if (id == android.R.id.home) {
      navigateUpTo(new Intent(this, WineListActivity.class));
      return true;
    }
    return super.onOptionsItemSelected(item);
  }
}
```


Providing Data

- Replace the dummy data with real data
  - Remove the dummy package and it contents
  - Remove/change all references to dummy data
    - Only the *Fragments* handle data
- Expected form of data
  - A *List* of data items, of any type
    - Each item must have a unique id
  - A *Map* from id to data item

The Wine List Data

- The *Wine* class
  - Same as before
  - Move to new package: *wine*
  - The *name* is used as the unique id
- The *Wine List* class
  - A **static** array of *Wine* objects, same as before
  - A **static** *List* of *Wine* objects
  - A **static** *Map* from *name* (the id) to *Wine* objects

The Wine List Class

```
public class WineList {
  public static List<Wine> WINES = new ArrayList<Wine>();
  public static Map<String, Wine> WINE_MAP =
                        new HashMap<String, Wine>();
  private static void addItem(Wine wine) {
    WINES.add(wine);
    WINE_MAP.put(wine.name, wine);
  }
  public static final Wine[] WINES_ARRAY = {
    new Wine("Barbera", Wine.Type.Red, … ), …
    new Wine("Zinfandel", Wine.Type.Red, … ),
  };
  static {
    for (Wine wine : WINES_ARRAY) addItem(wine);
  }
}
```


The List Fragment onCreate()

```
public class WineListFragment extends ListFragment  {
  public WineListFragment() {}
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setListAdapter(
      new ArrayAdapter< DummyContent.DummyItem >(
            getActivity(),
            android.R.layout.simple_list_item_activated_1,
            android.R.id.text1,
            DummyContent.ITEMS ));
  }
  …
}
```


The List Fragment onCreate()

```
public class WineListFragment extends ListFragment  {
  public WineListFragment() {}
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setListAdapter(
      new ArrayAdapter< Wine >(
            getActivity(),
            android.R.layout.simple_list_item_activated_1,
            android.R.id.text1,
            WineList.WINES ));
  }
  …
}
```

The List Fragment onListItemClick()

```
public class WineListFragment extends ListFragment {
  private Callbacks mCallbacks = sDummyCallbacks;
  …
  @Override
  public void onListItemClick(ListView listView,
              View view, int position, long id) {
    super.onListItemClick(listView, view, position, id);
    mCallbacks.onItemSelected(
      DummyContent.ITEMS.get(position).id );
  }
  …
}
```

DePaul University 116

The List Fragment onListItemClick()

```
public class WineListFragment extends ListFragment {
  private Callbacks mCallbacks = sDummyCallbacks;
  …
  @Override
  public void onListItemClick(ListView listView,
              View view, int position, long id) {
    super.onListItemClick(listView, view, position, id);
    mCallbacks.onItemSelected(
      WineList.WINES.get(position).getName() );
  }
  …
}
```

DePaul University 117

The Detail Fragment onCreate()

```
public class WineDetailFragment extends Fragment {
  public static final String ARG_ITEM_ID = "item_id";
  private DummyContent.DummyItem mItem;
  public WineDetailFragment() {}
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    if (getArguments().containsKey(ARG_ITEM_ID)) {
      mItem = DummyContent.ITEM_MAP
        .get(getArguments().getString(ARG_ITEM_ID));
    }
  }
  …
}
```

DePaul University 118

The Detail Fragment onCreate()

```
public class WineDetailFragment extends Fragment {
  public static final String ARG_ITEM_ID = "item_id";
  private Wine wine;
  public WineDetailFragment() {}
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    if (getArguments().containsKey(ARG_ITEM_ID)) {
      wine = WineList.WINE_MAP
        .get(getArguments().getString(ARG_ITEM_ID));
    }
  }
  …
}
```

Refactor, rename

DePaul University 119

The Detail Fragment onCreateView()

```
public class WineDetailFragment extends Fragment {
  …
  @Override
  public View onCreateView(LayoutInflater inflater,
          ViewGroup container, Bundle savedInstanceState) {
    View rootView = inflater.inflate(
        R.layout.fragment_wine_detail, container, false);
    if ( mItem != null) {
      ((TextView) rootView.findViewById(R.id.wine_detail))
        .setText( mItem.content );
    }
    return rootView;
  }
  …
```

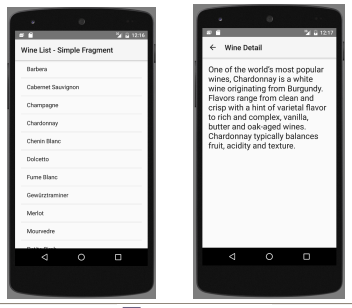
DePaul University 120

The Detail Fragment onCreateView()

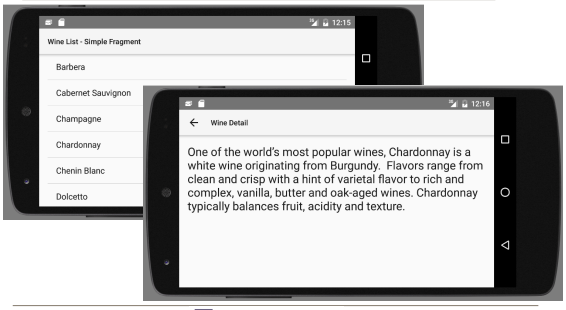
```
public class WineDetailFragment extends Fragment {
  …
  @Override
  public View onCreateView(LayoutInflater inflater,
          ViewGroup container, Bundle savedInstanceState) {
    View rootView = inflater.inflate(
        R.layout.fragment_wine_detail, container, false);
    if ( wine != null) {
      ((TextView) rootView.findViewById(R.id.wine_detail))
        .setText( wine.getLongDescription() );
    }
    return rootView;
  }
  …
```

DePaul University 121

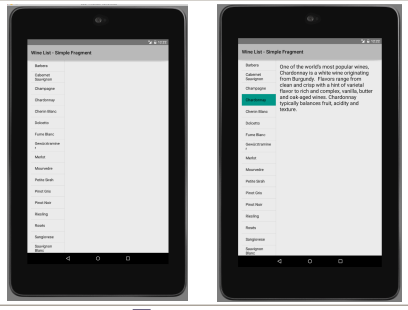
The Wine List App – Handset (Nexus 5), Portrait



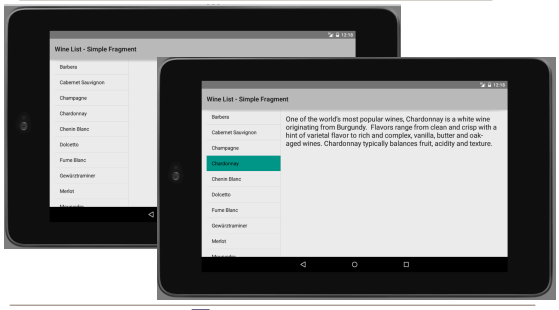
The Wine List App – Handset (Nexus 5), Landscape



The Wine List App – Tablet (Nexus 7), Portrait



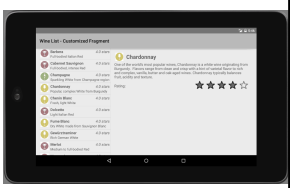
The Wine List App – Tablet (Nexus 7), Landscape



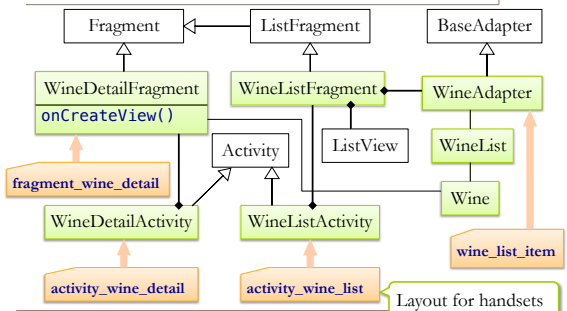
Wine List with Ratings – Customized Fragments

Wine List with Ratings – Using Customized Fragments

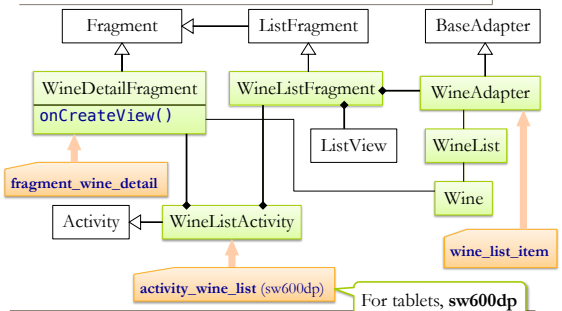
- A variation of the *Wine List Ratings* app
  - Use a two-pane layout for tablets
  - Customize *Fragments*
  - Extend the simple *Fragment* version of the *Wine List* app
- Challenge
  - Update the master *Fragment* when the rating changes
  - Detail ➔ Master



Wine List Rating App – Architecture in Single Pane Mode



Wine List Rating App – Architecture in Two-Pane Mode



Customize the List View in Wine List Fragment

- Use the same row layout in *Wine List Ratings* app
  - layout/wine_list_item.xml

- Use the same *Wine Adapter* in *Wine List Ratings* app

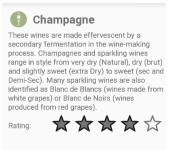
Wine List Fragment

```
public class WineListFragment extends ListFragment
    implements … {
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setListAdapter(new WineAdapter(getActivity()));
  }
  …
  static class WineAdapter extends BaseAdapter {
    Same as in Wine List Ratings
  }
}
```


Customize UI of Detail Fragment

- Replace the content of
  - layout/fragment_wine_detail.xml

  Use the same layout resource for the rating *Activity* of *Wine List Ratings* app
- Set the data in `onCreateView()`



The Detail Fragment – onCreate()

```
public class WineDetailFragment extends Fragment {
  public static final String ARG_ITEM_ID = "item_id";
  private Wine wine;
  public WineDetailFragment() {}
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    if (getArguments().containsKey(ARG_ITEM_ID)) {
      wine = WineList.WINE_MAP
        .get(getArguments().getString(ARG_ITEM_ID));
    }
  }
  …
}
```

Holds a reference to a *Wine* object in the *Wine List*, which is used by the *Wine Adapter*

The Detail Fragment – onCreateView()

```
public View onCreateView(LayoutInflater inflater,
          ViewGroup container, Bundle savedInstanceState) {
  View rootView = inflater.inflate(
       R.layout.fragment_wine_detail, container, false);
  if ( wine  != null) {
    TextView name = (TextView) rootView.findViewById(R.id.text1);
    TextView description = (TextView) rootView.findViewById(R.id.text2);
    ImageView icon = (ImageView) rootView.findViewById(R.id.image);
    RatingBar rating = (RatingBar) rootView.findViewById(R.id.rating);

    name.setText(wine.getName());
    description.setText(wine.getLongDescription());
    icon.setImageResource(Wine.getIconResource(wine.getType()));
    rating.setRating(wine.getRating());
    …
  }
  return rootView;
}
```

How to update the rating change in the List Fragment?

Notify Data Changes

- In two-pane mode, rating change must be reflected immediately in the *List View*
  - Use the *Publisher-Subscriber* pattern
  - Define a callback interface for rating changes.
  - The *Wine Detail Fragment* is the *Publisher*.
  - Make *Wine List Fragment* a listener, i.e., the subscriber, to receive notifications on changes
- In single pane mode, the *List View* only needs to be updated when the detail view is popped.
  - Use the `onResume()` callback to update the *List View*

The Detail Callback Interface

```
public class WineDetailFragment extends Fragment {
  …
  public interface DetailCallbacks {
    public void onItemChanged();
  }

  private DetailCallbacks mCallbacks;
  …
}
```

A user-defined listener interface

The *Subscriber*.

Manage the The Callback Listener

```
public class WineDetailFragment extends Fragment {
  …
  @Override public void onAttach(Activity activity) {
    super.onAttach(activity);
    FragmentManager fragmentManager = activity.getFragmentManager();
    Fragment wineListFragment =
      fragmentManager.findFragmentById(R.id.wine_list);
    if (wineListFragment instanceof DetailCallbacks) {
      mCallbacks = (DetailCallbacks) wineListFragment;
    }
  }
  @Override public void onDetach() {
    super.onDetach();
    mCallbacks = null;
  }
}
```


Notify Changes through Callback

```
public View onCreateView(LayoutInflater inflater,
          ViewGroup container, Bundle savedInstanceState) {
  View rootView = inflater.inflate(…);
  if ( wine  != null) { …
    RatingBar rating = (RatingBar) rootView.findViewById(R.id.rating);
    rating.setRating(wine.getRating());
    rating.setOnRatingBarChangeListener(
        new RatingBar.OnRatingBarChangeListener() {
      @Override public void onRatingChanged(RatingBar ratingBar,
                                            float v, boolean b) {
        wine.setRating(v);
        if (mCallbacks != null) {
          mCallbacks.onItemChanged();
        }
      }
    });
  }
  return rootView;
}
```

Receive Change Notifications

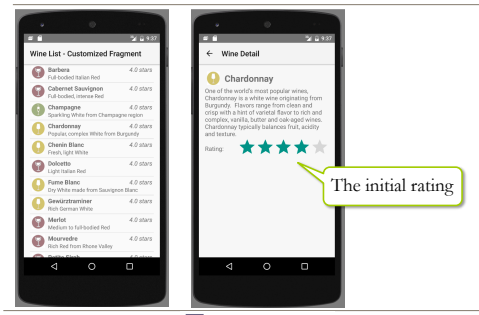
- The callback method

```
public class WineListFragment extends ListFragment
  implements  WineDetailFragment.DetailCallbacks {
  …
  @Override public void onItemChanged() {
    ((WineAdapter) getListAdapter()).notifyDataSetChanged();
  }
}
```

- For the single pane mode

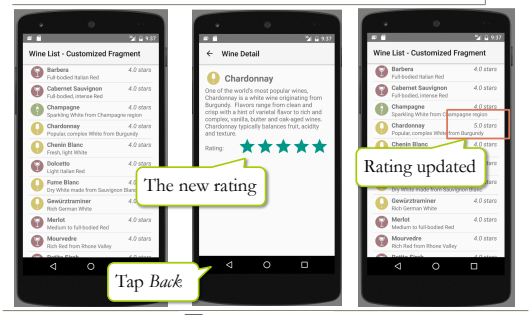
```
@Override public void onResume() {
  super.onResume();
  ((WineAdapter) getListAdapter()).notifyDataSetChanged();
}
```

The Wine List Rating App – Handset (Nexus 5)



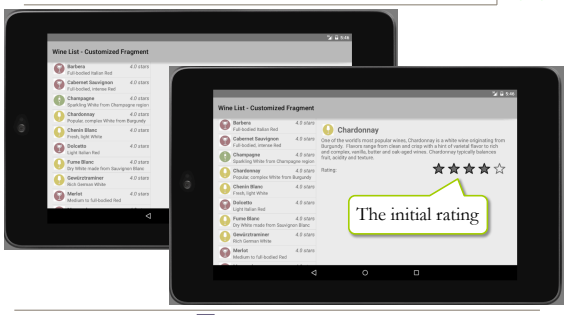
DePaul University 148

The Wine List Rating App – Handset (Nexus 5)



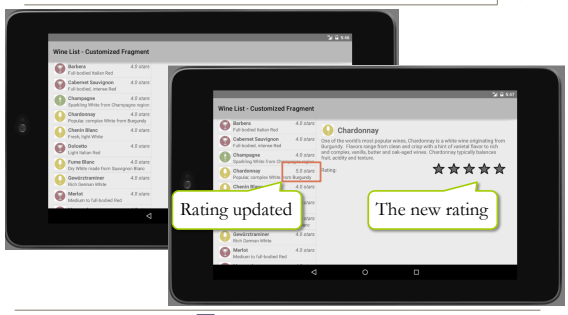
DePaul University 152

The Wine List Rating App – Tablet (Nexus 7)



DePaul University 155

The Wine List Rating App – Tablet (Nexus 7)



DePaul University 157

The Sample Code

- The sample apps in this lecture are available in D2L
  - WineList-SimpleFragment.zip
  - WineList-CustomizedFragment.zip
- Each zip archive contains the entire project folder
- Unzip the file and import to Android Studio

DePaul University 158

Next …

- Threads
- Two dimensional drawing
- Animations

DePaul University 159