What is a Fragment?

A **Fragment** defines an isolated, **modular piece** of a larger **Activity** UI.

- Fragments are always **embedded** in an **activity**.
- Has a **separate** layout **XML** file defining its UI.
- Can be **dynamically** or **statically** added to an activity.
- Fragments have their own **lifecycle** events.

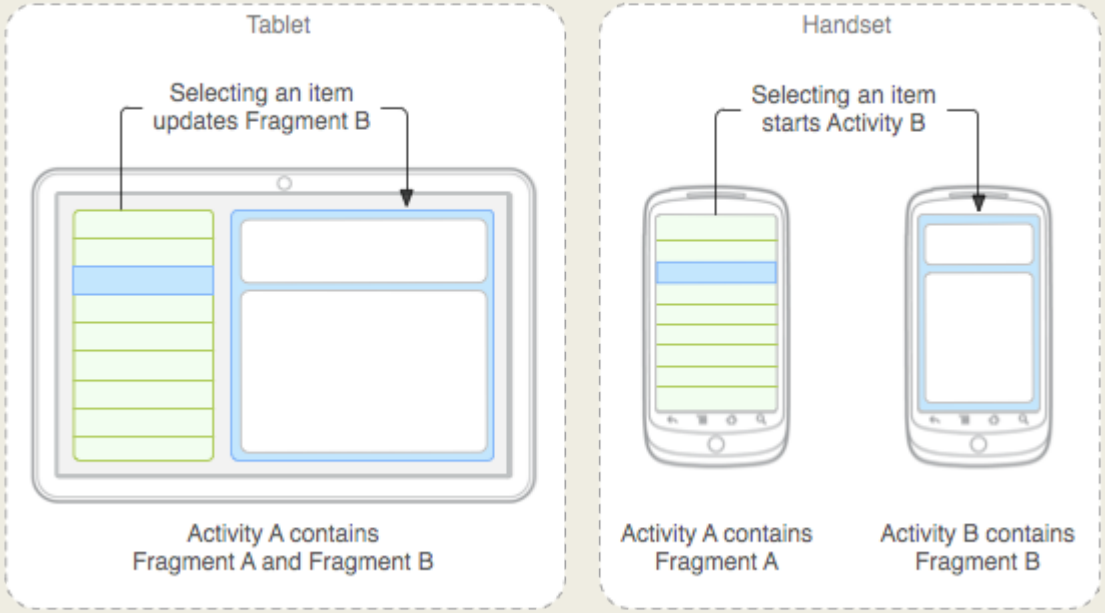
Why Fragments?

A **Fragment** defines an isolated, **modular piece** of a larger **Activity** UI.

- Encourages modular architecture for activities.
- Encapsulates functionality so that it is easier to reuse.
- Helps adapt to a variety of devices and screen sizes.
- Different layouts for landscape and portrait modes.
- Different UIs can reuse the same modular elements.

Why Fragments?

A **Fragment** enables powerful support for different orientations and screen-sizes.



Mobile can have two separate screens and tablets can combine them together.

Fragments Compatibility

Fragments are **broadly supported** by nearly **all** Android SDK **versions**.

- While Fragments were **introduced** in version **3.0**, **compatibility support** has been provided to all active versions of Android today.
- When using Fragments, we have to be **careful** to use the **compatibility libraries** for Gingerbread and lower versions to **work properly**.
- Fortunately, these libraries are **available** in every **generated** Android project.

Defining Fragments

Fragments are **defined** much **like** an **Activity**.

- Fragments have **XML layout files** that belong in `res/layouts/fragment_foo.xml`
- Fragments have **Java source files** much like an Activity as well e.g `FooFragment`.
- Every Fragment **extends** the `android.support.v4.app.Fragment` class or a descendant.
- Fragment view inflation happens on the `onCreateView` method.

Defining Fragments, Java

Fragments are **defined** much **like** an **Activity**.

- Notice the use of the **support** Fragment.
- **onCreateView** is used to inflate the XML view
- parent is the view the fragment is **embedded** within.

```
import android.support.v4.app.Fragment;

public class TweetListFragment extends Fragment {
     @Override
     public View onCreateView(LayoutInflater inf, ViewGroup parent,
          Bundle savedInstanceState) {
          // Inflate the layout for this fragment
          return inf.inflate(R.layout.article_view, parent, false);
     }
}
```

Defining Fragments, XML

Fragments are **defined** much **like** an **Activity**.

- **XML Layout** is defined the **same** as before
- Free to use all the same views and layouts

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <ListView
        android:id="@+id/lvTweets"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true" >
    </ListView>
</RelativeLayout>
```

Fragment Exercises, 1

- Open **Project** in Eclipse
- Create a Fragment XML Layout called fragment_demo.xml
- Add an EditText and a Button
- Create a Fragment Java file called DemoFragment.java extending from Fragment

Loading Fragments

Fragments must be **added** to a host **Activity**.

- Using a fragment involves **loading the fragment** as a view **into an Activity**.
- Fragments can be **embedded** either (*statically*) in an Activity **XML Layout** or (*dynamically*) in the Activity **Java code** at runtime.
- Fragments are **modular view components** that be included into an Activity just like any other view.

Loading Fragments, XML

Fragments must be **added** to a host **Activity**.

- Fragments can be **embedded** within any **Activity XML** Layout just like any other view.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

  <fragment
    android:id="@+id/fragment_tweet_list"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    class="com.example.apps.TweetListFragment" >
  </fragment>
</RelativeLayout>
```

Fragment Exercises, 2

- Open **Project** in Eclipse
- Switch an Activity to extend from FragmentActivity
- Add Fragment from Ex. 1 statically to the Activity XML Layout
- Activity should now display the Fragment View's input and button.

Loading Fragments, Java

Fragments can be **added** at runtime using a **FragmentTransaction**.

- Loading fragments **dynamically** happens in the **Activity Java source** at runtime.
- First step is to **access** the `FragmentManager` for an Activity using `getFragmentManager()`
- Next step is to start a `FragmentTransaction` which allows us to **modify** the fragments at **runtime**.
- Finally, the transaction is **committed** and the changes are reflected within the **Activity**.

Loading Fragments, Java

Fragments can be **added** at runtime using a **FragmentTransaction**.

- Within the Activity, first **setup a container** in the XML **layout** where the Fragment will be placed.
- **FrameLayout** is a **great placeholder** to be **replaced** at runtime with a fragment.

```
<FrameLayout
        android:id="@+id/flContainer"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true" >
</FrameLayout>
```

Loading Fragments, Java

Fragments can be **added** at runtime using a **FragmentTransaction**.

```
public class MainActivity extends FragmentActivity {
  protected void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.activity_main);
     // Only if this is a new activity, so fragments don't exist yet.
     if (savedInstanceState == null) {
           FragmentTransaction ft =
              getSupportFragmentManager().beginTransaction();
           ft.replace(R.id.flContainer, new DemoFragment());
           ft.commit();
     }
  }
}
```

Fragment Exercises, 3

- Switch Activity to extend from FragmentActivity
- Add FrameLayout to the Activity XML Layout
- Add Fragment from Ex. 1 dynamically to the Activity Java using FragmentManager
- Activity should now display the Fragment View's input and button

Loading Static vs Dynamic

Fragments can be added **statically** or **dynamically**.

- **Statically** added within the Activity XML Layout should be used when the fragment will **always be present**.
- **Statically** added Fragments **cannot be removed** from the Activity at runtime.
- **Dynamically** add fragments if you will need to **add**, replace, hide or **remove** the fragment views at **runtime**.

Referencing Fragments

Fragments can be **referenced** at runtime using the **FragmentManager**.

- To reference a Fragment at runtime, use the `FragmentManager` to `findFragmentById` within the activity.

```
TweetListFragment fragmentTweetList = (TweetListFragment)
    getSupportFragmentManager().findFragmentById(R.id.fragment_tweet_list);
 if (fragment == null || ! fragment.isInLayout()) {
   // fragment doesn't exist in activity
 } else {
   // fragment does exist
 }
```

Fragment Lifecycle

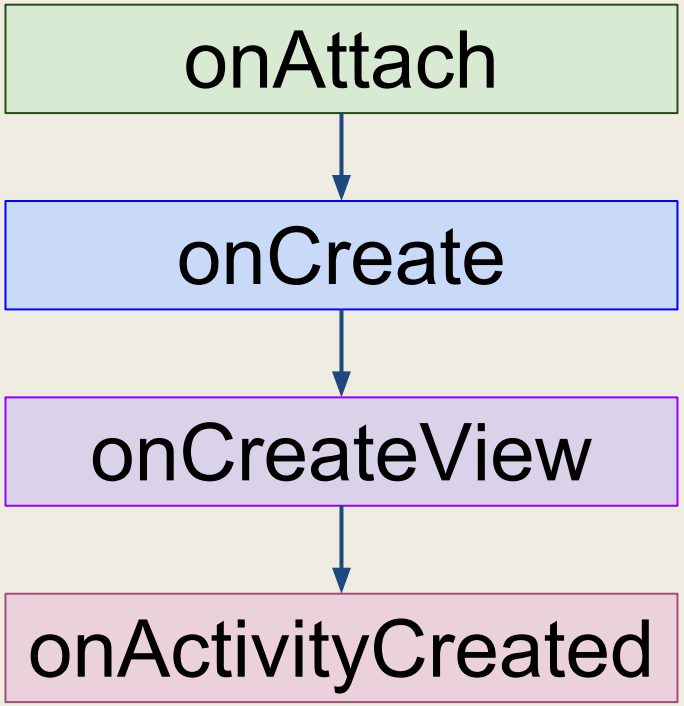
Fragments have their own **independent lifecycle** events similar to an Activity.

- **onAttach** - Fragment is attached to an activity
- **onCreate** - Fragment is initialized
- **onCreateView** - Fragment's view is inflated
- **onDestroyView** - Fragment is being destroyed
- **onActivityCreated** - Activity has finished onCreate
- **onStart** - Fragment is visible on screen

...and all the other existing lifecycle methods.

Fragment Lifecycle

Fragments have their own **independent lifecycle** events similar to an Activity.



Fragment instance is associated with an Activity instance (activity is not fully created yet)

Fragment instance is being created, or re-created. Standard fragment initialization step.

Fragment instance should now inflate the View object hierarchy it contains.

Activity containing the Fragment instance has been created, and the View objects contained by the Fragment have been created.

Fragment Lifecycle, cont'd

Fragments have their own **independent lifecycle** events similar to an Activity.

```
public class TweetListFragment extends Fragment {
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        Log.d("DEBUG", "Time to inflate the view");
        return inflater.inflate(R.layout.fragment_demo, container, false);
    }
     public void onAttach(Activity activity) {
          super.onAttach(activity);
          Log.d("DEBUG", "onAttach fired");
     }
     public void onActivityCreated(Bundle state) {
          super.onActivityCreated(state);
          // get the activity with getActivity()
          Log.d("DEBUG", "OnActivityCreated fired");
     }
}
```

Referencing the Activity

Fragments can **reference** their activity context at runtime to manage views.

- To reference the host activity context, use the `getActivity` method from within the `onActivityCreated` method.

```
public class TweetListFragment extends Fragment {
  @Override
  public void onActivityCreated(Bundle savedInstanceState) {
     super.onActivityCreated(savedInstanceState);
     ListView lvTweets = (ListView) getActivity().findViewById(R.id.lvTweets);
  }
}
```

Fragment Modularity

Fragments are **encapsulated modules** and should be **reusable**.

- Fragments **should not** directly **communicate** with **other fragments**.
- Fragments should **communicate** with their host **activity** using only a predefined **interface**.
- **Activity** is usually **responsible** for managing intents and other event callbacks.
- Always try to **avoid coupling** a fragment to a **particular** Activity.

Fragment Modularity, cont'd

```
public class SampleFragment extends Fragment {
  private OnSomeListener listener;
  public interface OnSomeListener {
    public void onSomeCustomEvent(String text);
  }

  public void onAttach(Activity activity) {
    super.onAttach(activity);
    if (activity instanceof OnSomeListener) {
      listener = (OnSomeListener) activity;
    } else {
      throw new ClassCastException("Must implement OnSomeListener interface");
    }
  }

  public void onDetach() {
    super.onDetach();
    listener = null;
  }
}
```

Fragment Modularity, cont'd

```
public class SomeActivity extends FragmentActivity implements OnSomeListener {
     SampleFragment fragmentSample;
     protected void onCreate(Bundle savedInstanceState) {
       fragmentSample =
          getFragmentManager().findFragmentById(R.id.fragment_sample);
     }

     @Override
     public void onSomeCustomEvent(String text) {
        Log.d("DEBUG", "onSomeCustomEvent " + text);
     }
}
```

```
public class SampleFragment extends Fragment {
   // ...
   // Using the listener based on the interface
   public void onClick(View v) {
      listener.onSomeCustomEvent("foo");
   }
}
```

Fragment Exercises, 4

- Activity should display the Fragment View's input and button from Exercise 3
- Add TextView to top of the activity above the Fragment
- When the Button is pressed in the Fragment, display the text from the Fragment's EditText in the Activity TextView.

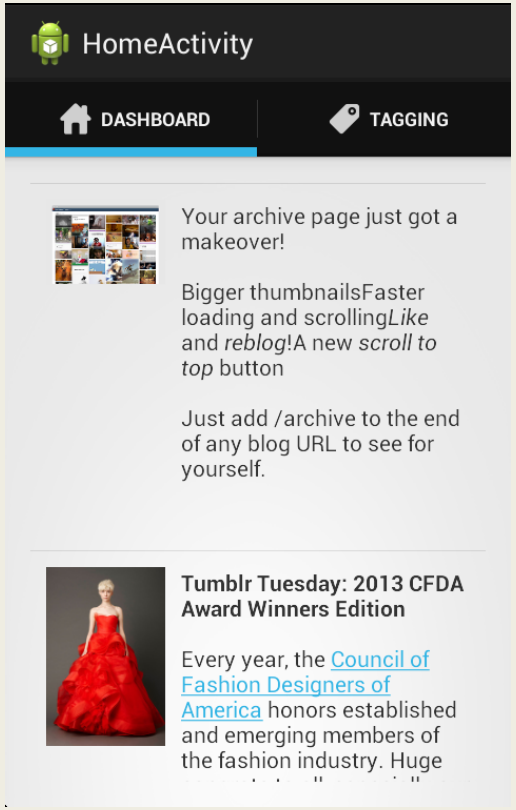
Fragments and Tabs

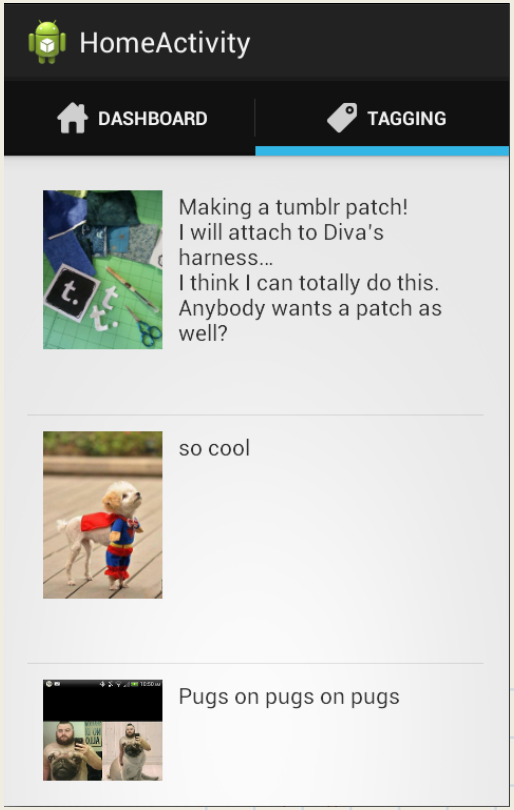
Fragments can be used to create a **tabbed** interface for different content. There are several ways to create tabbed fragment-based UIs:

- **ActionBar Tabs** - The ActionBar supports adding a tabbed interface
- **ViewPager** - A view that uses an adapter like a ListView which shows only a single subview at a time.
- **FragmentTabHost** - Simple interface for creating tabbed content based on Fragments.

Fragments and Tabs

The **ActionBar** can contain **tabs** which allow users to switch between fragments.





Fragments and Tabs

The **ActionBar** can contain **tabs** which allow users to switch between fragments.

```
public class HomeActivity extends FragmentActivity {
  // ...

  public void onCreate(Bundle savedInstanceState) {
    // ...
    ActionBar actionBar = getActionBar();
    actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);
    Tab tab1 = actionBar.newTab()
                    .setText("First Tab")
                    .setTabListener(new SampleFragment())
                    .setIcon(R.drawable.ic_first_tab);
    actionBar.addTab(tab1);
    // ... more tabs
  }
}
```

Fragments and Tabs

The **ActionBar** can contain **tabs** which allow users to switch between fragments.

```
private class SampleFragment extends Fragment
  implements TabListener {

  public void onTabSelected(Tab tab, FragmentTransaction ft) {
    FragmentTransaction fts =
           getSupportFragmentManager().beginTransaction();
    fts.replace(R.id.framelayout, new SampleFragment());
    fts.commit();
    // ...
  }

}
```

Fragment Exercises, 5

- Create **two fragments** with different layouts.
- Create an activity (extends FragmentActivity) with a FrameLayout and use the ActionBar to **add two tabs** named after the fragments.
- **Clicking** each **tab** should **load** the correct **fragment** into view.

Fragments and ActionBar

Fragments can **append items** to the **ActionBar** within their host activity.

- Simply call `setHasOptionsMenu` and inflate the menu

```
public class SampleFragment extends Fragment {
  public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setHasOptionsMenu(true);
  }

  public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
      inflater.inflate(R.menu.fragment_sample, menu);
  }
}
```

Specialized Fragments

Several specialized Fragments exist for particular use cases.

- **DialogFragment** - Dialog windows (overlayed over an Activity) can be extended from this.
- **WebViewFragment** - Embeddable WebView within an Activity.
- **ListFragment** - Embeddable ListView-only Fragment for displaying simple collections of items.
- **PreferenceFragment** - Fragment that is used to modify and view SharedPreferences.

Flexible Fragment Interfaces

Fragments enable **flexible user interfaces** for different screen sizes.

- Common pattern is to use fragments in **different places** for **different devices**.
- **Smaller** screens have fragments broken up **across multiple** activities.
- **Larger** screens combine the fragments into a **wider** user interface.
- This approach allows for **reusing** the same fragment **views** in different contexts and flows.

Flexible Fragment Interfaces

Fragments enable **flexible user interfaces** for different screen sizes.

- You can use `res/layout-[descriptor]` (i.e `layout-large`) to change a layout based on the screen size.
- Check if a fragment exists in a layout, if it does then load the content, otherwise load the content into another activity.

Fragments Wrap-up

- A **Fragment** is a part of a larger activity UI.
- Supports **modular** and **flexible** interfaces
- **Compatible** with most SDK versions
- Defined similar to an activity with **XML** and **Java**
- **Loaded** into an **Activity** via Java or XML
- **Manages** their own **lifecycle** events
- Should use an **interface** to **communicate**
- Fragments can be used with a **tabbed UI**