Fragments


Santa Clara University
Center for Science, Technology, and Society

Fragments

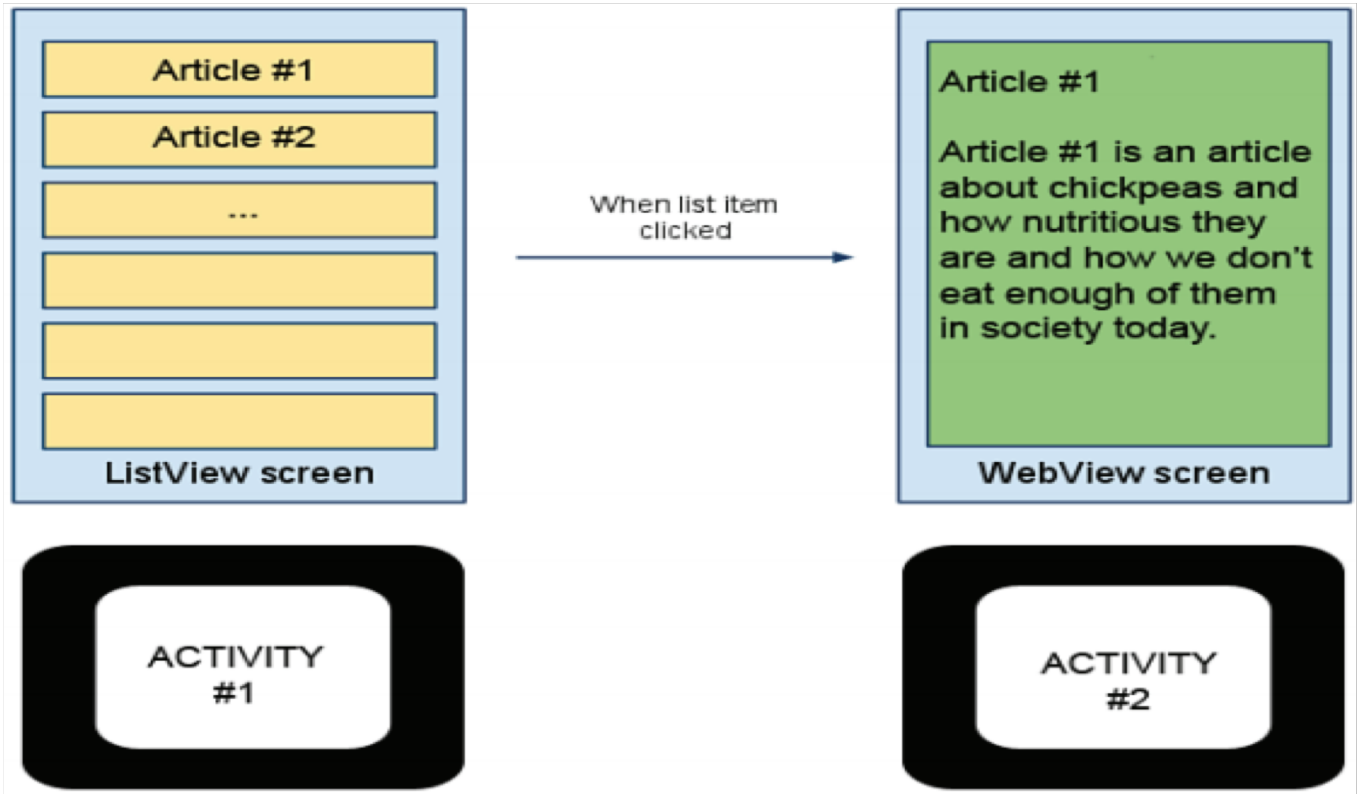
- A Fragment represents a behaviour or a portion of user interface in an Activity.
- The concept of fragment was introduced in Android 3.0 (Honeycomb).
  - Android.support.v4.app
  - Android.app (after API 11)

Why Fragments

- One activity, dynamic UI
  - UI components and behaviors that can be swapped into and out of activity
  - multi-pane user interface on Android,
  - behaves somewhat like a nested activity that can define its own layout and manage its own lifecycle.
- Fragments can be used in multiple activities!

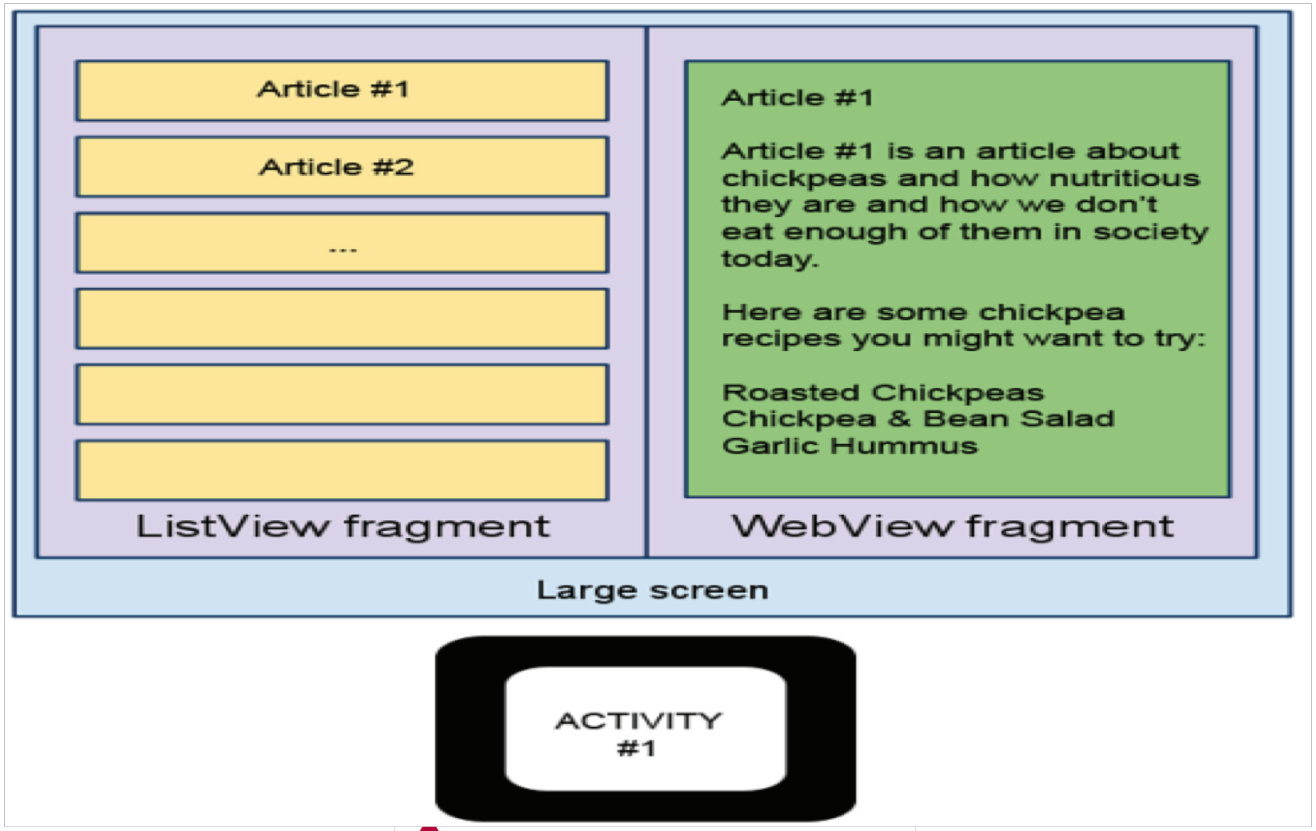
Why Fragments?

- Traditional screen workflow without fragments.



Why Fragments?(2)

- Improved screen workflow with fragments.

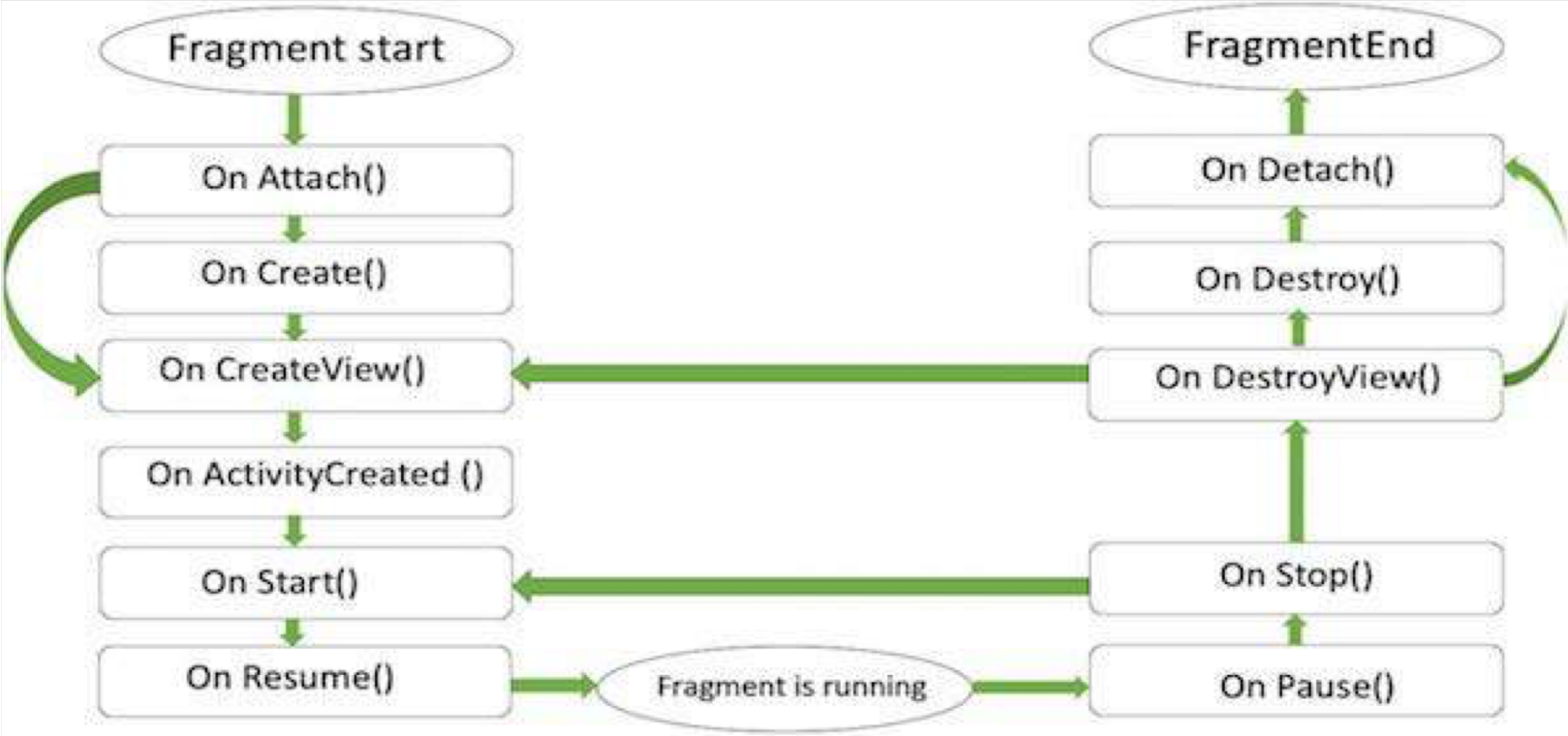


Understanding the `Fragment` Lifecycle

- A `Fragment` must be hosted within an `Activity` class.
- It has its own lifecycle, but it is not a stand-alone component that can exist outside the context of an `Activity`.
- The responsibilities of `Activity` class management are greatly simplified when the entire user interface state is moved off into individual fragments.
- `Activity` classes with only fragments in their layouts no longer need to spend a lot of time saving and restoring their state because the `Activity` object now keeps track of any `Fragment` that is currently attached automatically.
- The `Fragment` components themselves keep track of their own state using their own lifecycle.

Fragment Lifecycle

- A fragment must always be embedded in an activity and the fragment's lifecycle is directly affected by the host activity's lifecycle.



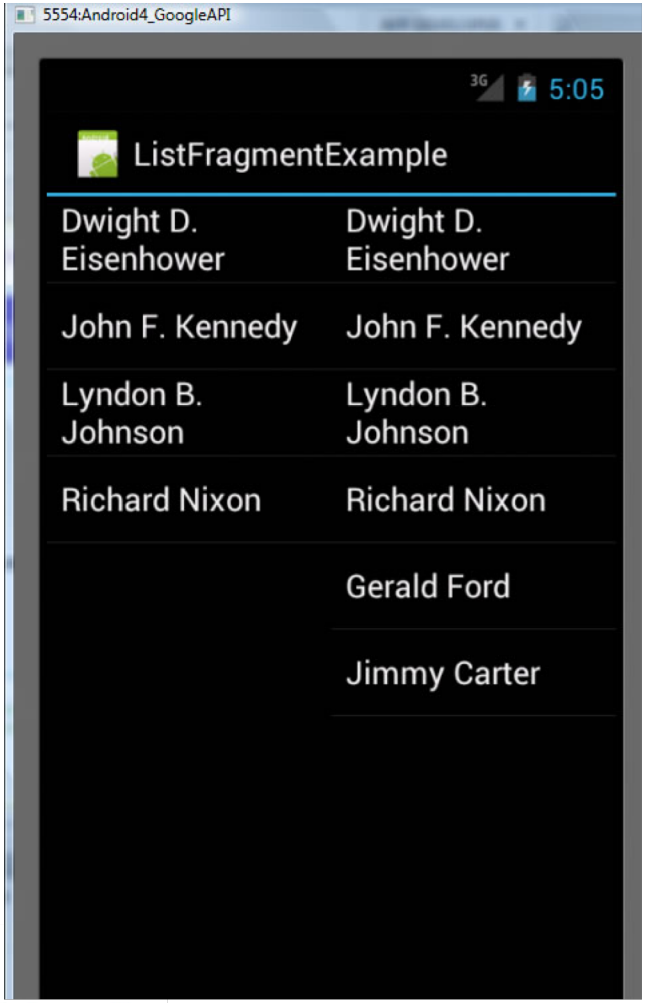
Important Life Cycle CB Methods

- *onCreate():* Android calls this when creating the fragment.
  - initialize essential components of the fragment that you want to retain when the fragment is paused or stopped, then resumed.
- *onCreateView():* Android calls this when it's time for the fragment to draw its user interface for the first time.
  - To draw a UI for your fragment, you must return a *View* from this method that is the root of your fragment's layout.
  - Return null if the fragment does not provide a UI.
- *onPause():*Android calls this method as the first indication that the user is leaving the fragment (though it does not always mean the fragment is being destroyed).
  - you should commit any changes that should be persisted beyond the current user session (because the user might not come back).

Fragment Types

- Single Frame Fragments
  - Used for hand held devices that can show only one fragment view
- ListFragments
  - Fragments having special list view
- DialogFragments
  - This fragment displays a dialog on top of its owner activity
- PreferenceFragments
  - Displays a hierarchy of preference objects as a list, useful when creating a "settings" activity
- WebviewFragments

Fragment Uses



Fragment UI

- To return a layout from *onCreateView():*
  - inflate it from a layout resource defined in XML.
  - *onCreateView()*provides a *LayoutInflater* object.

```
public static class ExampleFragment extends Fragment {
   @Override
   public View onCreateView(LayoutInflater inflater, ViewGroup container,
                            Bundle savedInstanceState) {
      // Inflate the layout for this fragment
      return inflater.inflate(R.layout.example_fragment, container, false);
   }
}
```

- Layout resource named *example_fragment.xml* saved in the application resources.

Adding a Fragment to an Activity (1)

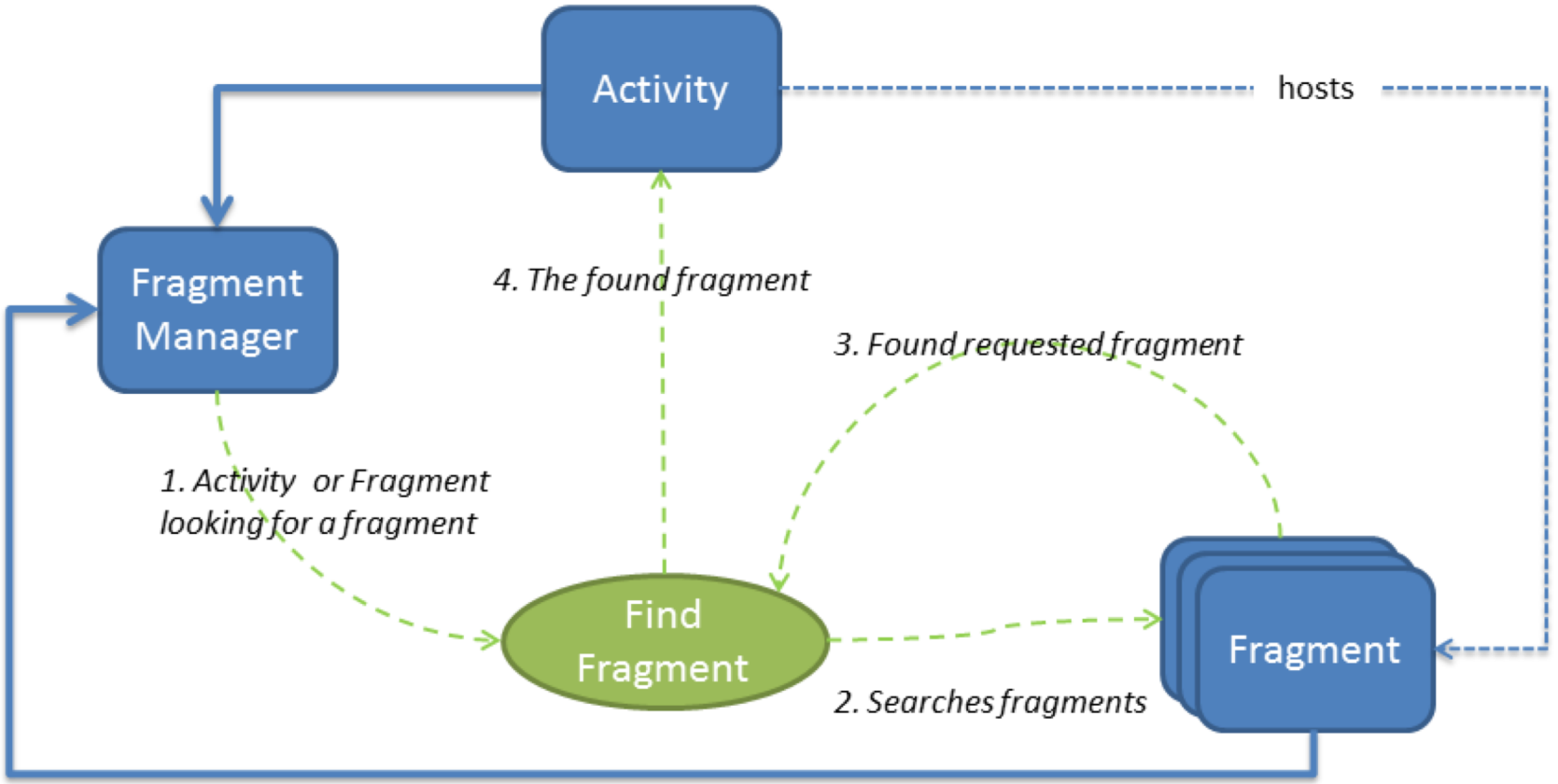
- Declare the fragment inside the activity's layout file

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="horizontal"
   android:layout_width="match_parent"
   android:layout_height="match_parent">
   <fragment android:name="com.example.news.ArticleListFragment"
         android:id="@+id/list"
         android:layout_weight="1"
         android:layout_width="0dp"
         android:layout_height="match_parent" />
   <fragment android:name="com.example.news.ArticleReaderFragment"
         android:id="@+id/viewer"
         android:layout_weight="2"
         android:layout_width="0dp"
         android:layout_height="match_parent" />
</LinearLayout>
```

Implementing Fragments

- Coordination between an Activity and its Fragment components is facilitated by the FragmentManager:
  - android.app.FragmentManager
- The FragmentManager is acquired from the getFragmentManager() method, which is available within the Activity and Fragment classes.
- When using the support library, coordination is facilitated by the FragmentManager from android.support.v4.app and is acquired from the getSupportFragmentManager() method of the FragmentActivity support APIs.

Fragment and Activity



Adding a Fragment to an Activity (2)

- Programatically, add the fragment:

```
FragmentManager fragmentManager = getSupportFragmentManager();
FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();

ExampleFragment fragment = new ExampleFragment();
fragmentTransaction.add(R.id.fragment_container, fragment);
fragmentTransaction.commit();
```

Managing Fragments

- Main fragment manager is: FragmentManager
- Use getSupportFragmentManager() from your activity
  - Get fragments that exist in the activity, with findFragmentById() (for fragments that provide a UI in the activity layout) or findFragmentByTag() (for fragments that do or don't provide a UI).
  - Pop fragments off the back stack, with popBackStack() (simulating a *Back* command by the user).
  - Register a listener for changes to the back stack, with addOnBackStackChangedListener().
- Open a FragmentTransaction, to perform transactions, such as add, remove, and commit fragments

Fragment Manager Example

```
// Create new fragment and transaction

Fragment     newFragment = new ExampleFragment();
FragmentTransaction     transaction =
                                getSupportFragmentManager().beginTransaction();

// Replace whatever is in the fragment_container view with this fragment,
// and add the transaction to the back stack

transaction.replace(R.id.fragment_container, newFragment);
transaction.addToBackStack(null);

// Commit the transaction

transaction.commit();
```

FragmentTransacation

- The order in which you add changes to a FragmentTransaction doesn't matter, except:
  - You must call commit() last
  - If you're adding multiple fragments to the same container, then the order in which you add them determines the order they appear in the view hierarchy
- Note: if there is no addToBackStack() when you perform a transaction that removes a fragment, then that fragment is destroyed after *commit*
  - if there is one, then it is merely stopped

Additional Ways to Use Fragments

- Fragments are great for creating reusable interface components.
- There are other ways to use fragments within your application.
  - You can create reusable behavior components without a user interface.
  - You can nest fragments within fragments.

Behavior Fragments without a User Interface

- Fragments are
  - decoupling a user interface component from an `Activity`.
  - decouple application behaviors, such as background processing, into a reusable `Fragment`.

```
Public class NonUIFragment extends Fragment{
        @Override
        public void onCreate (Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setRetainInstance(true); // Must be set to true}
        @Override
        Public View onCreateView(...) {
                return null;       // You need to return null}
}
```

Exploring Nested Fragments

- An addition to Android 4.2 (API Level 17) is the ability to nest fragments within fragments.

- Nested fragments have also been added to the Android Support Library, making this API capability available all the way back to Android 1.6 (API Level 4).

Exploring Nested Fragments (2)

- Add a `Fragment` within another `Fragment`
  - Invoke the `Fragment` method `getChildFragmentManager()`, which returns a `FragmentManager`.
- Start a `FragmentTransaction`
  - Call `beginTransaction()` and then invoking the `add()` method, including the `Fragment` to add and its layout, followed by the `commit()` method.

```
FragmentC fragmentC = new FragmentC();
FragmentTransaction transaction = getChildFragmentManager().beginTransaction();
transaction.add(R.id.child_fragment, fragmentC );
transaction.commit();
```

Exploring Nested Fragments (3)

- Nested fragments bring many possibilities for creating dynamic and reusable nested components.
- Some examples include:
  - Tabbed fragments within tabbed fragments
  - Paging from one `Fragment` item/`Fragment` detail screen to a next `Fragment` item/`Fragment` detail screen with `ViewPager`
  - Paging fragments with `ViewPager` within tabbed fragments

http://www.androidbegin.com/tutorial/android-actionbarsherlock-nested-fragment-tabs-tutorial/