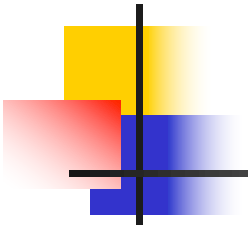
Android Fragments

Includes some fragments of Chapter 9

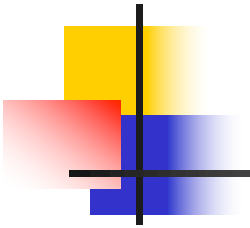


Overview

- Understand what fragments are and how to use them
- Learn how to use a ListFragment
- Introduce master-detail flow
- Learn how to create alternative layouts

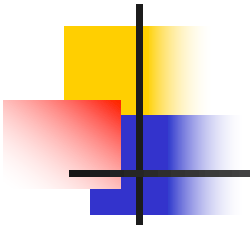
Understanding Fragments

- By incorporating Fragment components into your user interface design, you write one application that can be tailored to different screen characteristics and orientations instead of different applications tailored to different types of devices.
- This greatly improves code reuse, simplifies application testing needs, and makes publication and application package management much less cumbersome.



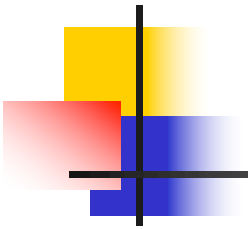
Understanding Fragments

- The basic rule of thumb for developing Android applications used to be to have one Activity per screen of an application.
- This ties the underlying “task” functionality of an Activity class very directly to the user interface.
- However, as bigger devices are now available, this technique faces some issues.



Understanding Fragments

- Without fragments, when you have more room on a single screen to do more, you must implement separate Activity classes, with very similar functionality, to handle the cases where you want to provide more functionality on a given (larger) screen.
- Fragments help manage this problem by encapsulating screen functionality into reusable components that can be mixed and matched within Activity classes.



Understanding Fragments

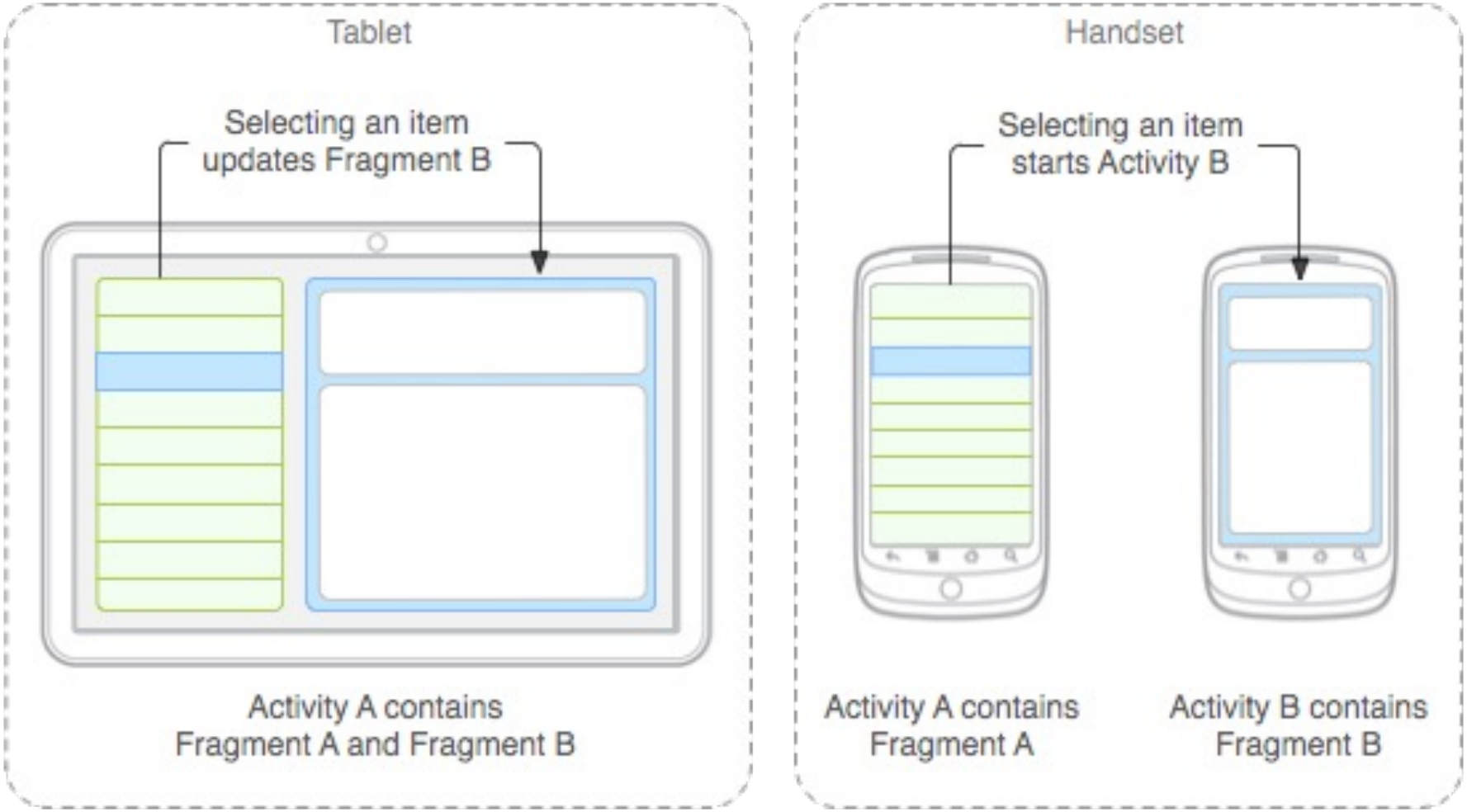
According to Android documentation:

"A Fragment represents a behavior or a portion of user interface in a FragmentActivity."

"You can think of a fragment as a modular section of an activity, which has its own lifecycle, receives its own input events, and which you can add or remove while the activity is running (sort of like a "sub activity" that you can reuse in different activities)."

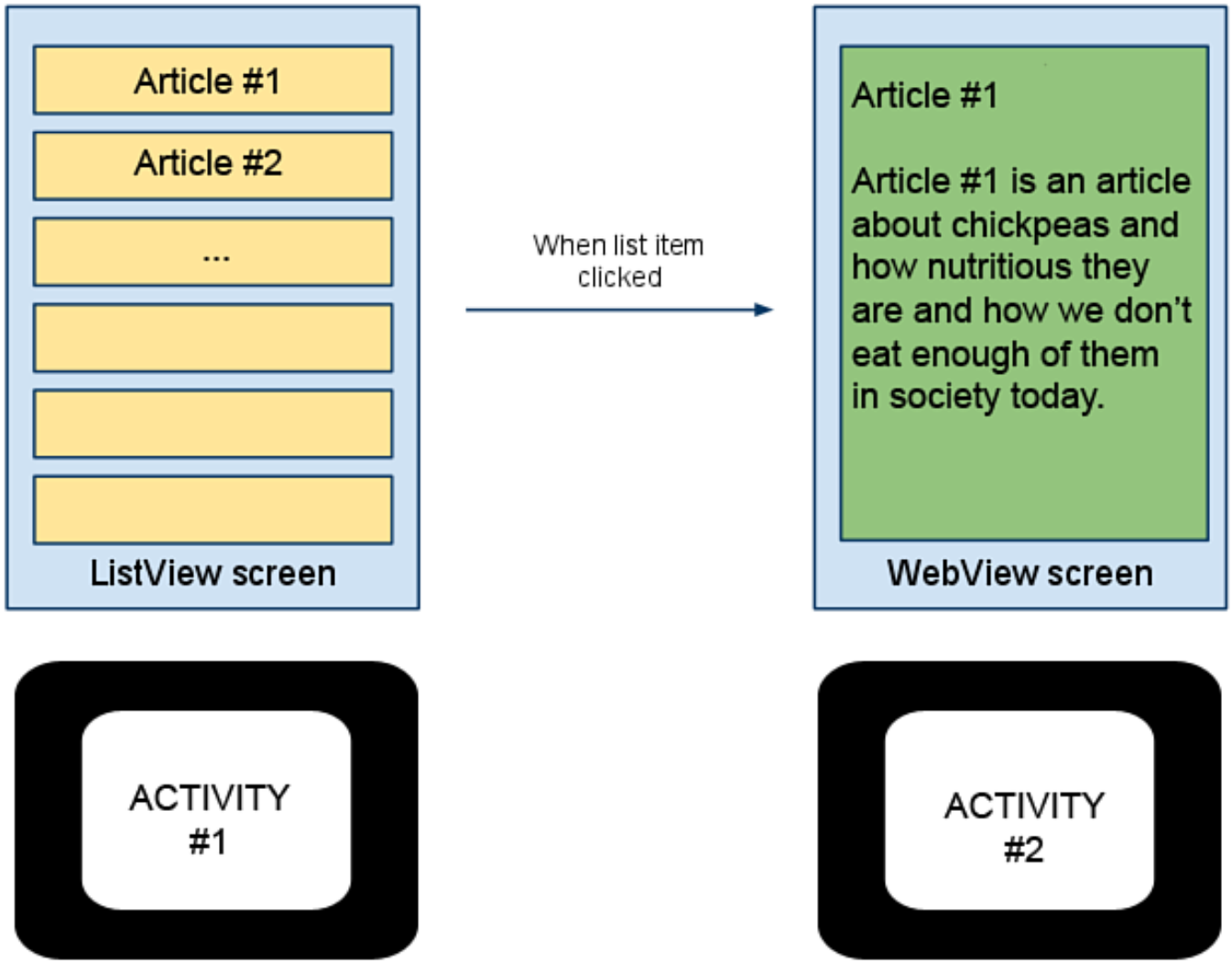
https://developer.android.com/guide/components/fragments

Understanding Fragments

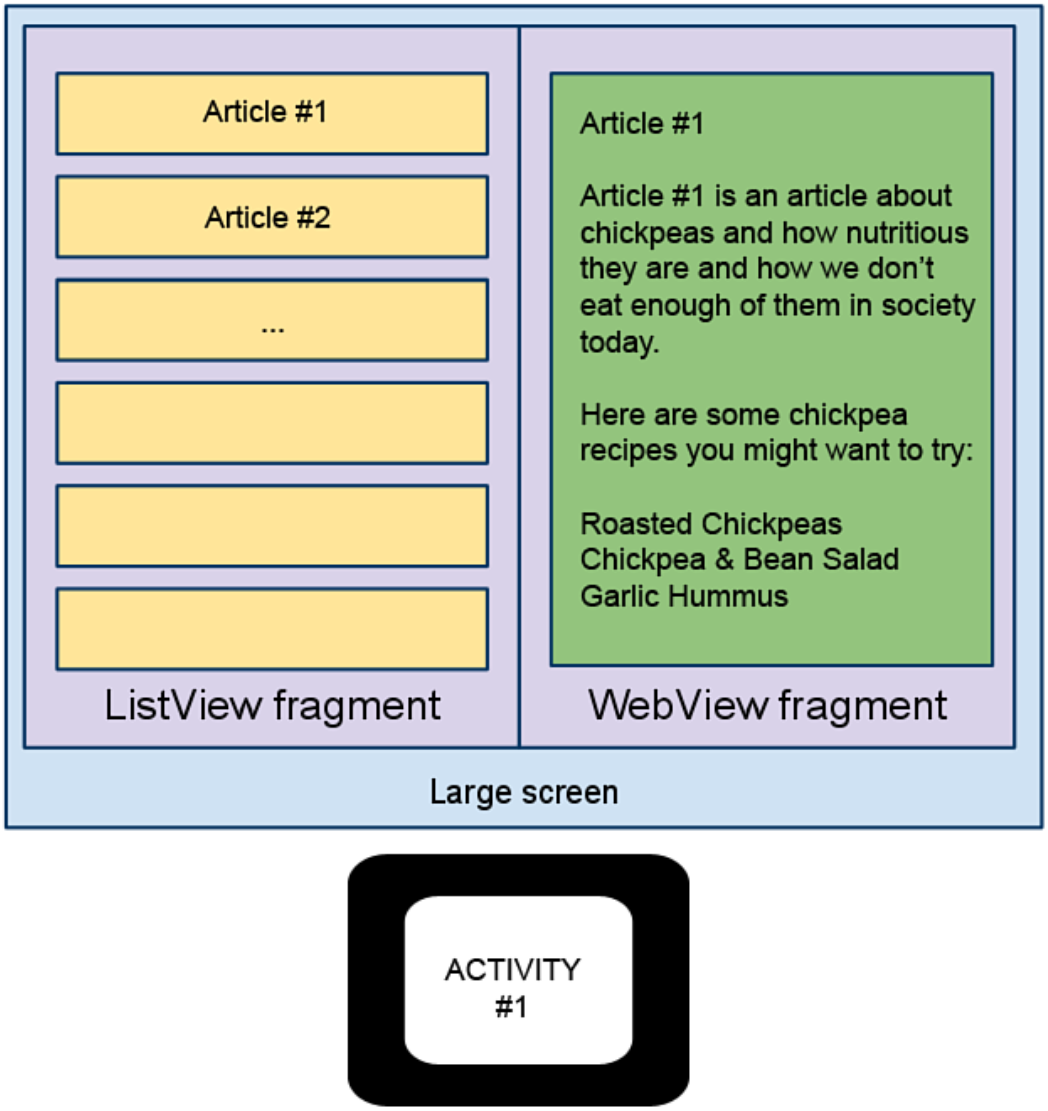


https://developer.android.com/guide/components/fragments

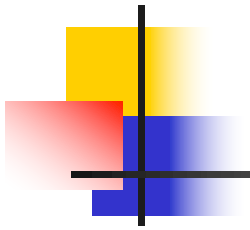
Understanding Fragments



Understanding Fragments

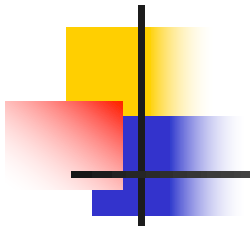


Understanding the Fragment Lifecycle



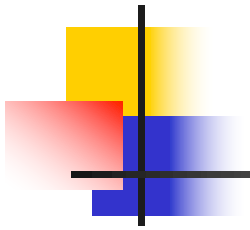
- A Fragment must be hosted within an Activity class.
- A Fragment has its own lifecycle, but it is not a stand-alone component that can exist outside the context of an Activity.
- The responsibilities of Activity class management are greatly simplified when the entire user interface state is moved off into individual fragments.

Understanding the Fragment Lifecycle



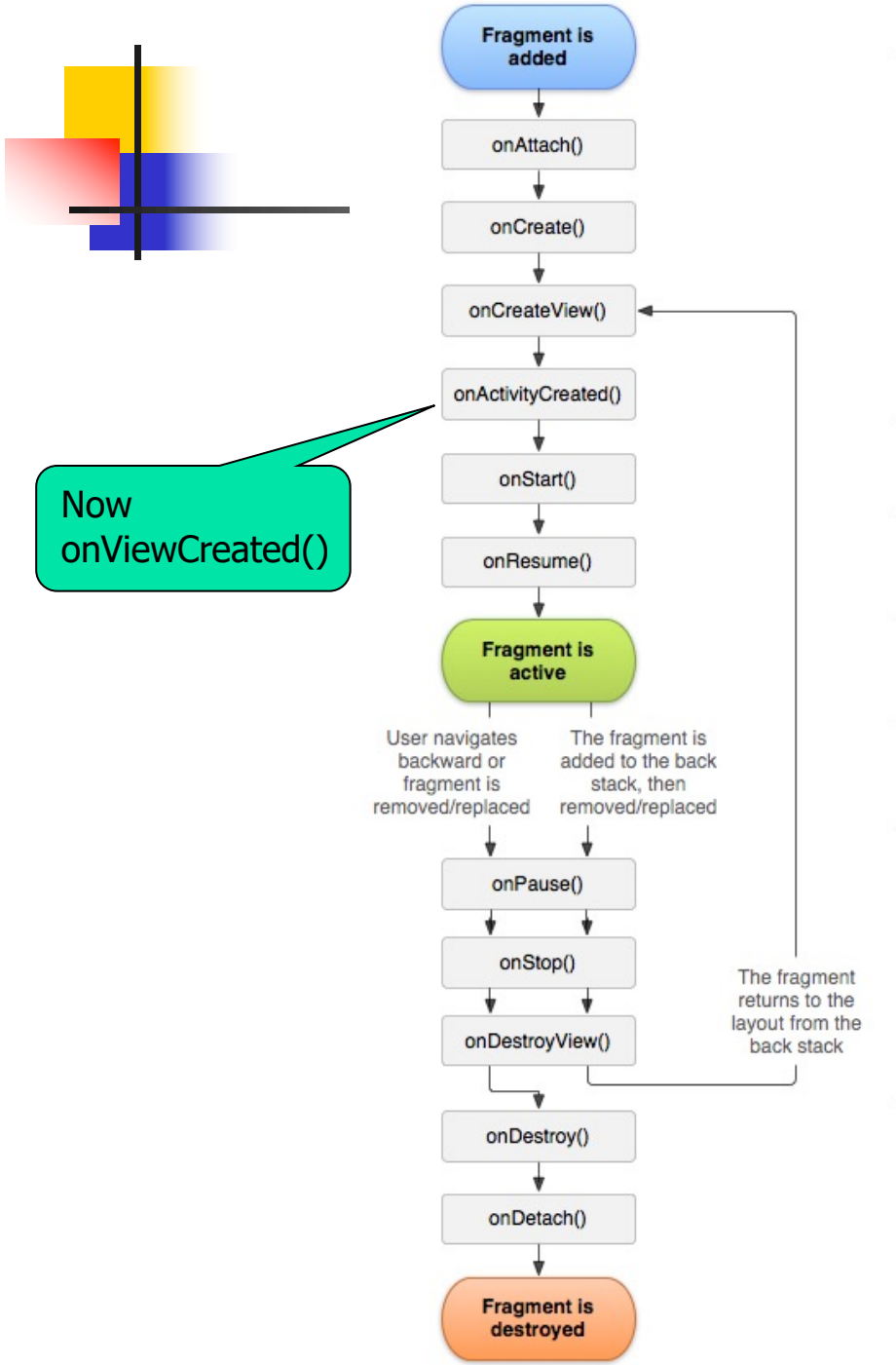
- Activity classes with only fragments in their layouts no longer need to spend a lot of time saving and restoring their state because the Activity object now keeps track of any Fragment that is currently attached automatically.

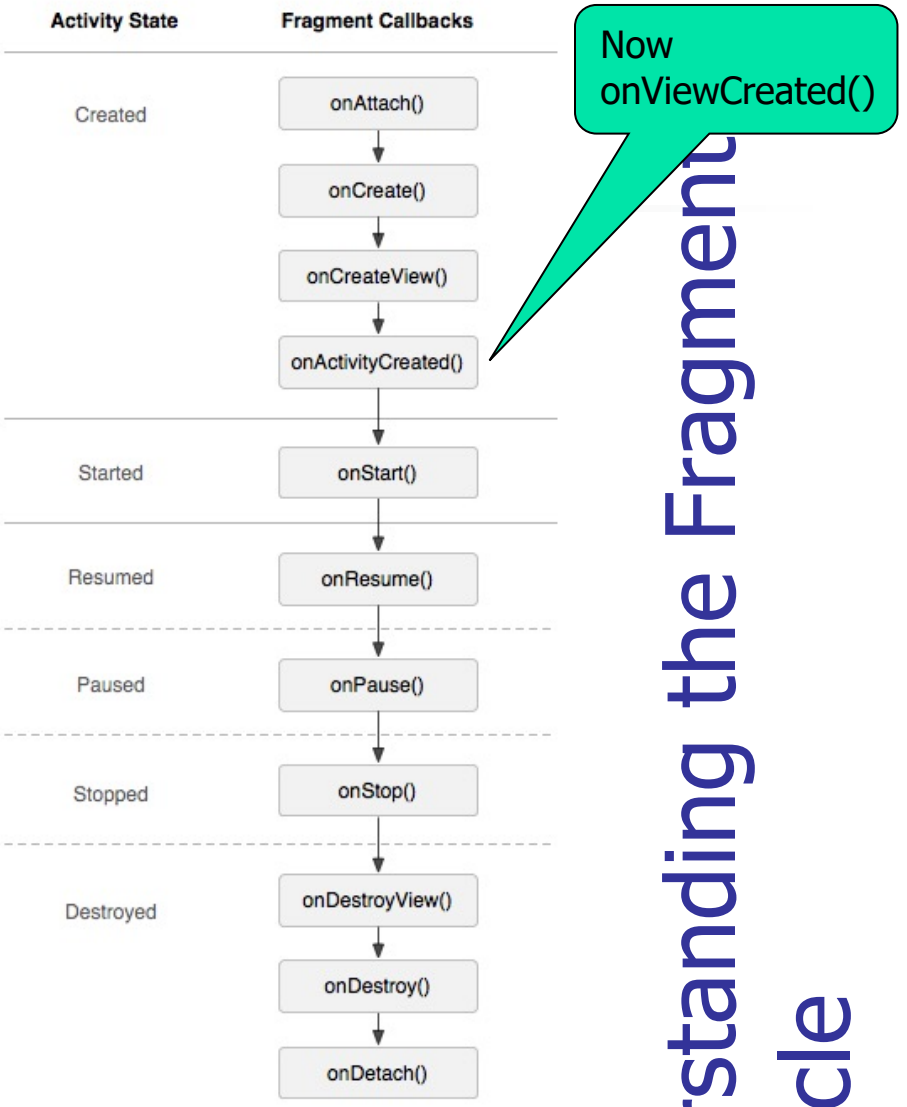
Understanding the Fragment Lifecycle



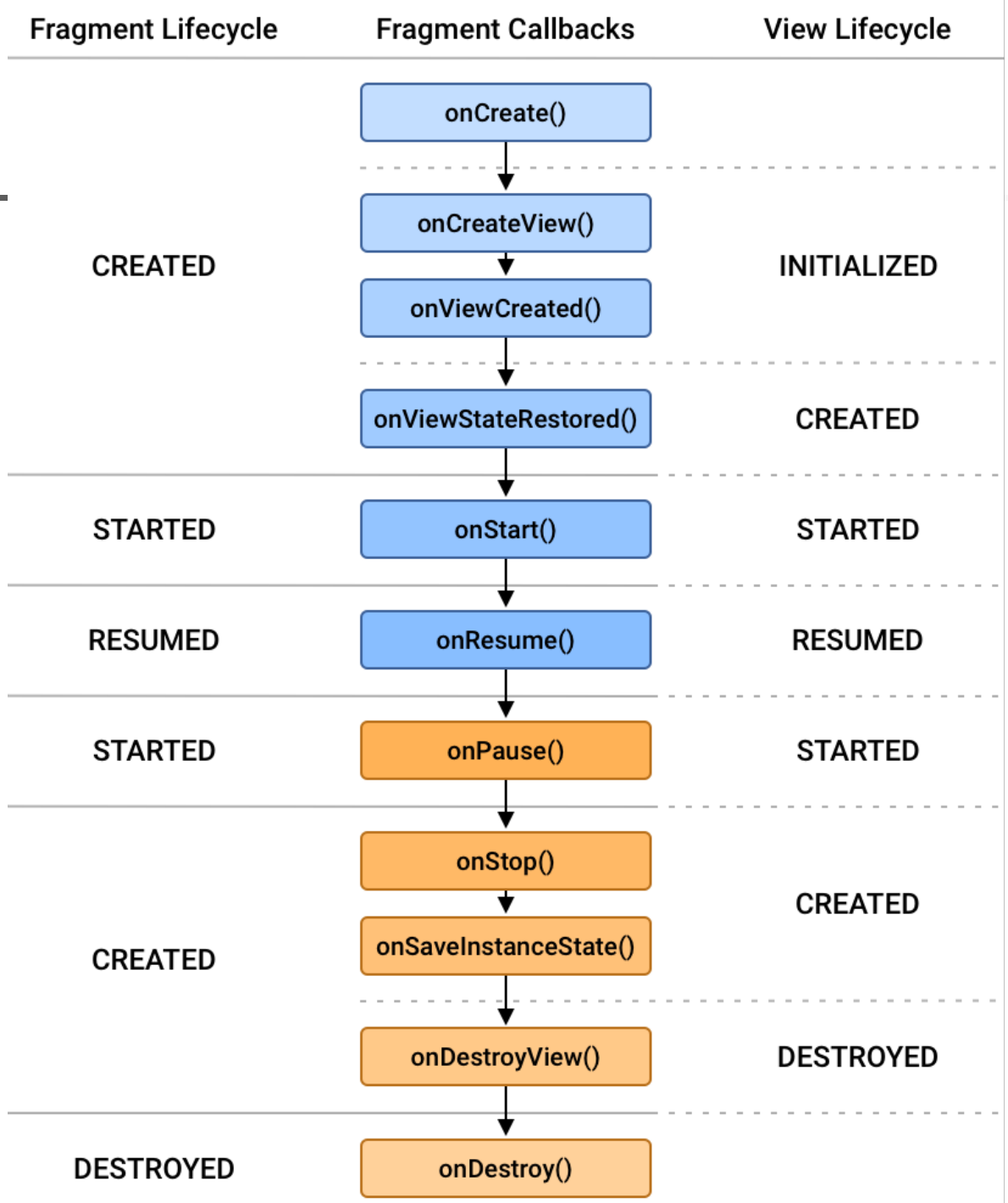
- The Fragment components themselves keep track of their own state using their own lifecycle.
- You can mix fragments with other Views directly in an Activity class.
- The Activity class will be responsible for managing the View controls, as normal.

Understanding the Fragment Lifecycle

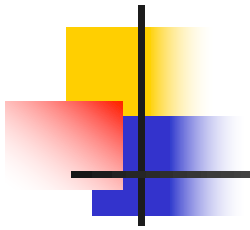




Understanding the Fragment Lifecycle in androidx



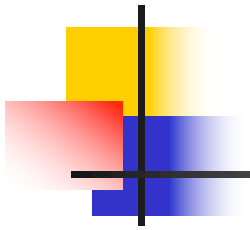
Understanding the Fragment Lifecycle



- The Activity must also focus on managing its Fragment classes.
- Coordination between an Activity and its Fragment components is facilitated by the `FragmentManager`:

```
androidx.fragment.app.FragmentManager
```

Understanding the Fragment Lifecycle



- The `FragmentManager` is acquired from the `getSupoportFragmentManager()` method, which is available within the Activity and Fragment classes.

Defining Fragments

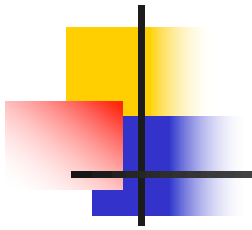
- Fragment implementations that have been defined as regular classes within your application can be added to any layout resource files by using the `<fragment>*` XML tag and then loaded into your Activity using the standard `setContentView()` method, which is normally called in the `onCreate()` method of your Activity.

* Now, androidx.fragment.app.FragmentContainerView is a proper choice

Defining Fragments (Cont'd)

- An example of a layout with a FragmentContainerView:

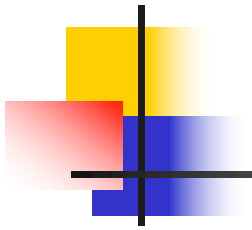
```
<androidx.fragment.app.FragmentContainerView
    android:id="@+id/fragment"
    android:name="edu.uga.cs.ugafragments.ChoiceFragment"
    class="edu.uga.cs.ugafragments.ChoiceFragment”
    …
/>
```



Managing Fragment Modifications

- As you can see, when you have multiple Fragment components on a single screen, within a single Activity, you often have user interaction on one Fragment (such as our ListView Fragment) causing the Activity to update another Fragment.
- In general, we will use FrameLayouts to hold additional fragments.

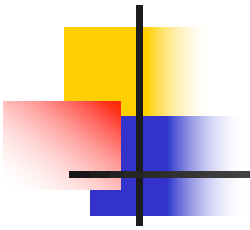
Managing Fragment Modifications



- An update or modification to a Fragment is performed using a `FragmentTransaction`:

  `androidx.fragment.app.FragmentTransaction`

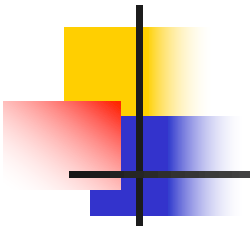
- A number of different actions can be applied to a Fragment using a `FragmentTransaction` operation:
  - A Fragment can be attached or reattached to the parent Activity.
  - A Fragment can be hidden and unhidden from view.



Managing Fragment Modifications

Handling of the Back button within a fragment-based user interface design

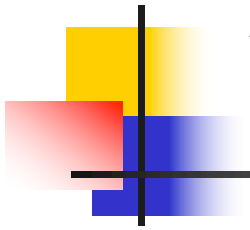
- We know the parent Activity class has its own back stack.
- As the developer, you can decide which `FragmentTransaction` operations are worth storing in the back stack and which are not, by using the `addToBackStack()` method of the `FragmentTransaction` object.



Managing Fragment Modifications

- In our news application example, we might want each of the articles displayed in the WebView Fragment to be added to the parent Activity class's back stack so that if the user hits the Back button, he or she traverses the articles already read before backing out of the Activity entirely.

Attaching and Detaching Fragments with Activities

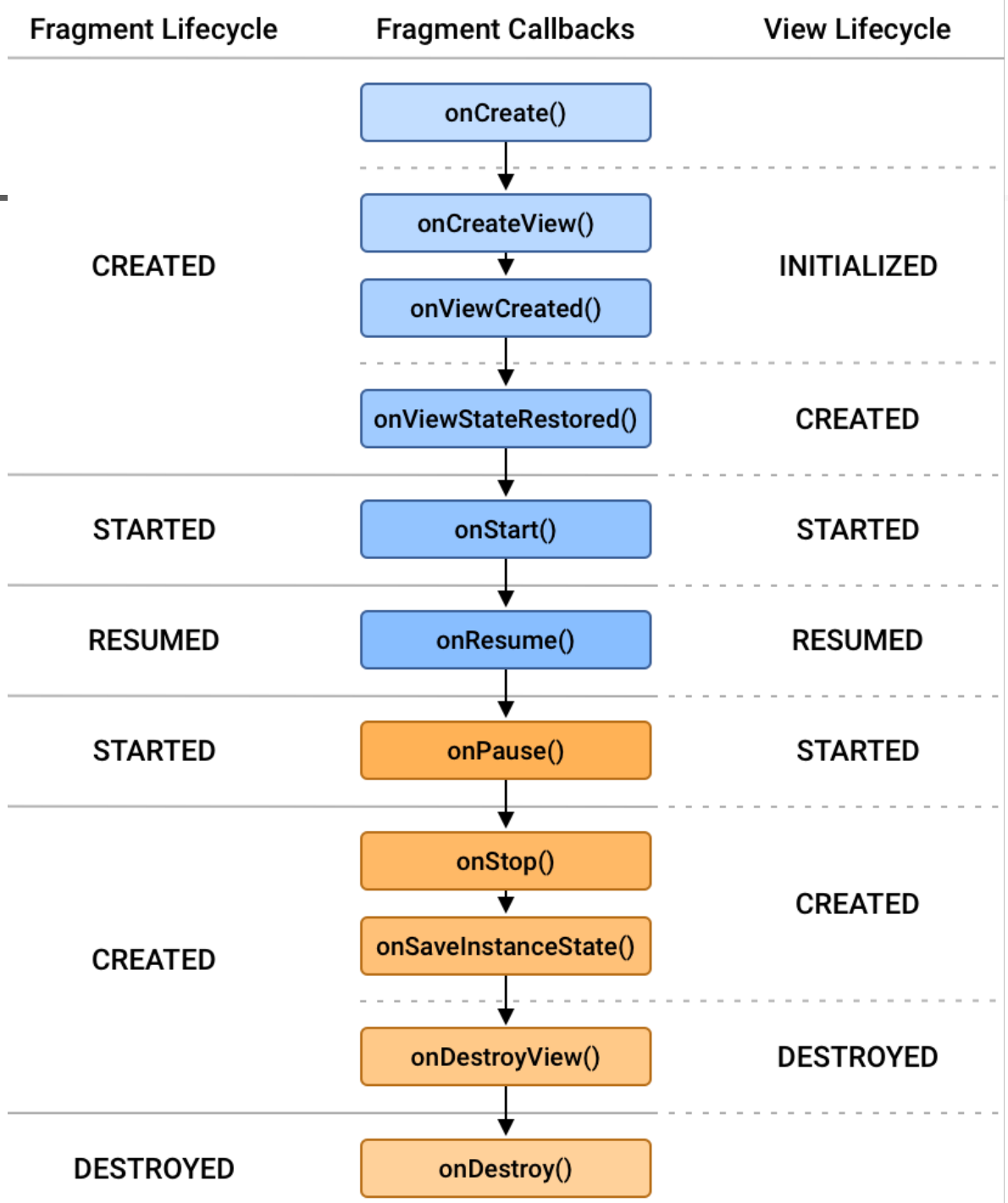


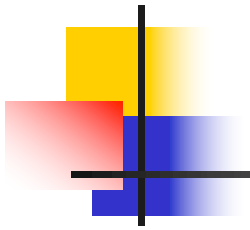
- After a Fragment is ready to be included within an Activity, the lifecycle of the Fragment becomes important.
- A Fragment's lifecycle is precisely synchronized with the Activity's lifecycle.

Attaching and Detaching Fragments with Activities

- The following callback methods are important to managing the lifecycle of a Fragment, as it is created and then destroyed when it is no longer used.
  - `onAttach()`
  - `onCreate()`
  - `onCreateView()`
  - `onViewCreated()` **==> previously, onActivityCreated (deprecated now)**
    `onStart()`
  - `onResume()`
  - `onPause()`
  - `onStop()`
  - `onDestroyView()`
  - `onDestroy()`
  - `onDetach()`

Understanding the Fragment Lifecycle (Androidx)





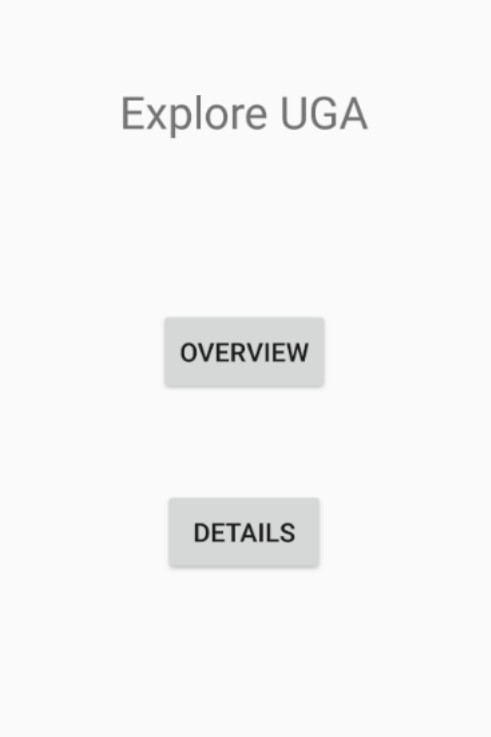
UGA Info Example

- I have created a simple example illustrating how to create and use Android fragments.
- The example is very simple, but it includes fragment replacement operations in both portrait and landscape orientations.
- We will create 2 fragments and use them in different activities.

UGA Info Example

Fragment one: ChoiceFragment

```
public class ChoiceFragment extends Fragment
```

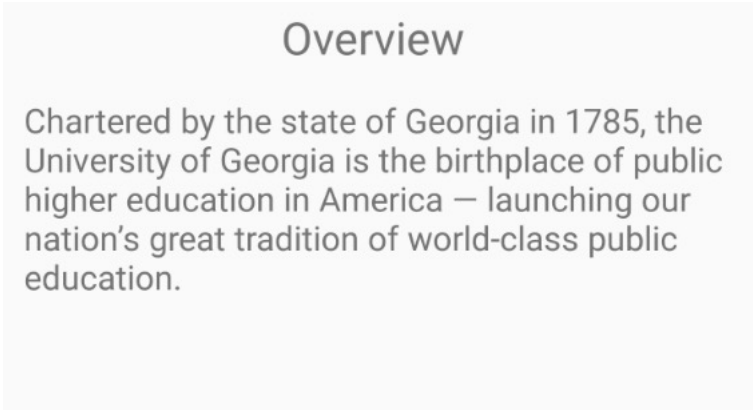



```
<ConstraintLayout
   ...
   <TextView ...
   ...
   <Button ...
      android:text="Overview”
   ...
   <Button ...
      android:text=”Details”
   ...
</ConstraintLayout>
```

UGA Info Example

Fragment two: InfoFragment

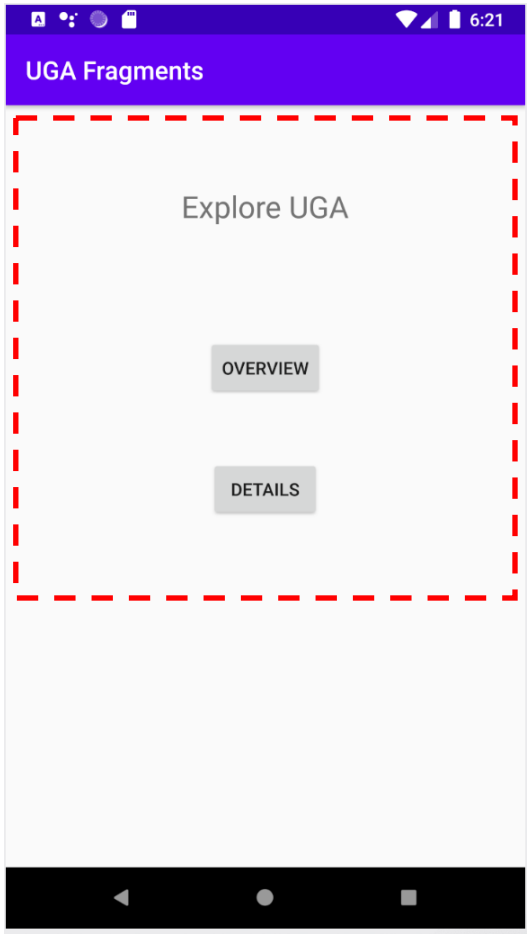
```
public class InfoFragment extends Fragment
```




```
<ConstraintLayout
   …
   <TextView …
      android:text=”Heading”
   …
   <TextView …
      android:text=”Information”
   …
</ConstraintLayout>
```

UGA Info Example

Portrait orientation: MainActivity



activity_main.xml

```
<ConstraintLayout
   …
   <fragment …
      class="edu.uga.cs.ugafragments.ChoiceFragment"
   …
</ConstraintLayout>
```

ChoiceFragment's Button listener:

```
Intent intent = new Intent();
intent.setClass( getActivity(), ViewInfoActivity.class );
startActivity( intent );
```

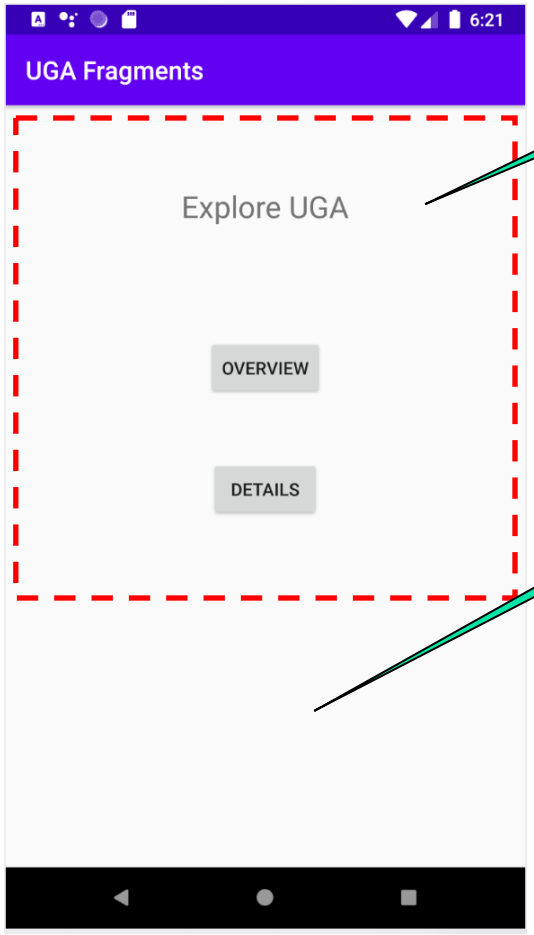
The Intent transitions to ViewInfoActivity

UGA Info Example

ChoiceFragment
Fragment

MainActivity
Activity

Portrait orientation: MainActivity



activity_main.xml

```
<ConstraintLayout
   …
   <fragment …
      class="edu.uga.cs.ugafragments.ChoiceFragment"
   …
</ConstraintLayout>
```

ChoiceFragment's Button listener:

```
Intent intent = new Intent();
intent.setClass( getActivity(), ViewInfoActivity.class );
startActivity( intent );
```

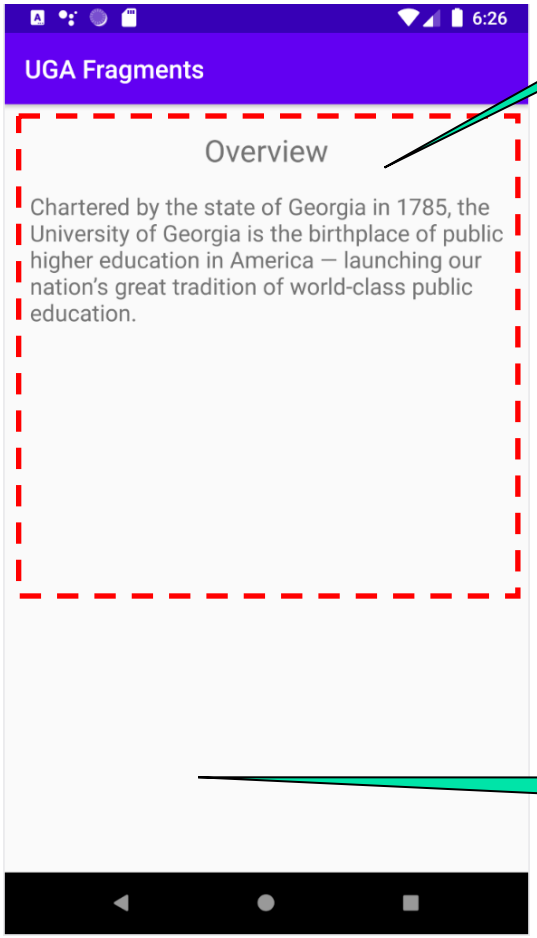
The Intent transitions to ViewInfoActivity

UGA Info Example

InfoFragment
Fragment

Portrait orientation: ViewInfo Activity after

Overview button pressed



ViewInfoActivity creates an InfoFragment and attaches it as its content:

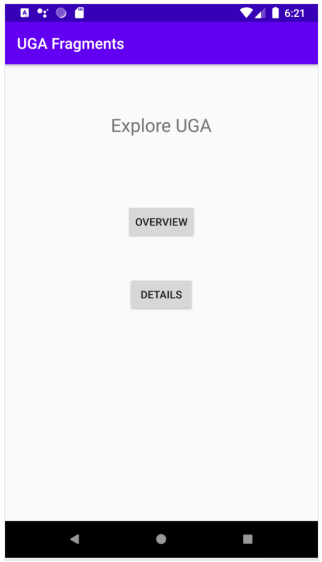
```
InfoFragment infoFragment = InfoFragment.newInstance( overview );
getSupportFragmentManager().beginTransaction()
   .replace( android.R.id.content, infoFragment ).commit();
```

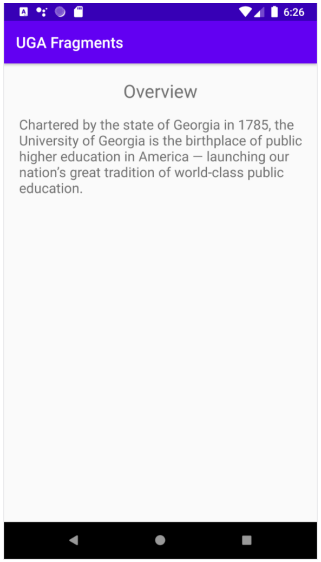
Each Activity has an Android-defined FrameLayout with the identifier "@+id/content".
It is its "content" view.

ViewInfoActivity
Activity

UGA Info Example

After the transition, the app has two activities:



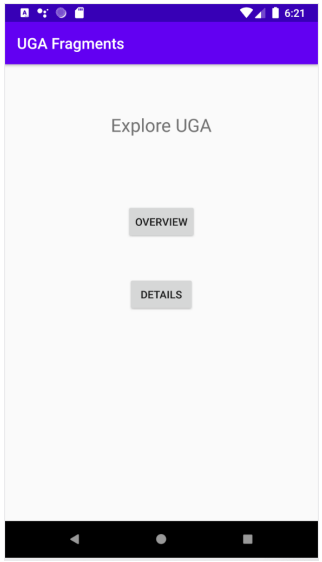


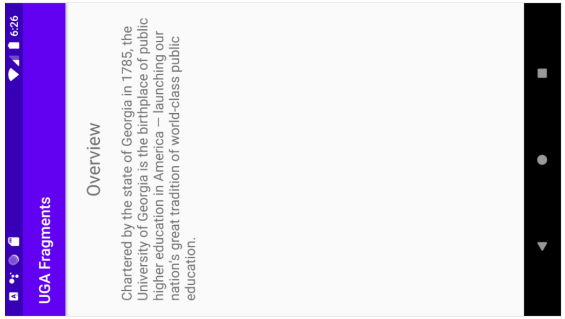
Main Activity
(which is Stopped)

ViewInfo Activity
(which is in Foreground)

UGA Info Example

Suppose the user rotates the device to landscape:



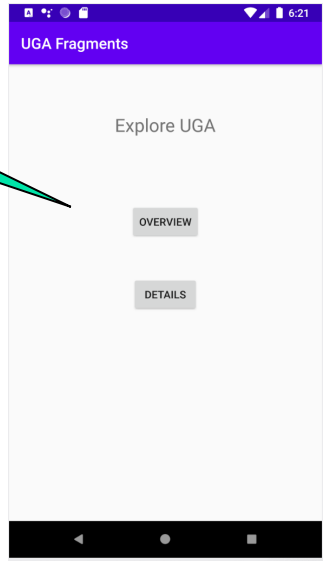


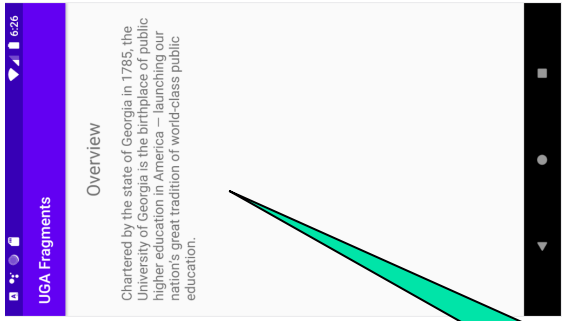
The current choice, i.e., overview vs. details selection can be saved using `saveInstanceState()`

UGA Info Example

Suppose the user rotates the device to landscape

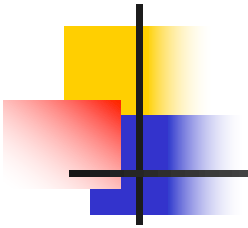
ChoiceFragment
Fragment





InfoFragment
Fragment

The current choice, i.e., overview vs. details selection can be saved using `saveInstanceState()`

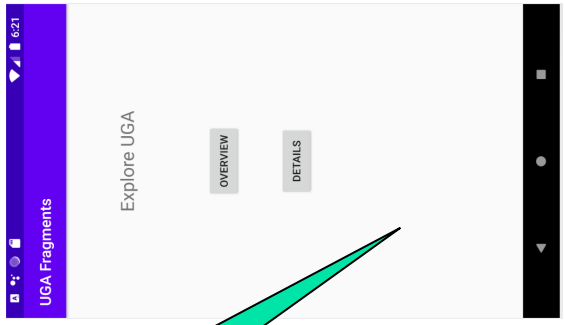


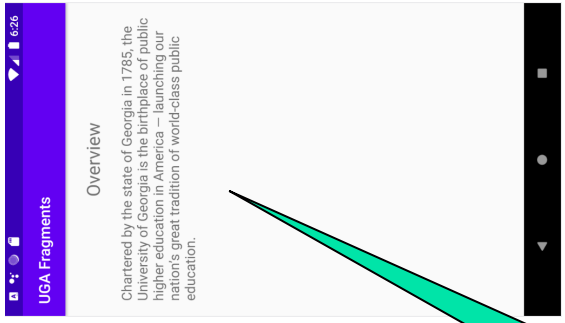
UGA Info Example

- After the orientation change, Android destroys **all** activities that belong to the app.
- This includes destroying all fragments hosted in the activities.
- The activities are then automatically restarted by Android in the new orientation, likely with new layouts.
- This is what happens with our UGA Info app.

UGA Info Example

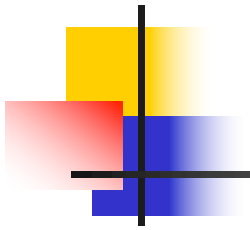
- In our UGA Info app, we would now have two activities in landscape, but only one is needed, since we can show both fragments now!





ChoiceFragment Fragment

InfoFragment Fragment

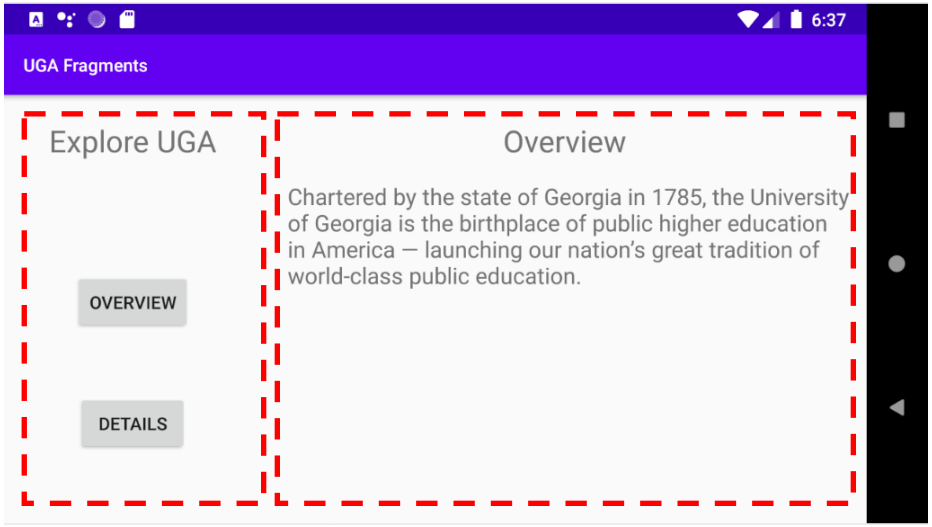


UGA Info Example

- We programmatically destroy the topmost activity, i.e., the one with the InfoFragment.
- We do it in its `onCreate` callback (after checking if we are in the landscape mode), using the `finish()` method.
- The only remaining activity becomes the foreground activity.
- After restart, `onCreate` builds a different layout designed for the landscape orientation.

UGA Info Example

- Landscape orientation: MainActivity



ChoiceFragment is the "driver". It is created and attached by Android.

It senses that it is in landscape mode and creates an InfoFragment and attaches it to the MainActivity in the FrameLayout view.

Overview vs. details choice is saved using `saveInstanceState`.

activity_main.xml (land)

```
<ConstraintLayout
   …
   <fragment …
         class="edu.uga.cs.ugafragments.ChoiceFragment"
   …
   <FrameLayout …
         android:id="@+id/frameLayout"
</ConstraintLayout>
```

UGA Info Example

ChoiceFragment
Fragment

- Landscape orientation: MainActivity

ChoiceFragment is the "driver". It is created and attached by Android.

It then senses that its in landscape and creates an InfoFragment and attaches it to the MainActivity in the FrameLayout view.

Overview vs. details choice is saved using `saveInstanceState`.

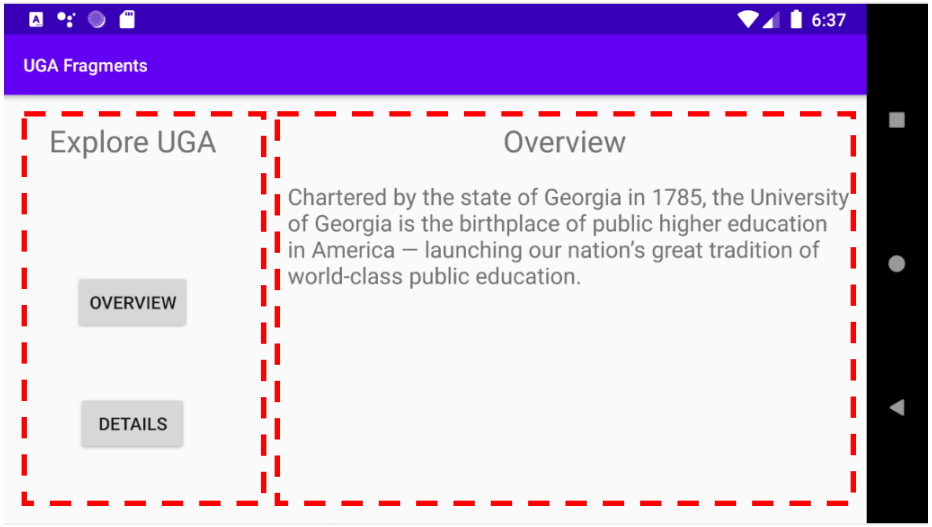
InfoFragment
Fragment

activity_main.xml (land)

```
<ConstraintLayout
   …
   <fragment …
         class="edu.uga.cs.ugafragments.ChoiceFragment"
   …
   <FrameLayout …
         android:id="@+id/frameLayout"
</ConstraintLayout>
```

UGA Info Example

- Landscape orientation: MainActivity

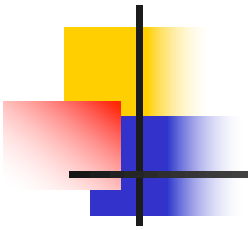



```
InfoFragment infoFragment = InfoFragment.newInstance(overview);

FragmentTransaction fragmentTransaction = getParentFragmentManager ().beginTransaction();

fragmentTransaction.replace(R.id.frameLayout, infoFragment);

fragmentTransaction.commit();
```

Special Types of Fragments

- Android provides some pre-defined Fragment classes, including the following:
  - `ListFragment (androidx.fragment.app.ListFragment)`
  - `PreferenceFragment (androidx.preference.PreferenceFragment)`
  - `WebViewFragment (android.webkit.WebViewFragment)`

    Now, not in androidx; use WebView instead
  - `DialogFragment (androidx.fragment.app.DialogFragment)`



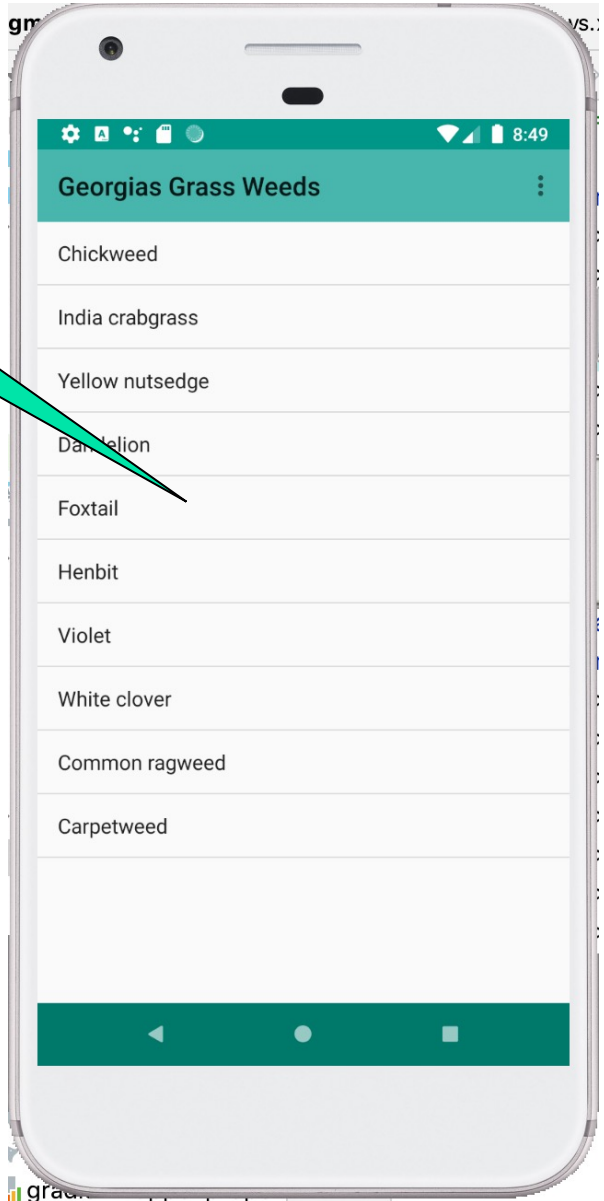
SimpleFragments App Example

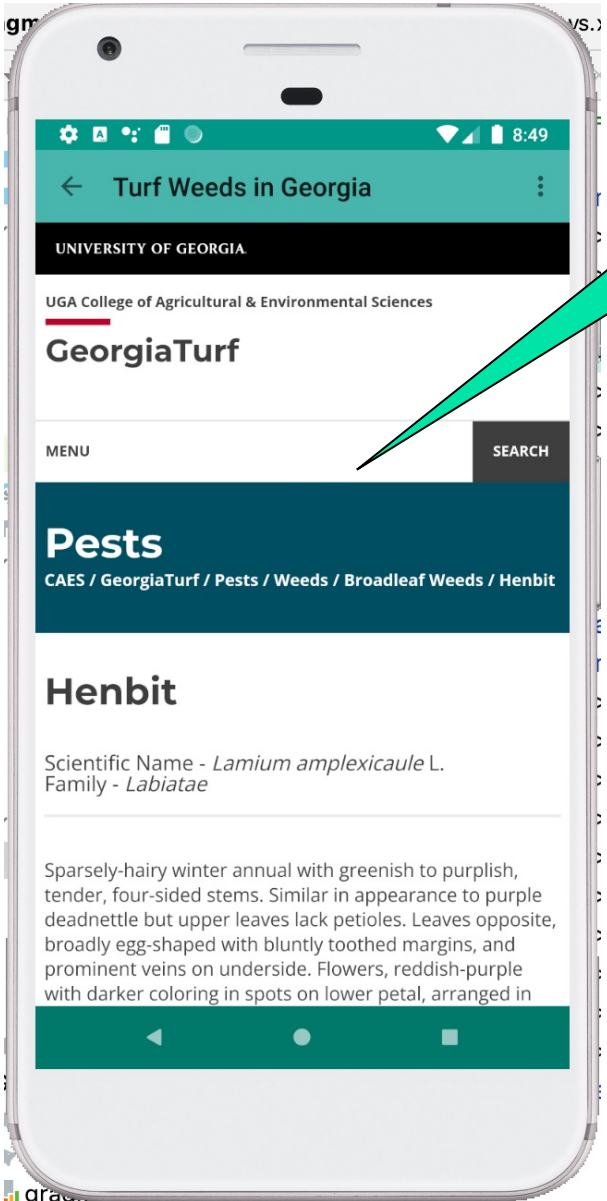
- I modified the SimpleFragments app example from the textbook.
- It was using a defunct websites, so I incorporated some information about Georgia weeds from one of the UGA's sites.

SimpleFragments App Example

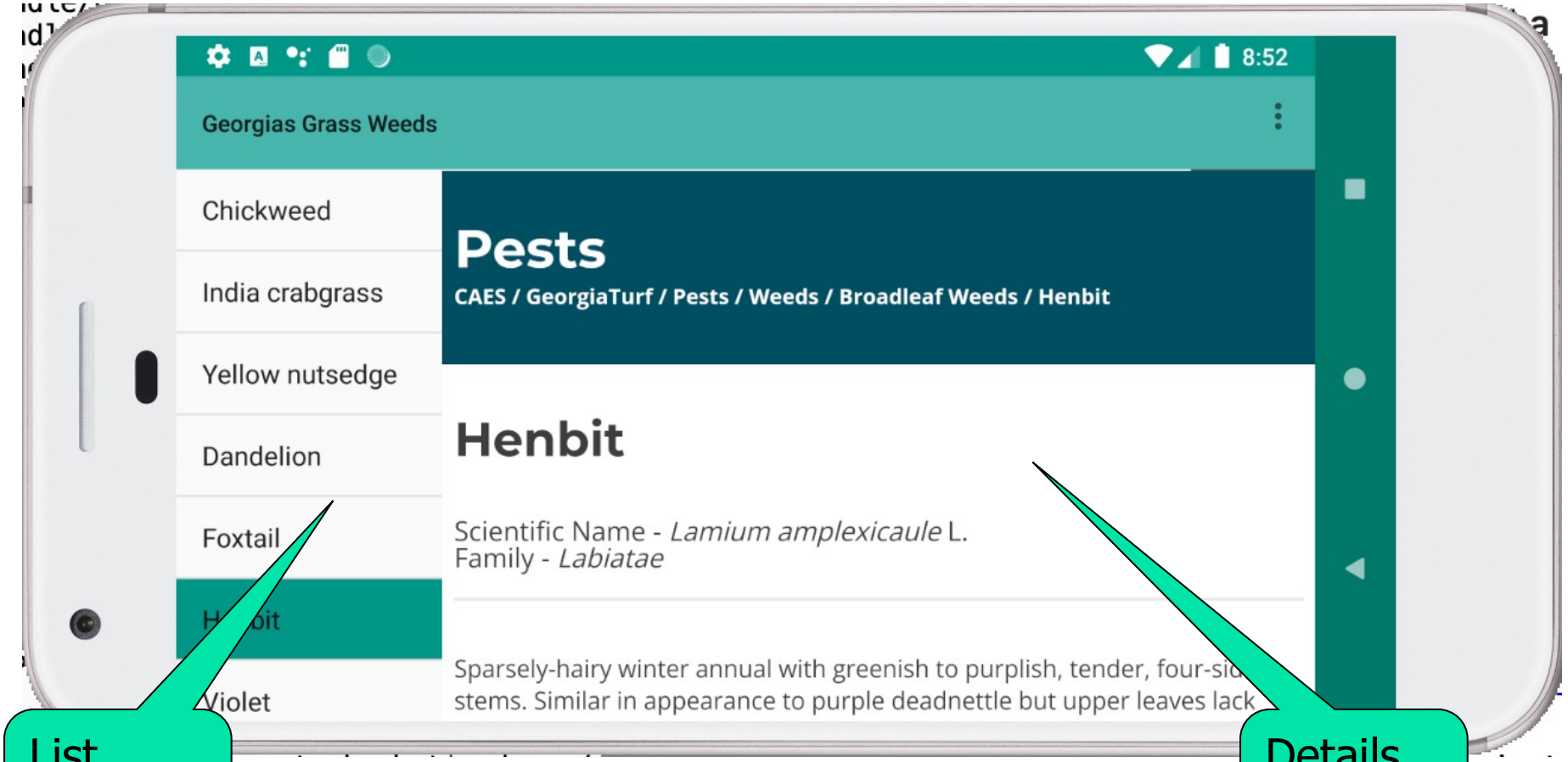
List Fragment

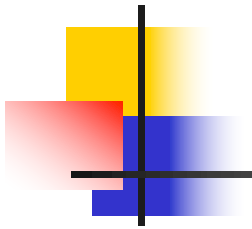
Details Fragment





SimpleFragments App Example





Implementing a ListFragment

- Note that this example app includes the older implementation of Fragments.

```
public class WeedListFragment
   extends ListFragment
   implements
   FragmentManager.OnBackStackChangedListener
{
     private static final String DEBUG_TAG =
                                "WeedListFragment";
     int mCurPosition = -1;
     boolean mShowTwoFragments;
              …
}
```

Implementing a ListFragment

```
public void onActivityCreated(Bundle savedInstanceState) {
    super.onActivityCreated(savedInstanceState);
    getListView().setChoiceMode(ListView.CHOICE_MODE_SINGLE);
    String[] weeds = getResources().getStringArray(
        R.array.weeds_array);
    setListAdapter(new ArrayAdapter<String>(getActivity(),
            android.R.layout.simple_list_item_activated_1,
                                                    weeds));
    View detailsFrame = getActivity().findViewById(R.id.weedentry);
    mShowTwoFragments = detailsFrame != null
                && detailsFrame.getVisibility() == View.VISIBLE;
    if (savedInstanceState != null) {
        mCurPosition = savedInstanceState.getInt("curChoice", 0);
    }
    if (mShowTwoFragments == true || mCurPosition != -1) {
        viewWeedInfo(mCurPosition);
    }
    getFragmentManager().addOnBackStackChangedListener(this);
}
```

Implementing a ListFragment

```
void viewWeedInfo(int index) {
    mCurPosition = index;
    if (mShowTwoFragments == true) {
    // Check what fragment is currently shown, replace if needed.
        WeedWebViewFragment details =
            (WeedWebViewFragment) getFragmentManager()
                .findFragmentById(R.id.weedentry);
        if (details == null || details.getShownIndex() != index)
        {
            WeedWebViewFragment newDetails =
                WeedWebViewFragment.newInstance(index);

            FragmentManager fm = getFragmentManager();
            FragmentTransaction ft = fm.beginTransaction();
            ft.replace(R.id.weedentry, newDetails);
    …
```

Older

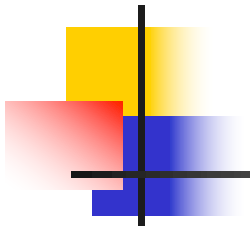
Implementing a ListFragment

```
        …
        if (index != -1) {
            String[] weeds =
            getResources().getStringArray(R.array.weeds_array);
            String strBackStackTagName = weeds[index];
            ft.addToBackStack(strBackStackTagName);
        }
      ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE);
        ft.commit();
    }
} else {
    Intent intent = new Intent();
    intent.setClass(getActivity(),
                    WeedViewActivity.class);
    intent.putExtra("index", index);
    startActivity(intent);
} } }
```

Implementing a ListFragment

```
@Override
public void onBackStackChanged() {
    WeedWebViewFragment details =
        (WeedWebViewFragment)
                getFragmentManager()
            .findFragmentById(R.id.weedentry);
    if (details != null) {
        mCurPosition = details.getShownIndex();
        getListView().setItemChecked(mCurPosition, true);

        if (!mShowTwoFragments) {
            viewWeedInfo(mCurPosition);
        }
    }
}
```

Implementing a WebViewFragment



- Next, we create a custom WebViewFragment class called `WeedWebViewFragment` to host the basic info related to each weed.
- This Fragment class does little more than determine which info entry URL to load and then load it in the WebView control.

```
public class WeedWebViewFragment
             extends WebViewFragment {
    …
}
```

Implementing a WebViewFragment

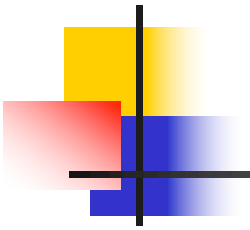
```
@Override
public void onActivityCreated(Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);
        String[] weedsUrls = getResources()
            .getStringArray(R.array.weedurls_array);
        int weedUrlIndex = getShownIndex();
        WebView webview = getWebView();
        webview.setPadding(0, 0, 0, 0);
        webview.getSettings().setLoadWithOverviewMode(true);
        webview.getSettings().setUseWideViewPort(true);
        if (weedUrlIndex != -1) {
                String weedUrl = weedUrls[weedUrlIndex];
                webview.loadUrl(weedUrl);
        }  else {
                String weedUrl = “https://turf.caes.uga.edu/pest-
management/weeds/broadleaf-weeds/common-chickweed.html";
                webview.loadUrl(weedUrl);
        }
    } }
```

Defining the Layout Files

- Now that you have implemented your Fragment classes, you can place them in the appropriate layout resource files.
- You need to create two layout files:
  - In landscape mode, you'll want a single activity_simple_fragments.xml layout file to host both Fragment components.
  - In portrait mode, you will want a comparable layout file that hosts only the ListFragment you implemented.
- The WebViewFragment you implemented will have a user interface generated at runtime.

Defining the Layout Files

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <include android:id="@+id/toolbar"
        layout="@layout/tool_bar" />
    <LinearLayout android:orientation="horizontal"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:baselineAligned="false">
        <fragment
            android:name="com.androidintro.simplefragments.WeedListFragment"
            android:id="@+id/list"
            android:layout_weight="1"
            android:layout_width="0dp"
            android:layout_height="match_parent" />
        <FrameLayout android:id="@+id/weedeentry"
            android:layout_weight="4"
            android:layout_width="0dp"
            android:layout_height="match_parent" />
    </LinearLayout>
</LinearLayout>
```



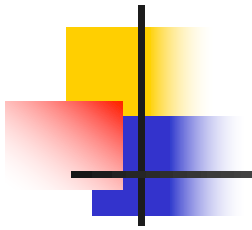
Defining the Layout Files

- The resources stored in the normal layout directory will be used whenever the device is not in landscape mode (in other words, portrait mode).
- Here, we need to define two layout files.
- First, let's define our static ListFragment (using the <fragment> element) in its own file:

  res/layout/activity_simple_fragments.xml

Defining the Layout Files

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <include android:id="@+id/toolbar"
        layout="@layout/tool_bar" />
    <fragment
        android:name=
            "com.androidintro.simplefragments.WeedListFragment"
        android:id="@+id/list"
        android:layout_weight="1"
        android:layout_width="0dp"
        android:layout_height="match_parent" />
</LinearLayout>
```



Defining the Activity Classes

- Now, you need to define your Activity classes to host your Fragment components.
- You will need two Activity classes:
  - A primary class
    - Let's call the primary Activity class SimpleFragmentsActivity.
  - A secondary class that is used only to display the WeedWebViewFragment when in portrait mode
    - Let's call the secondary Activity class WeedViewActivity.

Defining the Activity Classes

- As mentioned earlier, moving all your user interface logic to Fragment components greatly simplifies your Activity class implementation.
- To add a landscape layout, open a layout in Design view, then open the activity resource drop down and select “Create Landscape Qualifier”.

Defining the Activity Classes

```
public class SimpleFragmentsActivity
        extends AppCompatActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_simple_fragments);
        Toolbar toolbar;
        Toolbar = (Toolbar) findViewById(R.id.toolbar);
        getSupportActionBar(toolbar);
    }
}
```

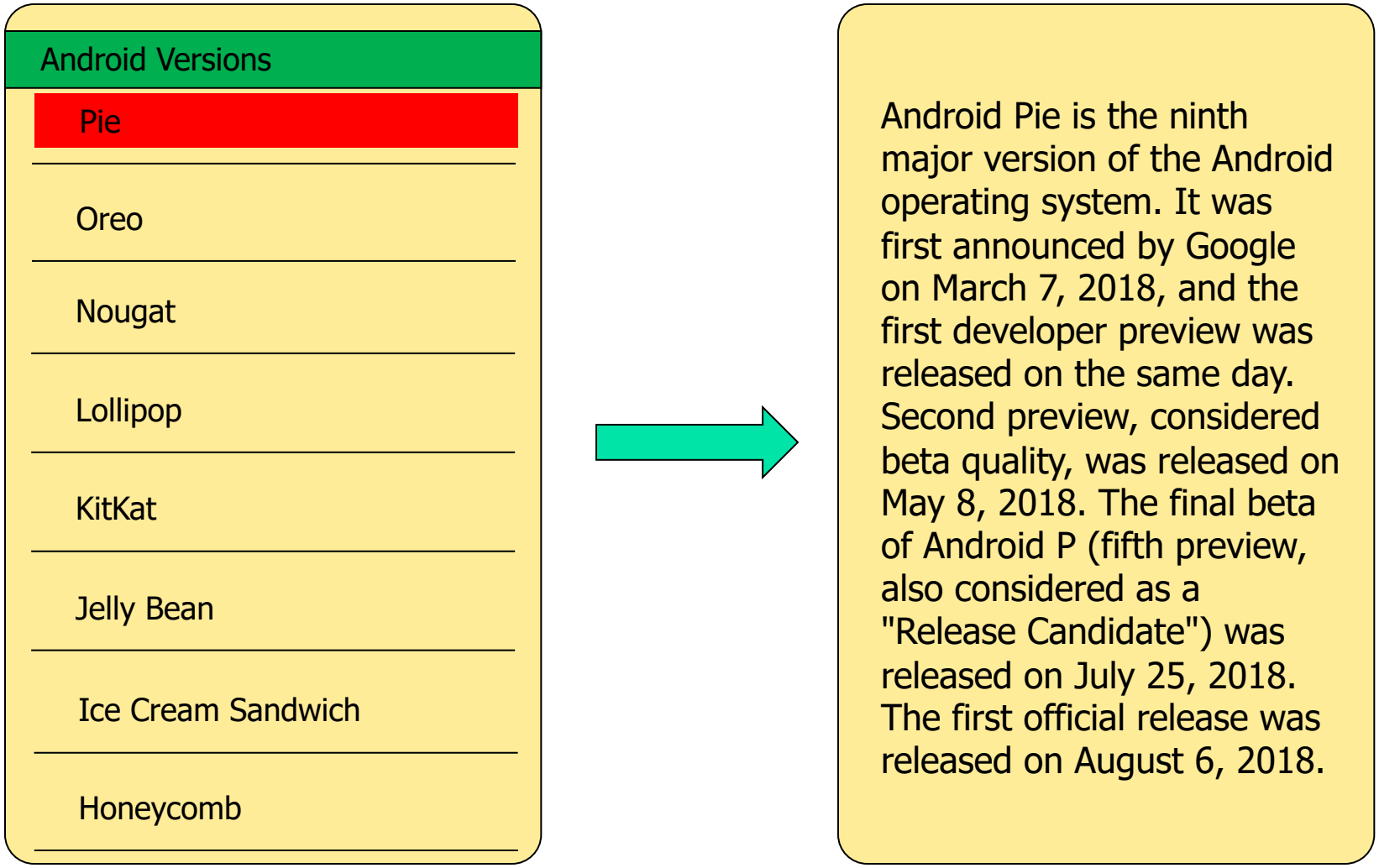
Defining the Activity Classes

```
public class WeedViewActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        if (getResources().getConfiguration().orientation ==
                Configuration.ORIENTATION_LANDSCAPE) {
            finish();
            return;
        }
        if (savedInstanceState == null) {
            setContentView(R.layout.activity_simple_fragments);
            Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
            getSupportActionBar(toolbar);
            getSupportActionBar().setDisplayHomeAsUpEnabled(true);
            WeedWebViewFragment details = new WeedWebViewFragment();
            details.setArguments(getIntent().getExtras());
            FragmentManager fm = getFragmentManager();
            FragmentTransaction ft = fm.beginTransaction();
            ft.replace(android.R.id.content, details);
            ft.commit();
        }
    } }
```

Android Versions Example

- I have created another app to further illustrate the use of Android fragments.
- The app shows some general information about various versions of the Android system.
- One more time, the app implements a very typical pattern frequently referred to as master-detail flow.
- The app will be available on eLC in the Sample Apps folder.

Understanding Fragments



Understanding Fragments

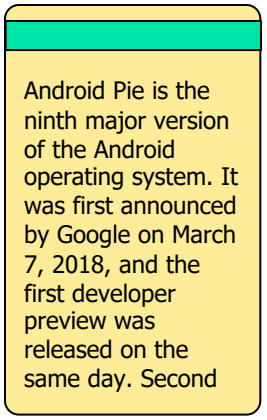
Android Versions

- Pie
- Oreo
- Nougat
- Lollipop
- KitKat
- Jelly Bean
- Ice Cream Sandwich

Android Pie is the ninth major version of the Android operating system. It was first announced by Google on March 7, 2018, and the first developer preview was released on the same day. Second preview, considered beta quality, was released on May 8, 2018. The final beta of Android P (fifth preview, also considered as a "Release Candidate") was released on July 25, 2018. The first official release was released on August 6, 2018.

Understanding Fragments

```
<FragmentContainerView>
    id="list"
    class="edu.uga…"
    …
</FragmentContainerView>
```




```
<FragmentContainerView>
    id="list"
    …
    …
</FragmentContainerView>
```

Android creates an instance automatically and attaches it at FrameContainerView; the class attribute indicates the Fragment class to use.

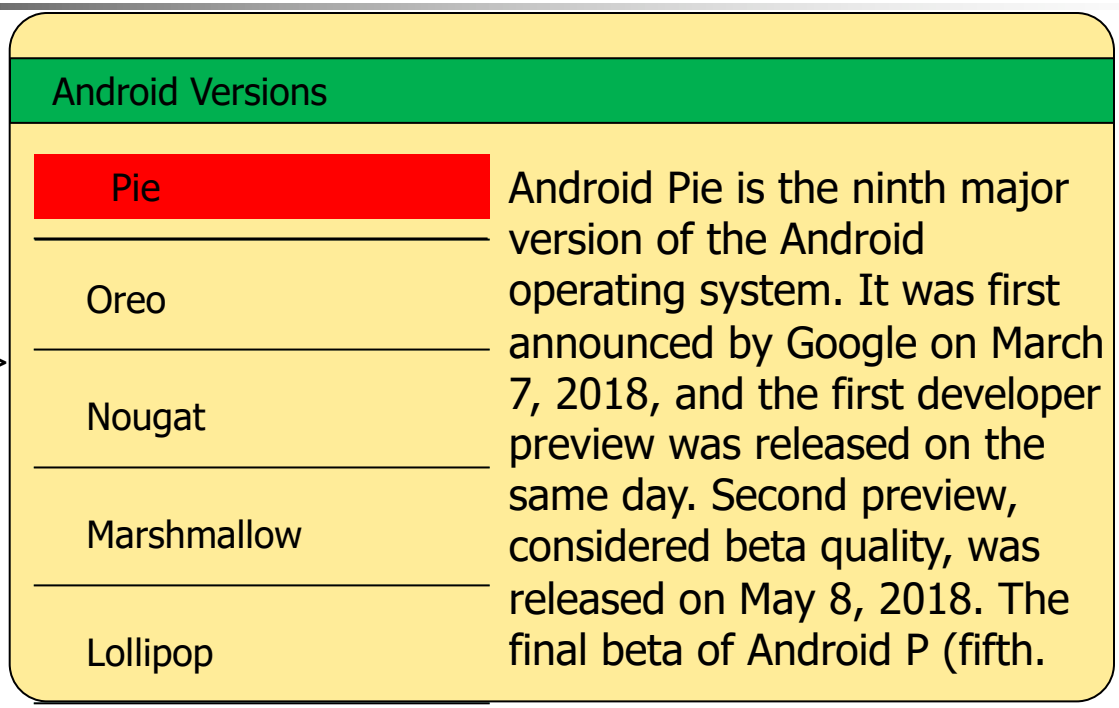
```
info = …Fragment.newInstane();
… beginTransaction().
replace( android.R.id.content, info ).
commit();
```

```
class AndroidVersionsListFragment
{
…
…
…
}
```

```
class AndroidVersionInfoFragment
{
…
…
…
}
```

Entire content view element is used here

Understanding Fragments




```
<FragmentContainerView>
   id="list"
   class="edu…"
   …
</FragmentContainerView>
```

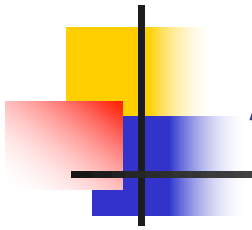
```
<FrameLayout>
   id="info"
   …
   …
</FrameLayout>
```

An instance created and attached by Android automatically, as before.

```
info = AndroidVersionInfoFragment.
                .newInstane();
… .replace(R.id.info, info )
```

```
class AndroidVersionsListFragment
{
…
…
…
}
```

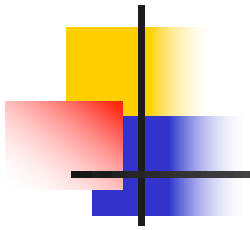
```
class AndroidVersionInfoFragment
{
…
…
…
}
```



Additional Ways to Use Fragments

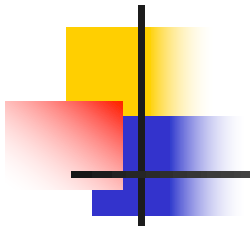
- Fragments are great for creating reusable interface components.
- There are other ways to use fragments within your application.
  - You can create reusable behavior components without a user interface.
  - You can nest fragments within fragments.

Behavior Fragments without a User Interface

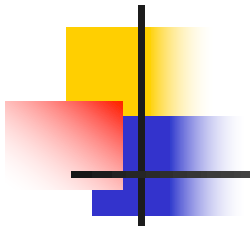


- Fragments are not only for decoupling a user interface component from an Activity.
- You may also want to decouple application behaviors, such as background processing, into a reusable Fragment.
- Rather than providing the ID of a resource, you simply provide a unique string tag when adding or replacing a Fragment.

Behavior Fragments without a User Interface



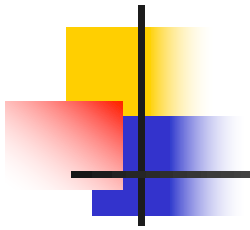
- Because you are not adding a particular View to your layout, the call to the `onCreateView()` method is never called.
- Just make sure to use the `findFragmentByTag()` to retrieve this behavior Fragment from your Activity.



Exploring Nested Fragments

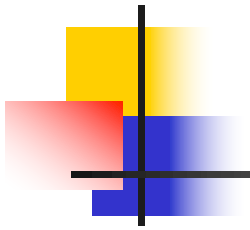
- A recent addition to Android 4.2 (API Level 17) is the ability to nest fragments within fragments.
- Nested fragments have also been added to the Android Support Library, making this API capability available all the way back to Android 1.6 (API Level 4).

Exploring Nested Fragments

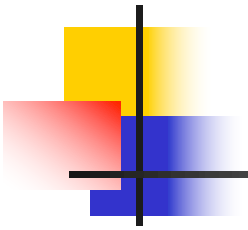


- In order to add a Fragment within another Fragment, you must invoke the Fragment method `getChildFragmentManager()`, which returns a `FragmentManager`.
- Once you have the `FragmentManager`, you can start a `FragmentTransaction` by calling `beginTransaction()` and then invoking the `add()` method, including the Fragment to add and its layout, followed by the `commit()` method.

Exploring Nested Fragments



- 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
  - Along with a host of many other use cases



Summary

- We have learned about the Fragment lifecycle and how it relates to the Activity lifecycle.
- We have learned how to define fragments.
- We are now able to manage Fragment modifications.
- We are now able to work with ListFragment and WebViewFragment.
- We should now be confident designing Fragment-based applications.
- We have learned that the Android Support Package is required for providing Fragment support to applications prior to API Level 11.
- We have introduced the concept of nested fragments.

References and More Information

- Android Training: “Building a Dynamic UI with Fragments”:
  - http://d.android.com/training/basics/fragments/index.html
- Android API Guides: “Fragments”:
  - http://d.android.com/guide/components/fragments.html
- Android SDK Reference regarding the application Fragment class:
  - https://developer.android.com/reference/androidx/fragment/app/Fragment.html
- Android SDK Reference regarding the application ListFragment class:
  - https://developer.android.com/reference/android/support/v4/app/ListFragment.html