CS371m - Mobile Computing

More UI
Navigation, Fragments, and Action Bars
EFFECTIVE ANDROID NAVIGATION
Back and Up

• Android design and developer documentation stresses the desire for consistent global navigation in and between apps

• Android 2.3 and earlier relied on the Back button for navigation within app
Action Bar Navigation

• With addition of the action bar another navigation option was added
• Up
• App icon and left pointing caret
Activity Hierarchy Within Apps

- from Google IO 2012
Up vs. Back

• Up is used to navigate between screens / activities within an app
• Up is to move through the hierarchy of screens within an app
• Example: Tic Tac Toe
  – Settings Activity
  – should offer icon and Up option on action bar to get back to main Tic Tac Toe screen
Up vs. Back

http://developer.android.com/design/patterns/navigation.html
Back Button Actions

- Back still used to move through apps and activities in reverse time order
- Back button also:
  - dismisses floating windows such as dialogs or popups
  - dismisses contextual action bars
  - hides the onscreen keyboard
Back vs. Up

• Many times Up functions **exactly** like Back— as shown in Gmail example on previous slide
• If a screen / activity accessible from multiple other screens in app
• Up takes user from screen / activity to previous screen / activity
• **same as back**
Back and Up Equivalent in Many Cases

Example: swiping between items in a list.

http://developer.android.com/design/patterns/navigation.html
Back vs. Up

• **Sometimes** back and up lead to different behavior
• Browsing related detailed views not tied together by list view - up hierarchy
• Google Play - albums by same artist or apps by the same developer
Back vs. Up
Back vs. Up

• Starting at screen / activity deep inside of an app
  – Another instance where Back and Up are not the same

• Widgets on home screen, notifications, or pop up notifications may take user deep into application

• In this case Up should take user to the logical parent of the screen / view / UI
Specifying Up Button Behavior

• Done in the manifest file for Android 4.0 and higher

```xml
<application ...

... 

<!-- The main/home activity (it has no parent activity) -->
<activity
    android:name="com.example.myfirstapp.MainActivity" ...
    ...
</activity>

<!-- A child of the main activity -->
<activity
    android:name="com.example.myfirstapp.DisplayMessageActivity"
    android:label="@string/title_activity_display_message"
    android:parentActivityName="com.example.myfirstapp.MainActivity"/>

<!-- Parent activity meta-data to support 4.0 and lower -->
<meta-data
    android:name="android.support.PARENT_ACTIVITY"
    android:value="com.example.myfirstapp.MainActivity" />
</activity>
</application>
```
Specifying Up Button Behavior

• Adding Up Action, in onCreate of Activity

```java
@Override
public void onCreate(Bundle savedInstanceState) {
    ...
    getActionBar().setDisplayHomeAsUpEnabled(true);
}
```

• When icon pressed onOptionsItemSelected called

```java
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
    // Respond to the action bar's Up/Home button
    case android.R.id.home:
        NavUtils.navigateUpFromSameTask(this);
        return true;
    }
    return super.onOptionsItemSelected(item);
}
```
Specifying Up Behavior - Other App Started

```java
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        // Respond to the action bar's Up/Home button.
        case android.R.id.home:
            Intent upIntent = NavUtils.getParentActivityIntent(this);
            if (NavUtils.shouldUpRecreateTask(this, upIntent)) {
                // This activity is NOT part of this app's task, so create a new task
                // when navigating up, with a synthesized back stack.
                TaskStackBuilder.create(this)
                    // Add all of this activity's parents to the back stack
                    .addNextIntentWithParentStack(upIntent)
                    // Navigate up to the closest parent
                    .startActivities();
            } else {
                // This activity is part of this app's task, so simply
                // navigate up to the logical parent activity.
                NavUtils.navigateUpTo(this, upIntent);
            }
        return true;
    }
    return super.onOptionsItemSelected(item);
```
FRAGMENTS
Fragments

• Added in Android 3.0 / API level 11, a release aimed at tablets

• A fragment is a portion of the UI in an Activity

• multiple fragments can be combined into multi-paned UI

• fragments can be used in multiple activities—an attempt to create re-usable UI components
Fragments

• Part of an activity
  – directly affected by Activity's lifecycle

• Fragments can be swapped into and out of activities without stopping the activity

• On a handset one with limited screen space, common for app to switch from one activity to another
  – with a larger screen swap fragments in and out, don't start new Activity
Fragments

old

new

List View screen

WebView screen

ListView fragment

WebView fragment

Learn Fragments

Learn Fragments

Large screen

When list item clicked
Use of Fragments

• For a time Android development documents recommended ALWAYS using Fragments
• Now (2016) not as much
• Provide for flexibility of UIs
• Activity tightly coupled with its View
• Fragments provide flexibility, looser coupling between Activity and UI Views
  – fragment becomes a building block
• downside, more complexity in code, more moving parts
Fragments

• Fragments can typically control a UI
  – fragment has view that is inflated from a layout file
  – inflate: create runtime objects for values specified in xml layout file
  – elements of layout measured and drawn

• Activity can specify spots for fragments
  – in some instances one fragment
  – in other instance multiple fragments
  – can change on the fly
Fragments

• Have a life cycle similar to Activities

• But, Fragment lifecycle controlled by Activity not by the system – more complex, but more flexible
Fragment Example

• From the apiDemos app on the emulator
  – part of the sample code with Android SDK
• Displays Shakespeare play titles in a List
• Clicking on a title displays a sample from the play
• com.example.android.apis.app
  – FragmentLayout.java
Hear him but reason in divinity, and all-admiring with an inward wish you would desire the king were made a prelate: Hear him debate of commonwealth affairs, you would say it hath been all in all his study: List his discourse of war, and you shall hear a fearful battle render'd you in music: Turn him to any cause of policy, the Gordian knot of it he will unloose, familiar as his garter: that, when he speaks, the air, a charter'd libertine, is still, and the mute wonder lurketh in men's ears; To steal his sweet and honey'd sentences; So that the art and practic part of life must be the mistress to this theoric: Which is a wonder how his grace should glean it, since his addiction was to courses vain, his companies unletter'd, rude and shallow, his hours fill'd up with riots, banquets, sports, and never noted in him any study, any retirement, any sequestration from open haunts and popularity.
Portrait

- In portrait view app behaves as you would expect
- the play titles are in a list
  - old approach, would be a ListView inside of an Activity
- clicking a list items creates an Intent that starts another Activity with a TextView inside of a ScrollView
- Click back button to go back to list
Landscape

• When switched to landscape designer decided there is enough real estate to display list and summary side by side
  – imagine an app that looks one way on phone another way on a tablet
Landscape

FragmentLayout Activity

TITLES FRAGMENT

DETAILS FRAGMENT
TitlesFragment

• extends the ListFragment class
  – other useful subclasses of Fragment
  – DialogFragment
  – PreferenceFragment
  – WebViewFragment

• Displays a list of Shakespeare play titles
Summary - Detail Fragment

• Displays some prose from the play
• A subclass of Fragment
• Sometimes displayed in the FragmentLayout Activity
  – landscape
• Sometimes displayed in a DetailsActivity Activity
  – portrait
General approach for creating a Fragment:

1. Create user interface by defining widgets in a layout file
2. create Fragment class and set view to be defined layout
   – in the onCreateView method
3. wire up widgets in the fragment when inflated from layout code

From Android Programming - The Big Nerd Ranch Guide
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/
android:layout_width="match_parent"
android:layout_height="match_parent">
    <TextView
        android:id="@+id/text_view_fragment_detail"
        android:padding="10dp"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="18sp"
        android:orientation="vertical"/>
</ScrollView>
DetailsFragment

```java
public static class DetailsFragment extends Fragment {
    //

    • If necessary override onCreate(Bundle)
    • DO NOT inflate the View in onCreate
      – just complete any items for Fragment to get ready other than the View
      – internal logic / object data for example
```
DetailFragment

- onCreateView method used to inflate View
  - generally must override this method

```java
@override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                           Bundle savedInstanceState) {

    View v = inflater.inflate(R.layout.detail_fragment_layout,
                               container, false);
    Log.d("FRAGMENT", "" + v);
    TextView tv =
        (TextView) v.findViewById(R.id.text_view_fragment_detail);

    tv.setText(Shakespeare.DIALOGUE[getShownIndex()]);

    return v;
```
getShownIndex

• In the DetailsFragment
• returns int corresponding to which play is currently displayed

```java
public int getShownIndex() {
    return getArguments().getInt("index", 0);
}
```

• used in DetailsFragment onCreateView to find proper text and in TitlesFragment to decide if new Fragment needed
getArguments

• Fragments can have a Bundle object attached to them
• referred to as arguments
• Create Bundle and attach after fragment created, but before fragment added to Activity
• convention: create static method newInstance that creates Fragment and bundles up arguments
/**
 * Create a new instance of DetailsFragment, initialize
 * show the text at 'index'.
 */

public static DetailsFragment newInstance(int index) {
    DetailsFragment f = new DetailsFragment();

    // Supply index input as an argument.
    Bundle args = new Bundle();
    args.putInt("index", index);
    f.setArguments(args);

    return f;
}
List of Titles

• Uses a ListFragment
  – analogous to a ListActivity

```java
public static class TitlesFragment extends ListFragment {

    boolean mDualPane;
    int mCurCheckPosition = 0;
}
```

• Top level fragment in example
ListFragment

• No layout necessary as ListFragments have a default layout with a single ListView

• Set up done for this Fragment done in onActivityCreated
TitlesFragment onActivityCreated

Called when the Activity that holds this Fragment has completed its onCreate method

```java
@override
public void onActivityCreated(Bundle savedInstanceState) {
    super.onActivityCreated(savedInstanceState);

    // Populate list with our static array of titles.
    setListAdapter(new ArrayAdapter<String>(getActivity(),
        android.R.layout.simple_list_item_item_activated_1,
        Shakespeare.TITLES));

    // Check to see if we have a frame in which to embed the details
    // fragment directly in the containing UI.
    View detailsFrame = getActivity().findViewById(R.id.details);
    mDualPane = detailsFrame != null
        && detailsFrame.getVisibility() == View.VISIBLE;
```
if (savedInstanceState != null) {
    // Restore last state for checked position.
    mCurCheckPosition
        = savedInstanceState.getInt("curChoice", 0);
}

if (mDualPane) {
    // In dual-pane mode, the list view highlights the selected item.
    getListView().setChoiceMode(ListView.CHOICE_MODE_SINGLE);
    // Make sure our UI is in the correct state.
    showDetails(mCurCheckPosition);
}
showDetails

• used to show portion of play selected from the list fragment

• in portrait mode, starts a new Activity
  – DetailsActivity that hosts a DetailsFragment
  – similar to what we have seen before, one Activity, starting another Activity with an Intent

• in landscape mode (mDualPane) if the DetailsFragment does not exist or is a different play, a new DetailsFragments is created
TitlesFragment ShowDetails

• Portrait mode - !mDualPane
• traditional start another Activity via an Intent

```java
} else {
    // Otherwise we need to launch a new activity to display the dialog fragment with selected text.
    Intent intent = new Intent();
    intent.setClass(getActivity(), DetailsActivity.class);
    intent.putExtra("index", index);
    startActivity(intent);
}
```
TitlesFragment ShowDetails

- DetailsFragment placed side by side with titles

```java
if (mDualPane) {
    // We can display everything in-place with fragments, so update
    // the list to highlight the selected item and show the data.
    getListView().setItemChecked(index, true);

    // Check what fragment is currently shown, replace if needed.
    DetailsFragment details = (DetailsFragment)
        getFragmentManager().findFragmentById(R.id.details);
```
if (details == null || details.getShownIndex() != index) {
    // Make new fragment to show this selection.
    details = DetailsFragment.newInstance(index);

    // Execute a transaction, replacing any existing fragment
    // with this one inside the frame.
    FragmentTransaction ft
        = getFragmentManager().beginTransaction();

    ft.replace(R.id.details, details);
    ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE);
    ft.commit();
}
Using the Fragments

• Activities add Fragments in two ways:
  1. As part of the layout file (hard coded, less flexible)
  2. Programmatically (in the code, more flexible)
Shakespeare Example

• Titles Fragment in the layout file, hard coded
• One layout file for portrait, single fragment
• In landscape layout file:
  – the other fragment, the details fragment, is added programmatically
Name of Fragment class:
FragmentLayout$TitlesFragment
an inner class
Shakespeare Landscape Layout

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal" />

<fragment
    android:id="@+id/titles"
    android:layout_width="0px"
    android:layout_height="match_parent"
    android:layout_weight="1"
    class="com.example.android.apis.app.FragmentLayout$TitlesFragment" />

<FrameLayout
    android:id="@+id/details"
    android:layout_width="0px"
    android:layout_height="match_parent"
    android:layout_weight="1"
    android:background="@android:attr/detailsElementBackgroundColor" />
</LinearLayout>
Adding Fragment Programmatically

• Back to TitleFragment showDetails method

```java
if (mDualPane) {
    // We can display everything in-place with fragments, so update
    // the list to highlight the selected item and show the data.
    listView().setItemChecked(index, true);

    // Check what fragment is currently shown, replace if needed.
    DetailsFragment details = (DetailsFragment)
        fragmentManager().findFragmentById(R.id.details);
```
if (details == null || details.getShownIndex() != index) {
    // Make new fragment to show this selection.
    details = DetailsFragment.newInstance(index);

    // Execute a transaction, replacing any existing fragment
    // with this one inside the frame.
    FragmentTransaction ft
        = getFragmentManager().beginTransaction();

    ft.replace(R.id.details, details);
    ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE);
    ft.commit();
}
Adding a Fragment

• To add fragment to an Activity during runtime:
  • must specify a ViewGroup in the Activity's layout to place the fragment
  • In Shakespeare Activity it is the FrameLayout, second element in LinearLayout in the portrait layout file
Adding a Fragment

• To actually add the Fragment must get the `FragmentManager` for the Activity
• and perform a `FragmentTransaction`
• `Activity.getFragmentManager()` and `Fragment.getFragmentManager()`
FragmentManager

- In example:
- A little odd that it is the TitleFragment, not the Activity managing the DetailsFragment
- Fragment manager used to determine if fragment already exists
- uses id for layout
  - for Fragments without a layout findFragmentByTag method
FragmentManager

• maintains a *Back Stack* of fragment transactions
• analogous to the Activity Stack
• allows Activity to go back through changes to fragments, like back button and activities themselves
• methods to get Fragments, work with back stack, register listeners for changes to back stack
FragmentTransaction

• Make changes to fragments via FragmentTransactions
• obtained via FragmentManager
• used to add, replace, remove Fragments
if (details == null || details.getShownIndex() != index) {
    // Make new fragment to show this selection.
    details = DetailsFragment.newInstance(index);

    // Execute a transaction, replacing any existing fragment
    // with this one inside the frame.
    FragmentTransaction ft
        = getFragmentManager().beginTransaction();

    ft.replace(R.id.details, details);
    ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE);
    ft.commit();
}
Inter Fragment Communication

• In an Activity with multiple Fragments, the Fragments sometimes have to send information back and forth
• Fragment to Fragment communication is frowned upon
• Instead use the Activity that holds the Fragments to pass messages around
• Create your own interface with call back methods
  – fragment defines the interface
  – Activity implements the interface
Options Menu and Action Bar

• prior to Android 3.0 / API level 11 Android devices required a dedicated menu button
• Pressing the menu button brought up the options menu
action bar

• menu button no longer required
• shift options menu to action bar
• action bar is a combination of on-screen action items and overflow options

Action items  overflow options  overflow menu

action bar
Action Bar

- identify app and users location in app
- display important actions
  - old options menu
- support consistent navigation and view switching within the app

![Image of Action Bar with labels for Icon/Logo, Navigation Tabs, Overflow Menu, and Action Item]
Action Bar

- ActionBar items declared in menu.xml

```xml
<menu xmlns:android="http://schemas.android.com/apk/res/android">
  <item
    android:id="@+id/new_game"
    android:icon="@drawable/new_game"
    android:title="New Game"
    android:showAsAction="ifRoom|withText"/>
</menu>
```
Action Bar

• If menu items declared in xml, added to menu in order they appear

• Extra items brought up with overflow button
Action Bar

• When activity starts
• Action Bar populated by a call to Activity's onCreateOptionsMenu method
• This method inflates (converts XML into runtime objects) the menu resource that defines all the action items
Action Bar Items in XML

```xml
res/menu/main_activity_actions.xml

<menu xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:id="@+id/action_search"
        android:icon="@drawable/ic_action_search"
        android:title="@string/action_search"/>
  <item android:id="@+id/action_compose"
        android:icon="@drawable/ic_action_compose"
        android:title="@string/action_compose"/>
</menu>
```
@Override
public boolean onCreateOptionsMenuMenu(Menu menu) {
    // Inflate the menu items for use in the action bar
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.main_activity_actions, menu);
    return super.onCreateOptionsMenuMenu(menu);
}

Request item be shown on Action Bar (instead of overflow menu) with ifRoom attribute

android:showAsAction="ifRoom|withText"
Split Action Bar

• Split Action Bar between top and bottom of screen
  – especially if narrow screen
  – more room for action items
  – declaration in manifest file
Navigation Tabs

• Used to switch between fragments

Action Views

• Action views appear in the action bar in place of action buttons
• Accomplish some common action
• Such as searching
Enabling ActionViews

- use either the actionLayout or actionViewClass attribute
- specify either a layout resource or widget class to use, respectively

```xml
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:yourapp="http://schemas.android.com/apk/res-auto" >
  <item android:id="@+id/action_search"
        android:title="@string/action_search"
        android:icon="@drawable/ic_action_search"
        yourapp:showAsAction="ifRoom|collapseActionView"
        yourapp:actionViewClass="android.support.v7.widget.SearchView"/>
</menu>
```
ActionProviders

• Similar to ActionView in that it replaces an action button with a customized layout
• but can also display a submenu
• create your own subclass of ActionProvider
• or use a prebuilt ActionProvider such as ShareActionProvider (shown above) or MediaRouteActionProvider
MORE ACTION BAR NAVIGATION
Action Bar Navigation

• Action Bar can also be used for in app navigation beyond the Up button

• Two Options:
  • Navigation Tabs
  • Drop Down Navigation
Action Bar Navigation Tabs

- wide screen action bar

- narrow screen stacked action bar
Action Bar Drop Down Navigation

• Alternative to tabbed navigation in action bar

• Create a spinner drop down list that is accessed with "down triangle"
Action Bar on pre Android 3.0

• pre 3.0 a little more than 25% of Android OS versions as of November 2013
• Support library includes provides code and classes to allow some newer features of Android to be used on older versions
• Example: ActionBar
• 3rd Party tool - ActionBarSherlock
  – deal with Action Bar via single API
  – http://actionbarsherlock.com/
OTHER MENUS
Menus

• Three types of menus:
• options menu or action bar
• context menu and contextual action mode
• popup menu
ContextMenu

• pre 3.0, aka Floating Menus
• subtype of Menu
• display when a long press is performed on a View
  – Activity is a descendant of View
  – Activity may be broken up into multiple views
• implement `onCreateContextMenu` method
• must call `registerForContextMenu` method and pass View
ContextMenu

• From Tip Calculator
• Long press on total amount EditText
• Default behavior for EditText
• Nothing added in TipCalculator to create this
Contextual Action Mode

• Android 3.0 and later
• Menu that affects a specific item in the UI
  – typically a View
  – used most often for elements in ListView or GridView
floating context menu

http://developer.android.com/guide/topics/ui/menus.html#CAB
floating context menu

• register View with Activity.registerForContextMenu()

• can send ListView or GridView to method to register all elements

• Implement View.onCreateContextMenuListener
  – long click leads to method call
  – inflate menu (like action items/ options menu)

• Implement Activity.onContextItemSelected
contextual action mode

• alternative to floating context menu
• causes contextual action bar to appear at top of screen
• independent of regular action bar but, does overtake position of action bar
• For Android 3.0 and higher preferred to floating context menus
• Implement ActionMode.Callback interface
  – similar to options menu methods
STYLES
Styles

• Defined in XML file
• res/values/style
• similar to a cascading style sheet as used in html
• group layout attributes in a style and apply to various View objects (TextView, EditText, Button)
Sample Styles, in styles.xml

```xml
<style name="sample1">
  <item name="android:textSize">20pt</item>
  <item name="android:textColor">@color/Orange</item>
  <item name="android:textStyle">bold</item>
  <item name="android:gravity">center</item>
  <item name="android:padding">10dp</item>
</style>

<style name="sample2">
  <item name="android:textSize">8pt</item>
  <item name="android:textColor">@color/AliceBlue</item>
  <item name="android:textStyle">italic</item>
  <item name="android:gravity">right</item>
  <item name="android:padding">2dp</item>
</style>
```
<TextView
    android:id="@+id/textView1"
    style="@style/sample2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="field number 1" />

<EditText
    android:id="@+id/editText1"
    style="@style/sample1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:inputType="textCapWords"
    android:text="First Edit Text" />

<TextView
    android:id="@+id/textView2"
    style="@style/sample2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="field number 2" />
Result of Styles

- can override elements of style
  - bottom edit text overrides color
- one style can inherit from another
- use UI editor to create view and then extract to style