Android Fragments
Fragment Transactions

Jim Wilson

jimw@jwhh.com
@hedgehogjim

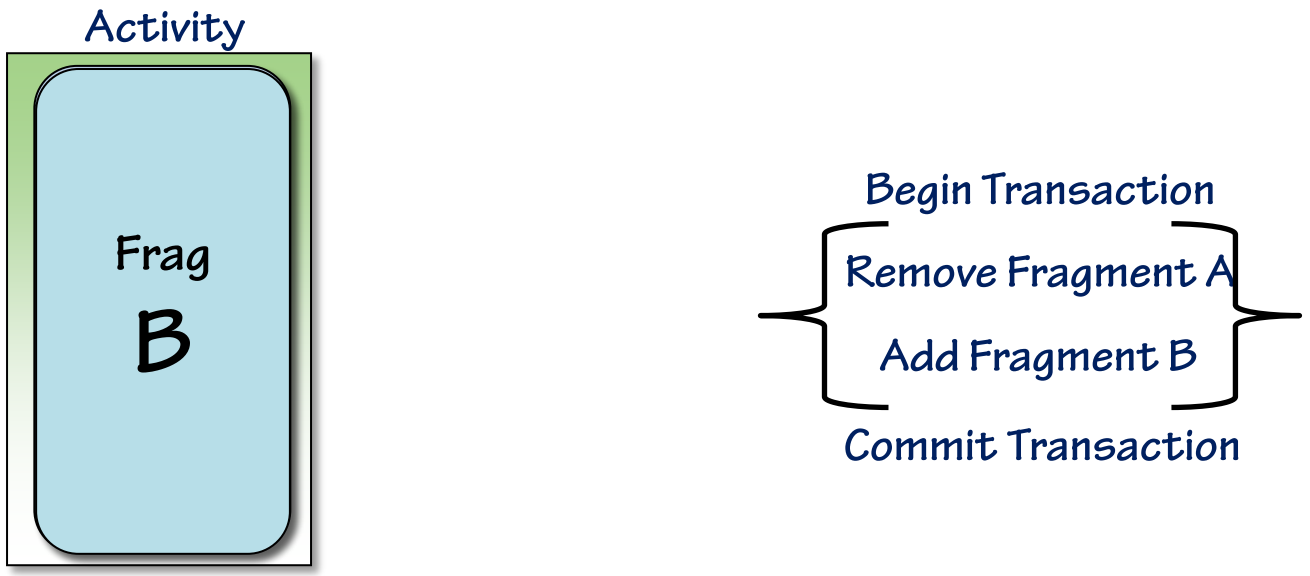
pluralsight
hardcore developer training

Outline

- Dynamically managing Fragments
- Managing Fragment state
- FragmentTransactions and the back button
- Programmatically interacting with the back stack

Dynamically managing Fragments

- **Fragments can be dynamically managed within an Activity**
  - Allows you to rearrange the display without leaving the Activity
  - FragmentTransactions group changes into demonstrable sets
    - All changes must occur within a transaction



FragmentTransaction in code

- **FragmentTransactions follow a common behavior**
  - Request FragmentManager from Activity
  - Create new transaction with FragmentManager
  - Add/Remove/Replace Fragments within the transaction
  - Commit the transaction

```
Class MyActivity extends Activity {
. . .
  void addFragment(Fragment frag) {

    FragmentManager fm = getFragmentManager();

    FragmentTransaction ft = fm.beginTransaction();

    // Perform Fragment action

    ft.commit();

  }
}
```

Dynamically adding Fragments to Activity

- **New Fragments added to Activity with add method**
  - Accepts an already constructed Fragment instance
  - All Fragment setup/display callbacks occur
  - Programmatic equivalent of the <fragment> layout element

```
Class MyActivity extends Activity {
. . .
  void addMyFragments() {
    MyFragment frag = new MyFragment();
    FragmentManager fm = getFragmentManager();
    FragmentTransaction ft = fm.beginTransaction();
    ft.add(R.id.myGroup, frag, "fragtag");
    ft.commit();
  }
}
```

Provide a Tag to simplify locating the Fragment

```
<LinerLayout android:id="@+id/myGroup">

  <fragment class="com.pluralsight.MyFragment"/>

</LinerLayout>
```

Dynamically removing Fragments from Activity

- **Fragments removed from Activity with remove method**
  - Completely removes Fragment from Activity
  - Need to have a reference to the Fragment to be removed
    - Use FragmentManager.findFragmentByTag/findFragmentById
  - Fragment instance can no longer be used within Activity

```
Class MyActivity extends Activity {
. . .
  void removeFragment(Fragment frag) {

    FragmentManager fm = getFragmentManager();

    Fragment frag = fm.findFragmentById(R.Id.myFrag);

    FragmentTransaction ft = fm.beginTransaction();

    ft.remove(frag);

    ft.commit();

  }
}
```

Replacing Fragments within a group

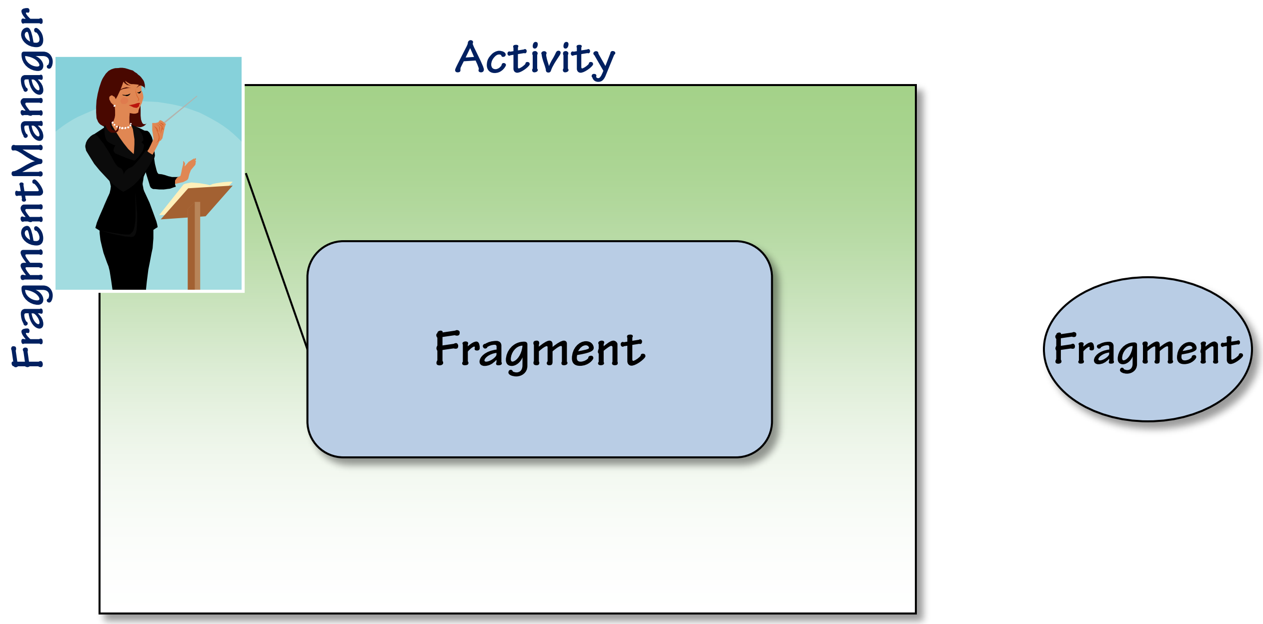
- **Remove one Fragment and add another with remove method**
  - Works within a ViewGroup
    - Existing Fragment receives all teardown callbacks
    - New Fragment receives all setup/display callbacks
  - Convenience method

```
DifferentFragment diffFrag = new DifferentFragment();
FragmentManager fm = getFragmentManager();
FragmentTransaction ft = fm.beginTransaction();
ft.replace(R.id.myGroup, diffFrag);
ft.commit();
```

```
<LinerLayout android:id="@+id/myGroup">
  <fragment
   class="com.pluralsight.MyFragment"
   android:id="@+id/theFrag" />
</LinerLayout>
```

Managing Fragment state

- **Fragments have 3 distinct levels of existence**
  - As a simple Java object
  - Associated with an Activity
  - Rendered UI

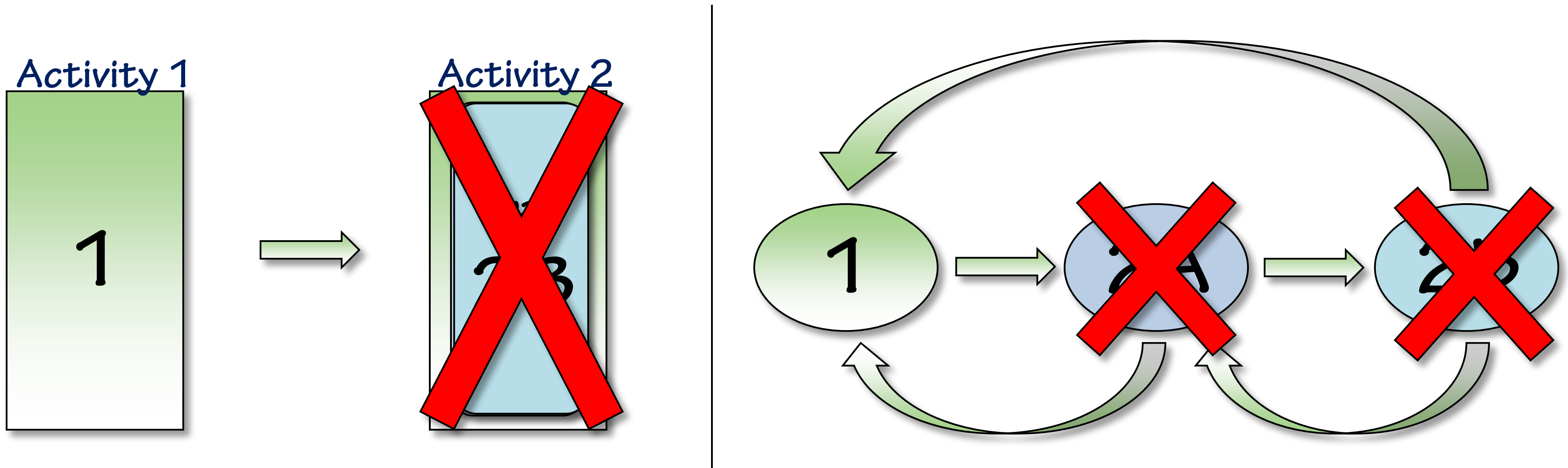


Fragment Attach/Detach

- **Fragment UI can be managed separate from Activity relationship**
  - Useful for frequently moving between Fragments
    - Avoids overhead of complete teardown and resetup
  - FragmentTransaction.detach
    - Tearsdown Fragment UI
    - Fragment instance remains intact and associated with Activity
    - Callbacks received: onPause, onStop, OnDestroyView
    - Note: onDetach callback **not** received
  - FragmentTransaction.attach
    - Reconstructs Fragment UI
    - Callbacks received: onCreateView, onActivityCreated, onStart, onResume
    - Note: onAttach callback **not** received

FragmentTransactions and the back button

- **By default FragmentTransactions have no awareness of back button**
  - Fragment transactions change the screen display state
    - Looks like a "new screen" to the user
    - Does not look like a new screen to back stack
  - User expects Back button to change screen back to previous display state



Programmatic Behavior

User Experience

Add back button support for transactions

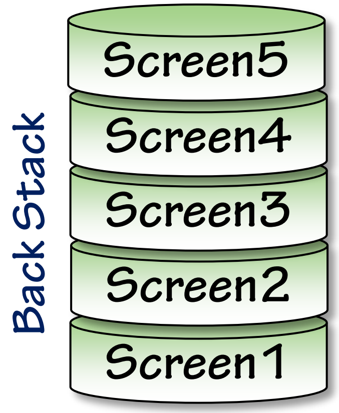
- **Transactions can be placed on the back stack**
  - Call addToBackStack to create new entry
    - Screen will revert to the state prior to transaction when back button pushed
  - Must be called prior to committing the transaction
  - Can optionally include a name for the entry

```
Class MyActivity extends Activity {
. . .
  void addFragmentWithBackStack(Fragment frag) {

    FragmentManager fm = getFragmentManager();

    FragmentTransaction ft = fm.beginTransaction();

    ft.add(R.id.theViewGroup, frag);

    ft.addToBackStack("Screen2");

    ft.commit();

  }
}
```

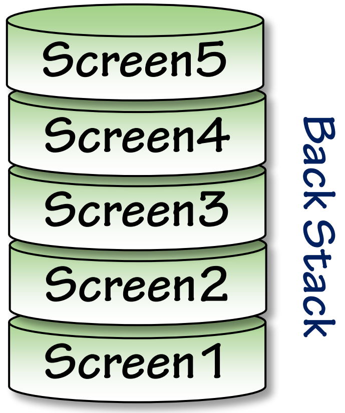
Name useful if you wish to programmatically move to specific entry In the back stack

Programmatically moving through the back stack

- **FragmentManager allows you to directly navigate the back stack**
  - Simulate the back button being pushed
    - Call popBackStack with no arguments
  - Roll the display back to a specific transaction
    - Call popBackStack with the name passed to addToBackStack with that transaction



```
FragmentManager fm = getFragmentManager();
fm.popBackStack("Screen3", 0);
```



```
FragmentManager fm = getFragmentManager();
fm.popBackStack("Screen3", POP_BACK_STACK_INCLUSIVE);
```

Accessing the back stack

- **The FragmentManager provides several ways to access the back stack**
  - Be notified anytime the backstack changes
    - Implement the FragmentManager.OnBackStackChangedListener interface
    - Pass implementation to addBackStackChangedListener
  - Accessing the entries contained within the back stack
    - Get number of entries with getBackStackEntryCount
    - Access a specific entry by index with getBackStackEntryAt
      - Use (getBackStackEntryCount() – 1) to access the entry at the top of back stack

Summary

- **FragmentTransactions are key to dynamically managing Fragments**
- **Fragment association with an Activity is separate from Fragment UI**
  - FragmentTransaction.detach/attach manage UI separate from Activity
  - detach/attach methods do not fire onDetach/onAttach callbacks
- **FragmentTransactions do not affect back stack by default**
  - Use FragmentTransaction.addToBackStack to affect back stack
- **FragmentManager makes back stack programmatically accessible**
  - popBackStack
  - addBackStackChangedListener / OnBackStackChangedListener interface
  - getBackStackEntryCount / getBackStackEntryAt