Fragments

- To handle various screen configurations
- Instead of one screen per activity
  - subdivide UI
  - present on one screen or multiple screens
- Fragment is like a simple Activity with a UI
  - UI defined by XML layout file
- Fragment has its own lifecycle
- Fragment in layout
  - fragment XML tag
  - name
  - of Fragment class

Figure 6.1  Traditional screen workflow without fragments.
Two presentations
- portrait – separate screens
- landscape – both on same screen

XML files for fragments

FragmentOne
- handles list for selection
  - onCreateView
    - context
    - inflate UI
  - onItemViewClick
    - orientation
      - landscape – have both views so update
      - portrait – only 1 view so start second activity passing selection

E.g. FragmentsProject
• FragmentTwo
  - displays selected item
  - inflate
• XML files for main activity
  - 2 versions – portrait & landscape
  - fragment tag
  * name – associated Fragment class
• MainActivity
  - portrait-landscape transition handled by resource loader
• ShowItemActivity
  - receives intent from FragmentOne
  - orientation landscape – nothing to do – both views visible
  - get selected item from Bundle & display (in second fragment)
• Manifest
  - minimum SDK level 11 (Fragments)
  - register ShowItemActivity

Creating Fragments in Code

• FragmentManager
  - manage fragments in an activity
  - access, add, remove, replace fragments
• Access
  - findFragmentByTag, findFragmentById
• FragmentTransaction
  - operations to modify fragments in activity (manager)
  - beginTransaction
  - add(container, fragment, tag)
  - replace(container, fragment, tag)
  - remove(fragment)
  - addToBackStack(name)
  - commit

• Passing arguments to fragments
  - setArguments(bundle)
• Retrieving arguments
  - getArguments
• Fragment state
  - onCreate
  * gets state
  - onSaveInstanceState
  * add items to Bundle
  - onRestoreInstanceState
  * restore items from bundle
E.g. FragmentsTwoProject

- Dynamically add fragments
- Same XML files for fragments
- XML files for main activity
  - use LinearLayout instead of fragment since dynamically loading
- FragmentOne
  - onCreate
  - obtain fragment manager
  - onItemClick
  - if second fragment exists must be in landscape mode
- FragmentTwo
  - unchanged

- MainActivity
  - get fragment manager
  - build transaction to add one or two fragments
    - fragment tag
      - identifies fragment in manager
    - in portrait, remove FragmentTwo if it exists
- ShowItemActivity
  - unchanged
- Manifest
  - unchanged

Fragment Subclasses

- Help in common uses of fragments
  - like specialized activities
- e.g.
  - ListFragment
  - DialogFragment
  - PreferenceFragment
ListFragment

- Like ListActivity
- Fragment with built-in ListView
  - no need for UI XML file
  - implements onListItemClick of ListView
- Methods
  - getListView()
  - setListAdapter(adapter)
  - onListItemClick(...)
- E.g. ListFragProject
  - simplified version that handles side-by-side layout of fragments
  - no XML for FragmentOne since extends ListFragment

Dialogs

- Feedback to user without replacing screen content
- Introduction of Fragments in API 11 changed model for dialogs
  - legacy apps will use Activity callbacks
  - new apps should use DialogFragment
  - callbacks are deprecated since API 13 but will still be available
- Classes for standard dialogs
  - Dialog, AlertDialog, CharacterPickerDialog, DatePickerDialog, TimePickerDialog, ProgressDialog
  - complex dialogs have Builder classes

DialogFragment

- Subclass of Fragment
- Encapsulates dialog activity and usable by more than one activity
- Subclass to define dialog
  - implement onCreateDialog
    - creates and returns instance of Dialog subclass
    - default implementation creates a simple Dialog
  - use Builder to create dialog
  - show method displays dialog defined by the fragment
    - FragmentManager handles synchronization of fragment & dialog lifecycle
    - tag for later identification of fragment
- E.g. DialogFrag
  - buttons trigger two different versions of dialog
  - factory method newInstance
  - onCreateDialog picks up args and builds dialog
  - onClick fires main activity's doPositiveButton
Preferences

- Lightweight persistent storage for application data
  - primitive data only
- Intended for user selected settings that are to be retained between use
- Stored as key/value pairs
- 2 types
  - private preferences
    - local to activity
  - shared preferences
    - all Activities in application
- Both use SharedPreferences interface

SharedPreferences

- In android.content
- Stored as XML file in the Android file system on the device
- Private preferences use Activity name
  - only one
- Shared preferences require a filename
  - as many as desired
- Process
  - access a preferences file (instance of SharedPreferences)
  - create a preferences editor (SharedPreferences.Editor)
  - use editor to modify preferences
  - commit the result

```java
import android.content.SharedPreferences;
SharedPreferences settings = getPreferences(MODE_PRIVATE);
SharedPreferencesEditor prefEdit = settings.edit();
prefEditor.putLong("SomeLong",1.0L);
prefEditor.commit();
```
• Methods
  - access
  - editing

• Handling changes to preferences
  - `settings.registerOnSharedPreferenceChangeListener`

• Location
  `/data/data/<package name>/shared_prefs/<pref filename>.xml`

• XML file
  - describes a map

### Table 12.1: Important `SharedPreferences` Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>contains()</code></td>
<td>Tests whether a specific preference exists by name.</td>
</tr>
<tr>
<td><code>edit()</code></td>
<td>Returns the editor to change these preferences.</td>
</tr>
<tr>
<td><code>getBoolean()</code></td>
<td>Returns a map of all preference key/value pairs.</td>
</tr>
<tr>
<td><code>getFloat()</code></td>
<td>Returns a specific float-type preference by name.</td>
</tr>
<tr>
<td><code>getInt()</code></td>
<td>Returns a specific integer-type preference by name.</td>
</tr>
<tr>
<td><code>getLong()</code></td>
<td>Returns a specific long-type preference by name.</td>
</tr>
<tr>
<td><code>getString()</code></td>
<td>Returns a specific string-type preference by name.</td>
</tr>
<tr>
<td><code>getSharedPreferences()</code></td>
<td>Returns a specific set of <code>String</code> preferences by name. (Method added in API Level 11.)</td>
</tr>
</tbody>
</table>

### Table 12.2: Important `SharedPreferences.Editor` Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>clear()</code></td>
<td>Removes all preferences. This operation happens before any put operation, regardless of when it is called within an editing session; then all other changes are read and committed.</td>
</tr>
<tr>
<td><code>remove()</code></td>
<td>Removes a specific preference by name. This operation happens before any put operation, regardless of when it is called within an editing session; then all other changes are read and committed.</td>
</tr>
<tr>
<td><code>putBoolean()</code></td>
<td>Sets a specific Boolean-type preference by name.</td>
</tr>
<tr>
<td><code>putFloat()</code></td>
<td>Sets a specific Float-type preference by name.</td>
</tr>
<tr>
<td><code>putInt()</code></td>
<td>Sets a specific Integer-type preference by name.</td>
</tr>
<tr>
<td><code>putLong()</code></td>
<td>Sets a specific Long-type preference by name.</td>
</tr>
<tr>
<td><code>putString()</code></td>
<td>Sets a specific String-type preference by name.</td>
</tr>
<tr>
<td><code>putStringSet()</code></td>
<td>Sets a specific set of <code>String</code> preferences by name. (Method added in API Level 11.)</td>
</tr>
<tr>
<td><code>commit()</code></td>
<td>Combines all changes from this editing session.</td>
</tr>
</tbody>
</table>
PreferenceFragment

- Custom Fragment for handling user preference settings
  - provides standard preference UI
  - updates default shared preferences
- Process
  - define a preference resource file (XML)
  - extend PreferenceFragment class
  - addPreferencesFromResource
  - user interactions automatically update shared preferences
  - getDefaultSharedPreferences

SharedPreferences.Editor.apply()

- Much like the commit() method, this method commits all preference changes from the adding session. However, this method commits the changes to the actual shared preferences immediately, but commits the changes to disk asynchronously within the application thread.

<!-- XML preference resources -->

```xml
<PreferenceFragment>
  ...
</PreferenceFragment>
```
Preference Resource File

- XML file in res/xml
- PreferenceScreen tag
- PreferenceCategory tag
- Preference views
  - define preference UI type
    - e.g. CheckBoxPreference
- Metadata
  - android:key
  - android:title
  - android:defaultValue

Table 6.1 preference views that can be displayed in preferences fragments

<table>
<thead>
<tr>
<th>Preference View</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>preferenceScreen</td>
<td>The root element of the XML used to define a preference screen</td>
</tr>
<tr>
<td>checkBoxPreference</td>
<td>Displays a single check box that returns true when checked or false when unchecked</td>
</tr>
<tr>
<td>radioPreference</td>
<td>Displays a dialog with an edit text dialog allowing the user to enter a text</td>
</tr>
<tr>
<td>radioButtonPreference</td>
<td>Displays radio buttons indicating the options available for selection</td>
</tr>
<tr>
<td>preferenceCategory</td>
<td>A section preference that acts like a section control</td>
</tr>
</tbody>
</table>

E.g. PrefFrag

- Update shared preferences
  - preferences.xml
    - defines UI for updating preferences
    - key is key in preference map
    - value comes from UI selection
- MainActivity
  - starts PrefsActivity to display preferences fragment
  - onResume - when regain control (i.e. after preference selection)
    - access shared preferences
    - extract preference values
    - set text fields for display
- PrefActivity
  - replaces UI from main with preferences fragment
PrefFragment
- addPreferencesFromResource inflates UI for PreferenceScreen
- preferences automatically updated as user interacts with UI
- onPreferenceClick
  - handles click on Submit button
  - terminates activity (prefActivity)
  - MainActivity resumes