Android Fragments

How and Why

http://goo.gl/gdXmCj

A Quick Survey

http://goo.gl/BzRoFy

About Me

- John Lombardo
- @johnnylambada
  john@lombardos.org
- Android for 5 years
- Boeing, Scopely, Originate

Follow along if you like

http://goo.gl/gdXmCj

- We'll get interactive later in the class.
- I'll be using the latest Android Studio with Gradle
- Don't worry about keeping up, the slides have lots of pictures
- You should fork my repo, that way you can:
  a. Make your own changes along the way
  b. Send me pull requests
- Here's the repo: https://github.com/johnnylambada/andevcon-2014-jl.git

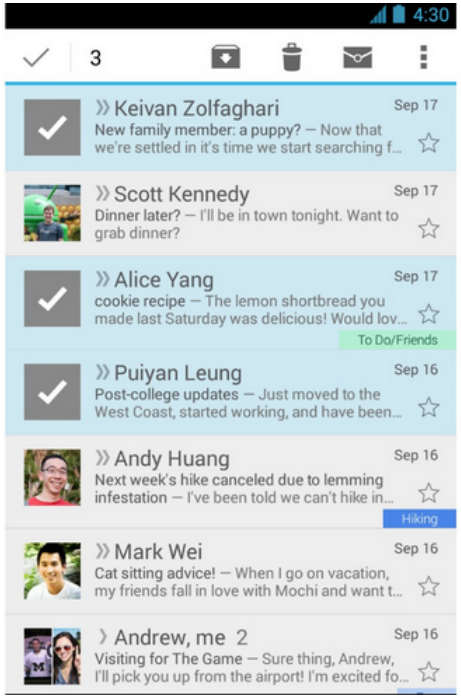
What is a Fragment?

To answer this question, we have to understand what an Activity is…

… an activity is (normally) one “screen” worth of information in an Android app.

Google defines it as follows:

> An activity is a single, focused thing that the user can do.



Activity

- Must be defined in AndroidManifest.xml

```
<activity android:name=".app.HelloWorld" android:label="App/Activity/Hello World">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.SAMPLE_CODE" />
    </intent-filter>
</activity>
```

- Must extend android.app.Activity

```
import android.app.Activity;
import android.os.Bundle;

public class HelloWorld extends Activity
{
```

- Normally has an associated layout

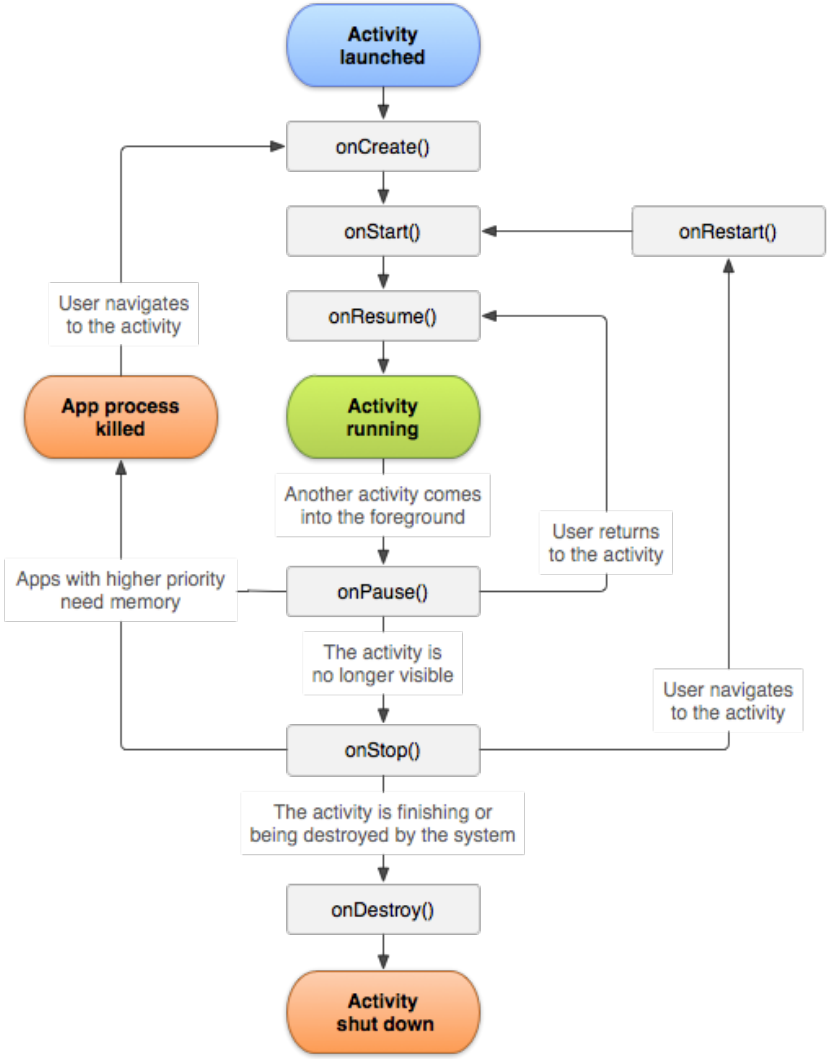
```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.hello_world);
}
```

- Has a lifecycle ...

Activity lifecycle

- Lifecycle Events typically come in pairs
  - onCreate / onDestroy
  - onStart / onStop
  - onResume / onPause

- All lifecycle events are important, but some are more important than others…
  - onCreate is used for initialization
  - onPause is is where you deal with the user leaving your activity



So, what are Fragments

Google says:

A Fragment represents a behavior or a portion of user interface in an Activity. You can combine multiple fragments in a single activity to build a multi-pane UI and reuse a fragment in multiple activities. 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).

So, a Fragment:

- Is part of one or more Activities
- Can be combined with other Fragments
- Has it’s own Lifecycle
- Can be added or removed

Wait, what about <include> and <merge>?

- <include> let's you include Views defined in one layout xml file in another xml file.
- <merge> does the same, but merges multiple xml elements into the parent container.
- They're awesome and you should use them!
- But...
- <include> and <merge> don't allow you to include any code. If you have a button in the included code, it must be wired up to code in each separate place.
- So, they're really helpful for keeping your xml DRY, but not much else.

Wait, what about subclassing a View?

- That seems perfect -- when you subclass a View, you can do anything you want!
- The problem is, you can't combine several Views (buttons, textboxes, etc.) when you subclass a View. You have to override onMeasure and onDraw.
- You're given a Canvas on which to draw.
- Also, if you subclass a View, you're going to have to add business logic to it or have a bunch of other classes that manage the View's data.
- So that's what a Fragment is -- it's a compromise between:

```
<include layout="@layout/stat">
```

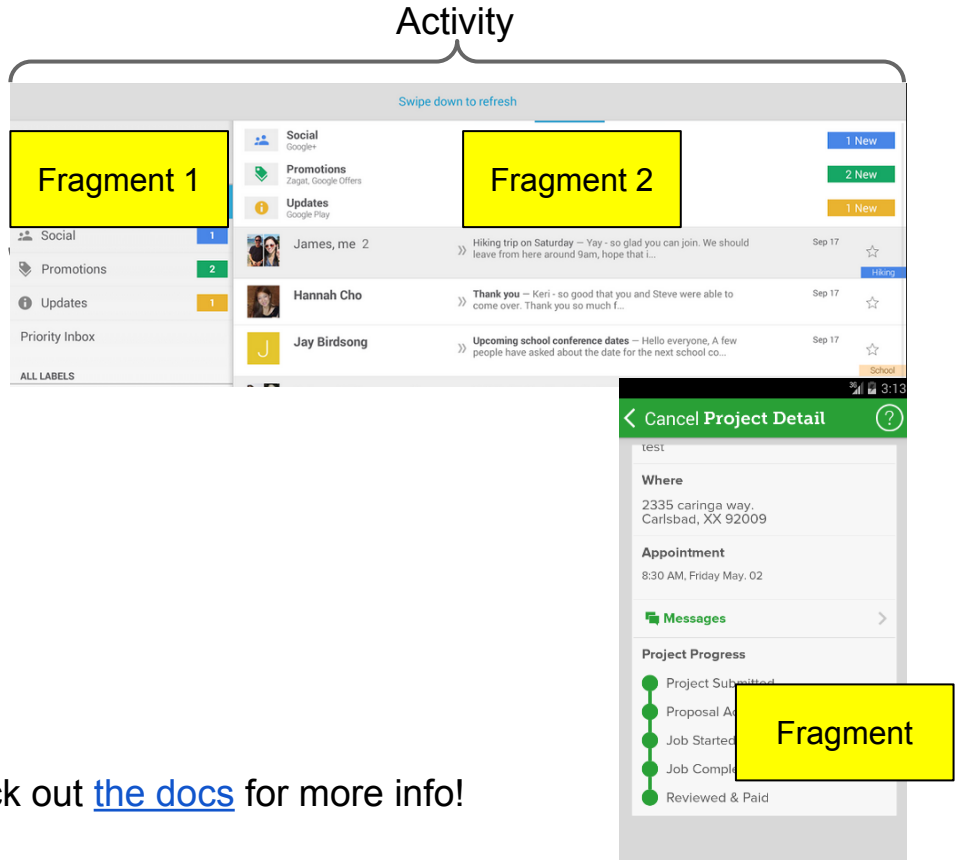
and

```
public StatView extends View{
```

Fragment Uses - Keep your code DRY

There are three high-level use cases for Fragments:

- Don’t Repeat Yourself when tabletizing a phone app.
- Don’t Repeat Yourself when UI elements are reused within your app.
- Non-UI Fragments can be used to host costly data that should be retained over configuration changes.



Check out [the docs](#) for more info!

Activity vs Fragment

- Must be defined in AndroidManifest.xml
- Must extend android.app.Activity
- Normally has an associated layout
- Has a lifecycle.

- ~~Must be defined in AndroidManifest.xml~~
- Must extend android.app.Fragment

  (or android.support.v4.app.Fragment, but MATL...)
- Normally has an associated layout
- Has a lifecycle.

Fragment

- Must extend android.app.Fragment

```
public static class MyFragment extends Fragment {
```

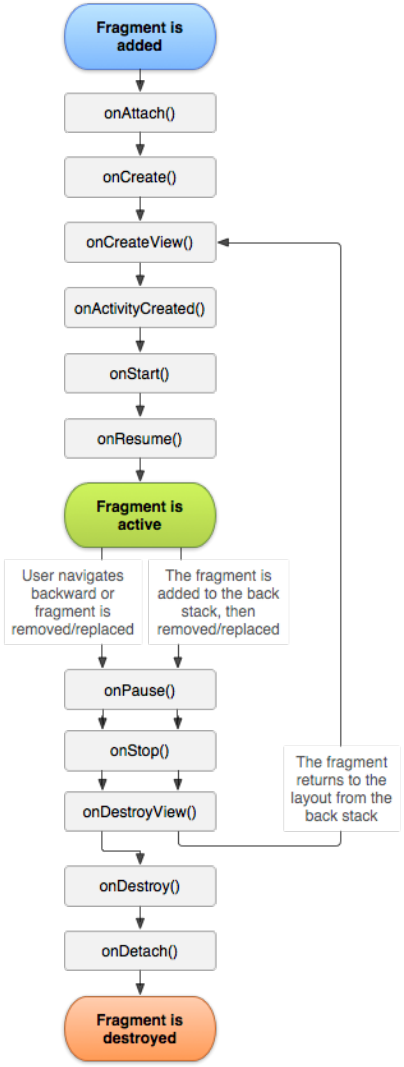
- Normally has an associated layout

```
@Override public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
    View v = inflater.inflate(R.layout.hello_world, container, false);
    return v;
}
```

- Has a lifecycle ...

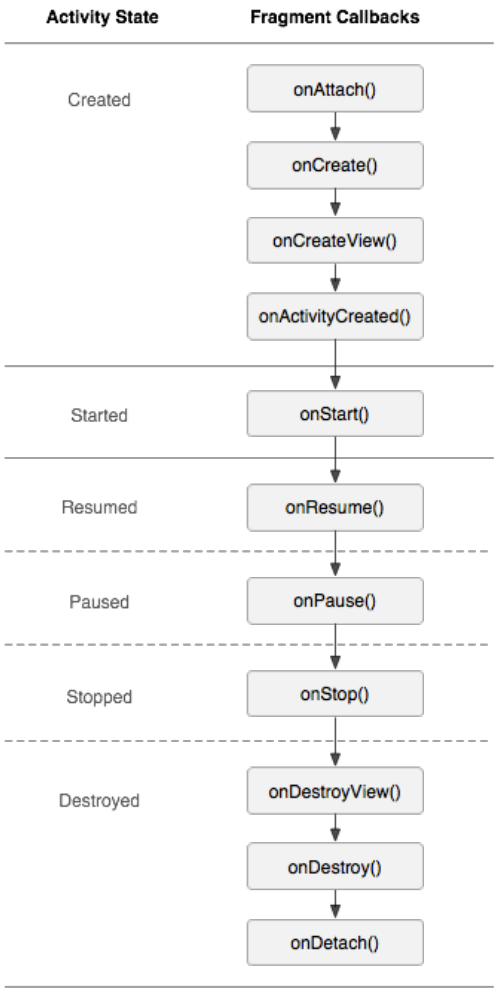
The Fragment Lifecycle

- Hmm, it *looks* familiar -- like the Activity Lifecycle.
- However, there are some other states
  - onInflate - static fragments get args from the layout
  - onAttach / onDetach - when associated with activity
  - onActivityCreated - onCreate of Activity completed
  - onCreateView / onDestroyView - return the View of the frag

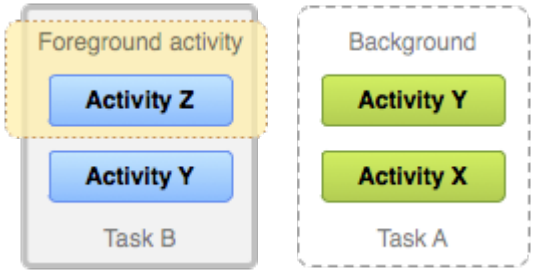


Fragment / Activity Lifecycle

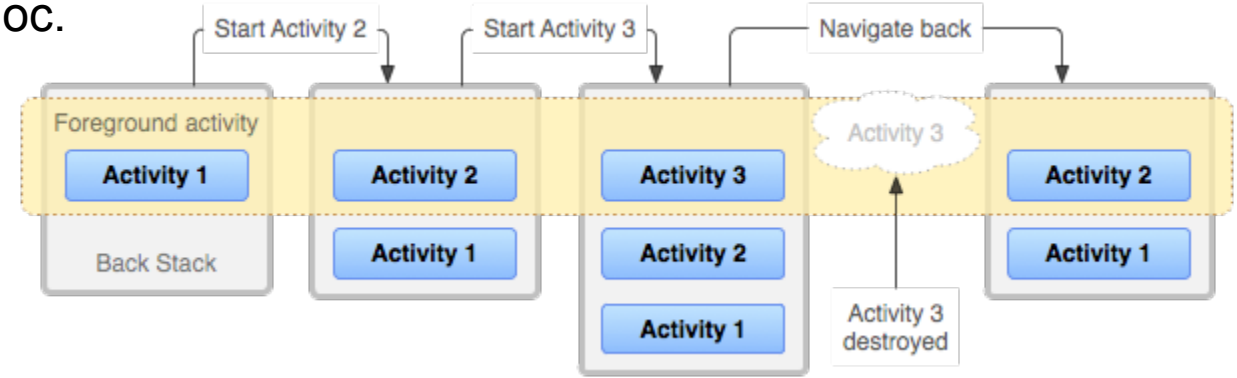
- The Fragment lifecycle closely matches the Activity's
- The Fragment lifecycle has a few extra states mostly having to do with when it is attached and detached from the Activity.
- When the Activity is in the resumed state, Fragments can be added and removed from the activity. Thus, only while the activity is in the resumed state can the lifecycle of a fragment change independently.
- At all other times, the Fragment's lifecycle is locked to the Activity's lifecycle.



Tasks and the Back Stack



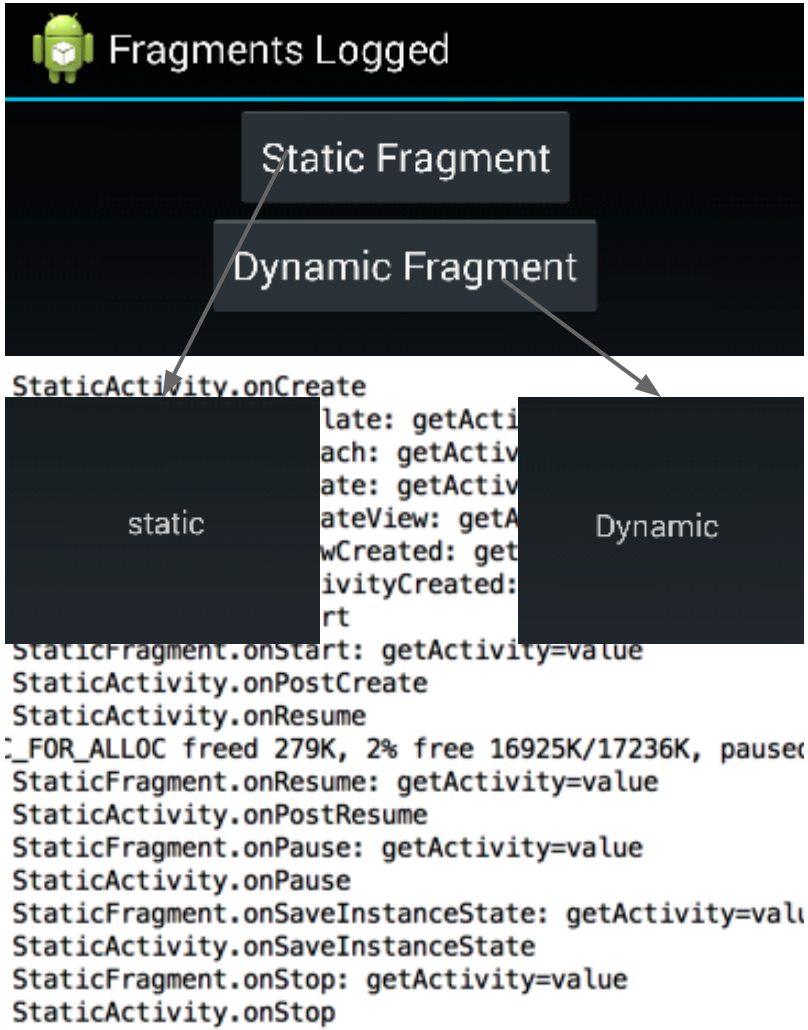
- Android Tasks act very much like tabs in a browser.
- The back button acts very much like the back button of a browser.
- There is no “forward button”, once you go Back the Activity is destroyed.
- When your Activity isn’t in the foreground, it’s in the onStopped state.
- When it’s in the onStopped state, it’s can be destroyed to recoup memory.
- In that case, when it’s recreated, it’s saveInstanceState will be non-null.
- Read the linked doc.



Let's See a Sample

We'll make a simple Fragment app that shows a simple menu:

- Static button: will display a new Activity showing the word "static" in the middle of a fragment.
- Dynamic button: shows a different activity with the same fragment, this time showing the word "dynamic"
- And we'll log all lifecycle events.

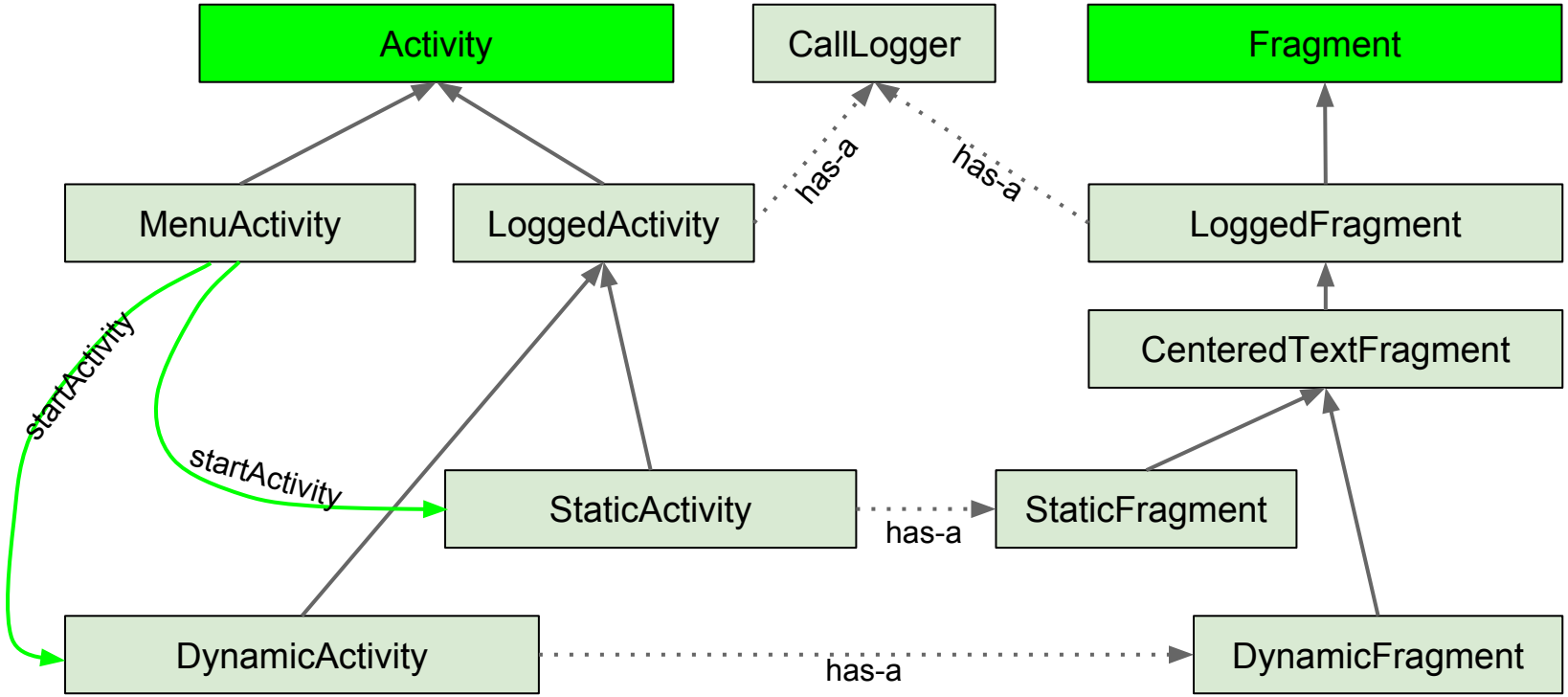


FragmentLogged: Concepts to consider

The FragmentLogged Example should help you understand the following:

- The difference between Static and Dynamic Fragments
    - How to initialize them both
- The interrelations between the Activity and Fragment lifecycle
- How to use the instance state Bundle to store state through the Android lifecycle

FragmentLogged: Class Overview

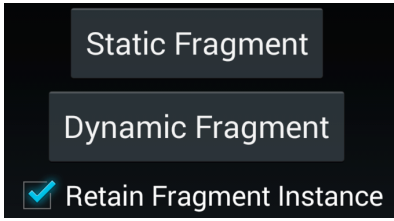


FragmentLogged: AndroidManifest.xml

Pro Tip: AndroidManifest is the map to any Android project

- targetSdkVersion = 19
  minSdkVersion = 13
  It is eligible to use native Fragments
- Three Activities:
  - MenuActivity
  - StaticActivity
  - DynamicActivity

FragmentLogged: MenuActivity




```java
public class MenuActivity extends Activity {
    public final static String RETAIN_INSTANCE = "RETAIN_INSTANCE";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.menu_layout);
    }

    public void startStaticFragmentActivity(View view){
        startActivity(getIntent(StaticActivity.class));
    }

    public void startDynamicFragmentActivity(View view){
        startActivity(getIntent(DynamicActivity.class));
    }

    private Intent getIntent(Class<?> cls) {
        boolean retainInstance = ((CheckBox)findViewById(R.id.retainInstance))
                .isChecked();
        Intent intent = new Intent(this,cls);
        intent.putExtra(RETAIN_INSTANCE, retainInstance);
        return intent;
    }
}
```

```xml
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Static Fragment"
    android:id="@+id/staticFragmentButton"
    android:layout_gravity="center_horizontal"
    android:onClick="startStaticFragmentActivity" />

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Dynamic Fragment"
    android:id="@+id/dynamicFragmentButton"
    android:layout_gravity="center_horizontal"
    android:onClick="startDynamicFragmentActivity" />

<CheckBox
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Retain Fragment Instance"
    android:layout_gravity="center_horizontal"
    android:id="@+id/retainInstance" />
```

CallLogger

- CallLogger is a handy utility class that you can use in your own projects.
- It logs the name of the calling function to the Android log along with its class.

Return the method name of caller

```
private static String getCaller(String stopAt) {
    StackTraceElement[] stacktrace = Thread.currentThread().getStackTrace();
    boolean logged = false;
    boolean foundMe = false;
    for(int i=0; i<stacktrace.length; i++) {
        StackTraceElement e = stacktrace[i];
        String methodName = e.getMethodName();
        if (foundMe) {
            if (!methodName.startsWith("access$")) {
                return methodName;
            }
        } else {
            if (methodName.equals(stopAt)) {
                foundMe = true;
            }
        }
    }
    return null;
}
```

Log a fragment, shows when getActivity is null

```
public static void logFragment(Fragment fragment){
    String caller = getCaller("logFragment");
    if (caller==null)
        logFail(fragment);
    else {
        StringBuffer b = new StringBuffer();
        b.append(fragment.getClass().getSimpleName())
                .append('.')
                .append(caller)
                .append(": getActivity=")
                .append(fragment.getActivity()==null?"null":"value");
        Log.i(TAG, b.toString());
    }
}
```

Log a generic method

```
public static void logMethod(Object object){
    String caller = getCaller("logMethod");
    if (caller==null)
        logFail(object);
    else
        Log.i(TAG,object.getClass().getSimpleName()+"."+caller);
}
```

LoggedFragment & LoggedActivity

- LoggedFragment & LoggedActivity both use CallLogger at many of the interesting lifecycle callbacks.
- You can insert these classes within your project's inheritance hierarchy to see what lifecycle calls are being made.

```
public class LoggedActivity extends Activity {
    @Override public void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstar
    @Override public void onDestroy() { super.onDestroy(); CallLogger.logMethod(this); }
    @Override public void onNewIntent(Intent intent) { super.onNewIntent(intent); CallLogg
    @Override public void onPause() { super.onPause(); CallLogger.logMethod(this); }
    @Override public void onPostCreate(Bundle savedInstanceState) { super.onPostCreate(sav
    @Override public void onPostResume() { super.onPostResume(); CallLogger.logMethod(this
    @Override public void onRestart() { super.onRestart(); CallLogger.logMethod(this); }
    @Override public void onRestoreInstanceState(Bundle savedInstanceState) { super.onRest
    @Override public void onResume() { super.onResume(); CallLogger.logMethod(this); }
    @Override public void onSaveInstanceState(Bundle outState) { super.onSaveInstanceState
    @Override public void onStart() { super.onStart(); CallLogger.logMethod(this); }
    @Override public void onStop() { super.onStop(); CallLogger.logMethod(this); }
    @Override public void onUserLeaveHint() { super.onUserLeaveHint(); CallLogger.logMetho
}
```

```
public class LoggedFragment extends Fragment {
    @Override public void onActivityCreated(Bundle sis) { super.onActivityCreated(sis);
    @Override public void onAttach(Activity activity) { super.onAttach(activity); Call
    @Override public void onConfigurationChanged(Configuration newConfig) { super.onCo
    @Override public void onCreate(Bundle sis) { super.onCreate(sis); CallLogger.logFra
    @Override public View onCreateView(LayoutInflater inflater, ViewGroup container, Bu
    @Override public void onDestroy() { super.onDestroy(); CallLogger.logFragment(this
    @Override public void onDestroyView() { super.onDestroyView(); CallLogger.logFragme
    @Override public void onDetach() { super.onDetach(); CallLogger.logFragment(this);
    @Override public void onHiddenChanged(boolean hidden) { super.onHiddenChanged(hidde
    @Override public void onInflate(Activity activity, AttributeSet attrs, Bundle sis)
    @Override public void onPause() { super.onPause(); CallLogger.logFragment(this);}
    @Override public void onResume() { super.onResume(); CallLogger.logFragment(this);
    @Override public void onSaveInstanceState(Bundle outState) { super.onSaveInstanceSt
    @Override public void onStart() { super.onStart(); CallLogger.logFragment(this);}
    @Override public void onStop() { super.onStop(); CallLogger.logFragment(this);}
    @Override public void onViewCreated(View view, Bundle sis) { super.onViewCreated(v
}
```

StaticActivity & DynamicActivity

Static: Fragment is defined statically in the layout -- including the “android:label”

```java
public class StaticActivity extends LoggedActivity{
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.static_fragment_activity_layout);
    }
}
```

```xml
<fragment class="andevcon14.FragmentComms.StaticFragment"
    android:id="@+id/embedded"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:label="static"
    />
```

Dynamic: Fragment is defined dynamically at run time -- a FrameLayout holds its spot

```java
public class DynamicActivity extends LoggedActivity{
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.dynamic_fragment_activity_layout);

        if (savedInstanceState == null) {
            FragmentTransaction ft = getFragmentManager().beginTransaction();
            DynamicFragment newFragment = DynamicFragment.newInstance("Dynamic");
            ft.add(R.id.dynamic, newFragment);
            ft.commit();
        }
    }
}
```

```xml
<FrameLayout
    android:id="@+id/dynamic"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    />
```

Dynamic: Why “if (savedInstState==null)”?

- Note that many Activities check for savedInstanceState to be null… why?

- If savedInstanceState is null, that means that this is the first time this Activity has been created in this Task.

```
if (savedInstanceState == null) {
    FragmentTransaction ft = getFragmentManager().beginTransaction();
    DynamicFragment newFragment = DynamicFragment.newInstance("Dynamic");
    ft.add(R.id.dynamic, newFragment);
    ft.commit();
}
```

- Fragments can be configured so that they are not destroyed during a configuration change (i.e. portrait to landscape), so they can be used to retain costly state.

- Make sure to call setRetainInstance(true) in the Fragment’s onCreate.

Static: Read XML from Java with R.styleable

- Reading XML elements from Java is possible.
- To do so, you must declare a specific element as styleable.
- You can create your own or re-use Android styles as the example shows.
- The example shows the onInflate method reading the android:label text.
- Want more details? Check out this blog post.

```
<fragment class="andevcon14.FragmentComms.StaticFragment"
    android:id="@+id/embedded"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:label="static"
    />
```

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <declare-styleable name="FragmentArguments">
        <attr name="android:label" />
    </declare-styleable>
</resources>
```

```
public void onInflate(Activity activity, AttributeSet attrs, Bundle sis) {
    super.onInflate(activity, attrs, sis);
    TypedArray   a = activity.obtainStyledAttributes(
            attrs,R.styleable.FragmentArguments);
    CharSequence cs = a.getText(
            R.styleable.FragmentArguments_android_label);
    if (cs!=null) setText(cs.toString());
    a.recycle();
}
```

StaticFragment & DynamicFragment

- Both inherit from CenteredTextFragment
- Each is initialized differently:
  - Static using an XML attribute from the layout.
  - Dynamic as an argument to the static factory method that creates the object.

StaticFragment

```
public class StaticFragment extends CenteredTextFragment {
    @Override
    public void onInflate(Activity activity, AttributeSet attrs, Bundle sis) {
        super.onInflate(activity, attrs, sis);
        TypedArray   a = activity.obtainStyledAttributes(
                attrs,R.styleable.FragmentArguments);
        CharSequence cs = a.getText(
                R.styleable.FragmentArguments_android_label);
        if (cs!=null) setText(cs.toString());
        a.recycle();
    }
}
```

DynamicFragment

```
public class DynamicFragment extends CenteredTextFragment {
    public static DynamicFragment newInstance(String text){
        DynamicFragment df = new DynamicFragment();
        df.setText(text);
        return df;
    }
}
```

CenteredTextFragment

- Must keep track of its state through configuration transition events.
- We use the Bundle passed to these functions for that:
  - →onActivityCreated
  - →onCreate
  - →onCreateView
  - →onInflate
  - ←onSaveInstanceState
  - →onViewCreated

I like to encapsulate all of the Bundle management logic & constants into a separate final static inner class as shown below.

```
private final static class BundleManager {
    private final static String TEXT = "TEXT";

    public static void saveToBundle(CenteredTextFragment fragment, Bundle bundle) {
        if (bundle != null && fragment != null)
            bundle.putString(TEXT, fragment.getText());
    }

    public static void getFromBundle(CenteredTextFragment fragment, Bundle bundle) {
        if (bundle != null && fragment != null)
            fragment.setText(bundle.getString(TEXT));
    }
}
```

CenteredTextFragment

- Encapsulating the Bundle logic in its own inner class makes each of the lifecycle methods as easy as:
  - BundleManager.saveToBundle

    or

  - BundleManager.getFromBundle

```
@Override
public void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);
    BundleManager.saveToBundle(this, outState);
}

@Override
public void onViewCreated(View view, Bundle sis) {
    super.onViewCreated(view, sis);
    BundleManager.getFromBundle(this, sis);
}
```

Static Fragment created, then back button

```
StaticActivity.onCreate
StaticFragment.onInflate: getActivity=null
StaticFragment.onAttach: getActivity=value
StaticFragment.onCreate: getActivity=value
StaticFragment.onCreateView: getActivity=value
StaticFragment.onViewCreated: getActivity=value
StaticFragment.onActivityCreated: getActivity=value
StaticActivity.onStart
StaticFragment.onStart: getActivity=value
StaticActivity.onPostCreate
StaticActivity.onResume
StaticFragment.onResume: getActivity=value
StaticActivity.onPostResume
<back>
StaticFragment.onPause: getActivity=value
StaticActivity.onPause
StaticFragment.onStop: getActivity=value
StaticActivity.onStop
StaticFragment.onDestroyView: getActivity=value
StaticFragment.onDestroy: getActivity=value
StaticFragment.onDetach: getActivity=value
StaticActivity.onDestroy
```

Notes

- Activity.onCreate happens first
- Frag.onInflate is for pulling attributes from the layout. It's getActivity returns null, but it has access to the Activity via an argument.
- Both Fragment and Activity are Stopped and Destroyed on the Back button.

Dynamic Fragment created, then back button

```
DynamicActivity.onCreate
DynamicFragment.onAttach: getActivity=value
DynamicFragment.onCreate: getActivity=value
DynamicFragment.onCreateView: getActivity=value
DynamicFragment.onViewCreated: getActivity=value
DynamicFragment.onActivityCreated: getActivity=value
DynamicActivity.onStart
DynamicFragment.onStart: getActivity=value
DynamicActivity.onPostCreate
DynamicActivity.onResume
DynamicFragment.onResume: getActivity=value
DynamicActivity.onPostResume
<back>
DynamicFragment.onPause: getActivity=value
DynamicActivity.onPause
DynamicFragment.onStop: getActivity=value
DynamicActivity.onStop
DynamicFragment.onDestroyView: getActivity=value
DynamicFragment.onDestroy: getActivity=value
DynamicFragment.onDetach: getActivity=value
DynamicActivity.onDestroy
```

Notes

- No onInflate
- Very similar to Static

Static Fragment Rotation

retainInstance=false

```
StaticActivity.onCreate
StaticFragment.onInflate: getActivity=null
StaticFragment.onAttach: getActivity=value
StaticFragment.onCreate: getActivity=value
StaticFragment.onCreateView: getActivity=value
StaticFragment.onViewCreated: getActivity=value
StaticFragment.onActivityCreated: getActivity=value
StaticActivity.onStart
StaticFragment.onStart: getActivity=value
StaticActivity.onPostCreate
StaticActivity.onResume
StaticFragment.onResume: getActivity=value
StaticActivity.onPostResume
<rotate>
StaticFragment.onPause: getActivity=value
StaticActivity.onPause
StaticFragment.onSaveInstanceState: getActivity=value
StaticActivity.onSaveInstanceState
StaticFragment.onStop: getActivity=value
StaticActivity.onStop
StaticFragment.onDestroyView: getActivity=value
StaticFragment.onDestroy: getActivity=value
StaticFragment.onDetach: getActivity=value
StaticActivity.onDestroy
StaticActivity.onCreate
StaticFragment.onInflate: getActivity=null
StaticFragment.onAttach: getActivity=value
StaticFragment.onCreate: getActivity=value
StaticFragment.onCreateView: getActivity=value
StaticFragment.onViewCreated: getActivity=value
StaticFragment.onActivityCreated: getActivity=value
StaticActivity.onStart
StaticFragment.onStart: getActivity=value
StaticActivity.onRestoreInstanceState
StaticActivity.onPostCreate
StaticActivity.onResume
StaticFragment.onResume: getActivity=value
StaticActivity.onPostResume
```

retainInstance=true

```
StaticActivity.onCreate
StaticFragment.onInflate: getActivity=null
StaticFragment.onAttach: getActivity=value
StaticFragment.onCreate: getActivity=value
StaticFragment.onCreateView: getActivity=value
StaticFragment.onViewCreated: getActivity=value
StaticFragment.onActivityCreated: getActivity=value
StaticActivity.onStart
StaticFragment.onStart: getActivity=value
StaticActivity.onPostCreate
StaticActivity.onResume
StaticFragment.onResume: getActivity=value
StaticActivity.onPostResume
<rotate>
StaticFragment.onPause: getActivity=value
StaticActivity.onPause
StaticFragment.onSaveInstanceState: getActivity=value
StaticActivity.onSaveInstanceState
StaticFragment.onStop: getActivity=value
StaticActivity.onStop
StaticFragment.onDestroyView: getActivity=value

StaticFragment.onDetach: getActivity=value
StaticActivity.onDestroy
StaticActivity.onCreate

StaticFragment.onAttach: getActivity=value

StaticFragment.onCreateView: getActivity=value
StaticFragment.onViewCreated: getActivity=value
StaticFragment.onActivityCreated: getActivity=value
StaticActivity.onStart
StaticFragment.onStart: getActivity=value
StaticActivity.onRestoreInstanceState
StaticActivity.onPostCreate
StaticActivity.onResume
StaticFragment.onResume: getActivity=value
StaticActivity.onPostResume
```

Notes

- Notice that the Fragment goes through onDestroy/Create when retainInstance=false.
- onInflate is not called the second time when retainInstance=true.

Dynamic Fragment Rotation

retainInstance=false

```
DynamicActivity.onCreate
DynamicFragment.onAttach: getActivity=value
DynamicFragment.onCreate: getActivity=value
DynamicFragment.onCreateView: getActivity=value
DynamicFragment.onViewCreated: getActivity=value
DynamicFragment.onActivityCreated: getActivity=value
DynamicActivity.onStart
DynamicFragment.onStart: getActivity=value
DynamicActivity.onPostCreate
DynamicActivity.onResume
DynamicFragment.onResume: getActivity=value
DynamicActivity.onPostResume
<rotate>
DynamicFragment.onPause: getActivity=value
DynamicActivity.onPause
DynamicFragment.onSaveInstanceState: getActivity=value
DynamicActivity.onSaveInstanceState
DynamicFragment.onStop: getActivity=value
DynamicActivity.onStop
DynamicFragment.onDestroyView: getActivity=value
DynamicFragment.onDestroy: getActivity=value
DynamicFragment.onDetach: getActivity=value
DynamicActivity.onDestroy
DynamicFragment.onAttach: getActivity=value
DynamicFragment.onCreate: getActivity=value
DynamicActivity.onCreate
DynamicFragment.onCreateView: getActivity=value
DynamicFragment.onViewCreated: getActivity=value
DynamicFragment.onActivityCreated: getActivity=value
DynamicActivity.onStart
DynamicFragment.onStart: getActivity=value
DynamicActivity.onRestoreInstanceState
DynamicActivity.onPostCreate
DynamicActivity.onResume
DynamicFragment.onResume: getActivity=value
DynamicActivity.onPostResume
```

retainInstance=true

```
DynamicActivity.onCreate
DynamicFragment.onAttach: getActivity=value
DynamicFragment.onCreate: getActivity=value
DynamicFragment.onCreateView: getActivity=value
DynamicFragment.onViewCreated: getActivity=value
DynamicFragment.onActivityCreated: getActivity=value
DynamicActivity.onStart
DynamicFragment.onStart: getActivity=value
DynamicActivity.onPostCreate
DynamicActivity.onResume
DynamicFragment.onResume: getActivity=value
DynamicActivity.onPostResume
<rotate>
DynamicFragment.onPause: getActivity=value
DynamicActivity.onPause
DynamicFragment.onSaveInstanceState: getActivity=value
DynamicActivity.onSaveInstanceState
DynamicFragment.onStop: getActivity=value
DynamicActivity.onStop
DynamicFragment.onDestroyView: getActivity=value

DynamicFragment.onDetach: getActivity=value
DynamicActivity.onDestroy
DynamicFragment.onAttach: getActivity=value
DynamicActivity.onCreate

DynamicFragment.onCreateView: getActivity=value
DynamicFragment.onViewCreated: getActivity=value
DynamicFragment.onActivityCreated: getActivity=value
DynamicActivity.onStart
DynamicFragment.onStart: getActivity=value
DynamicActivity.onRestoreInstanceState
DynamicActivity.onPostCreate
DynamicActivity.onResume
DynamicFragment.onResume: getActivity=value
DynamicActivity.onPostResume
```

Notes

- Notice that the Fragment goes through onDestroy/Create when retainInstance=false.
- onInflate is not a lifecycle event in a dynamic fragment

Activity / Fragment communication

- There are several ways to communicate among Activitys and Fragments
- We’ll explore five of them, but there are probably some I missed

Activity / Fragment Communication

Types of communication we would like to do:

- Once vs ongoing
- Activity → Fragment (A→F)
- Fragment → Activity (F→A)
- Fragment → Fragment (F→F)

Some techniques we can use:

- Activity's Intent
- Fragment setArguments
- Fragment Factory Method
- Local Broadcast
- Observer pattern

Activity / Fragment Communication

	once/ ongoing	static/ dynamic	A → F	F → A	F → F
Activity's Intent	once	both	yes	no	no
setArguments	once	dynamic	yes	no	yes
layout elements	once	static	yes	no	no
Factory Method	once	dynamic	yes	no	yes
*Local Broadcast	ongoing	both	yes	yes	yes
Observer Pattern	ongoing	both	yes	yes	yes

*Only supported by the Support Library

Activity Intent

- Each Activity is started with an Intent
- Intents can have Extras “putExtra(name,value)”
- A Fragment can get to the Intent using the Activity’s getIntent()
- The value can be pulled from Intent using getXExtra()
- It’s that easy.

```
public class ActivityIntentStarter extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        Intent intent = new Intent(this,ActivityIntentActivity.class);
        intent.putExtra("text",this.getClass().getSimpleName());
        startActivity(intent);
        finish();
    }
}
```

```
public class ActivityIntentActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_intent);
    }
}
```

```
@Override
public void onAttach(Activity activity) {
    super.onAttach(activity);
    String text;
    Intent intent = activity.getIntent();
    if (intent == null) text = TAG + " null Intent"; else {
        text = intent.getStringExtra("text");
        if (text==null) text = TAG+" no getStringExtra";
    }
    setText(text);
}
```

Fragment Factory Method

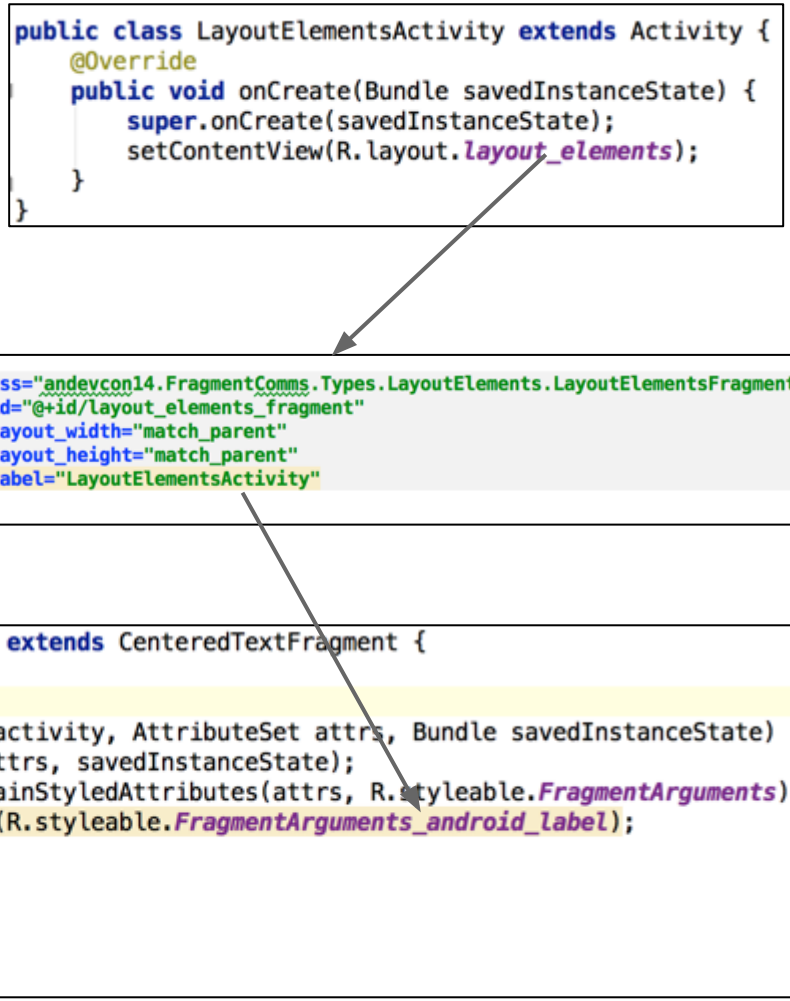
- Fragment has a static factory method used by Activity to instantiate a Fragment with the proper parameters.
- This is the preferred above using the new keyword.

```
public class FactoryMethodActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.set_arguments);

        if (savedInstanceState == null) {
            FactoryMethodFragment fragment = FactoryMethodFragment
                    .newInstance(this.getClass().getSimpleName());
            getFragmentManager()
                .beginTransaction()
                .add(R.id.dynamic, fragment)
                .commit();
        }
    }
}
```

```
public class FactoryMethodFragment extends CenteredTextFragment {
    private final static String TAG = FactoryMethodFragment.class.getSimpleName();

    public static FactoryMethodFragment newInstance(String text){
        FactoryMethodFragment fragment = new FactoryMethodFragment();
        fragment.setText(text);
        return fragment;
    }
}
```

Layout Elements

- We saw this in an earlier example.
- Only useful for static Fragments



```
public class LayoutElementsActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.layout_elements);
    }
}
```

```
<fragment class="andevcon14.FragmentComms.Types.LayoutElements.LayoutElementsFragment"
    android:id="@+id/layout_elements_fragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:label="LayoutElementsActivity"
    />
```

```
public class LayoutElementsFragment extends CenteredTextFragment {

    @Override
    public void onInflate(Activity activity, AttributeSet attrs, Bundle savedInstanceState) {
        super.onInflate(activity, attrs, savedInstanceState);
        TypedArray a = activity.obtainStyledAttributes(attrs, R.styleable.FragmentArguments);
        CharSequence cs = a.getText(R.styleable.FragmentArguments_android_label);
        if (cs!=null)
            setText(cs.toString());
        a.recycle();
    }
}
```

Observer Pattern

- Three sets of buttons like the one on the right.
- Press + to increment the total, pres - to decrement.
- When you press any + or -, all three totals change


```
<Button
    android:id="@+id/minusButton"
    android:layout_width="50dp"
    android:layout_height="wrap_content"
    android:text="-"
    />

<TextView
    android:id="@+id/count"
    android:layout_width="30dp"
    android:layout_height="wrap_content"
    android:layout_marginLeft="10dp"
    android:layout_marginRight="10dp"
    android:text="total"
    android:gravity="center_vertical|center_horizontal" />

<Button
    android:id="@+id/plusButton"
    android:layout_width="50dp"
    android:layout_height="wrap_content"
    android:text="+"
    />
```

Observer Pattern - Business Rules - Counter

- The Counter keeps track of the current count.
- It also has an Observer interface.
- Observers can register and unregister.
- Whenever the count is updated, Observers are updated.

```
public class Counter implements Serializable {
    private int count = 0;
    private final Set<Observer> observers = new HashSet<>();

    public void update(boolean isPlus){
        if (isPlus)
            count++;
        else
            count--;

        updateListeners();
    }
    public void register(Observer observer){
        if (observers.add(observer))
            observer.onCount(count);
    }
    public void unRegister(Observer observer){
        observers.remove(observer);
    }
    private void updateListeners(){
        for(Observer observer : observers)
            observer.onCount(count);
    }
    public interface Observer {
        public void onCount(int count);
    }
}
```

Observer Pattern - ObserverPatternActivity

- The Activity has the one and only Counter. The Fragment accesses the Counter through the MinusPlusButtonInterface

- The Activity either creates a new Counter or pulls it from the savedInstanceState.

```
@Override
public void onMinusButton(View view) { counter.update(false); }

@Override
public void onPlusButton(View view) { counter.update(true); }
```

```
public interface MinusPlusButtonInterface {
    public void onMinusButton(View view);
    public void onPlusButton(View view);
}
```

```
public class ObserverPatternActivity extends Activity
implements Counter.Observer, MinusPlusButtonInterface{
    private static final String COUNTER_KEY = "COUNTER_KEY";
    private TextView count;
    private Counter counter = null;
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.observer_pattern);
    Button minusButton = (Button) findViewById(R.id.minusButton);
    Button plusButton = (Button) findViewById(R.id.plusButton);
    count = (TextView) findViewById(R.id.count);

    if (savedInstanceState != null) {
        counter = (Counter) savedInstanceState.getSerializable(COUNTER_KEY);
    } else {
        counter = new Counter();
        ObserverPatternFragment fragment = new ObserverPatternFragment();
        getFragmentManager()
                .beginTransaction()
                .add(R.id.right, fragment)
                .commit();
    }
    minusButton.setOnClickListener((view) -> { onMinusButton(view); });

    plusButton.setOnClickListener((view) -> { onPlusButton(view); });

}
```

Observer Pattern - ObserverPatternActivity

- The Activity and both Fragments are registered / unregistered during the onResume / onPause lifecycle events

```
@Override
protected void onPause() {
    super.onPause();
    counter.unRegister(this);
    counter.unRegister(
            (Counter.Observer)
            getFragmentManager()
            .findFragmentById(R.id.left));
    counter.unRegister(
            (Counter.Observer)
            getFragmentManager()
            .findFragmentById(R.id.right));
}

@Override
protected void onResume() {
    super.onResume();
    counter.register(this);
    counter.register(
            (Counter.Observer)
            getFragmentManager()
            .findFragmentById(R.id.left));
    counter.register(
            (Counter.Observer)
            getFragmentManager()
            .findFragmentById(R.id.right));
}
```

Observer Pattern - ObserverPatternFragment

- The minus and plus buttons delegate their responsibility to the Activity

```
public class ObserverPatternFragment extends Fragment
implements Counter.Observer {
    private Button minusButton, plusButton;
    private TextView count;
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle sis) {
        super.onCreateView(inflater, container, sis);
        View view = inflater.inflate(R.layout.minus_text_plus, container, false);
        minusButton = (Button) view.findViewById(R.id.minusButton);
        plusButton = (Button) view.findViewById(R.id.plusButton);
        count = (TextView) view.findViewById(R.id.count);

        minusButton.setOnClickListener((view) -> {
                ((MinusPlusButtonInterface)getActivity()).onMinusButton(view);
        });

        plusButton.setOnClickListener((view) -> {
                ((MinusPlusButtonInterface)getActivity()).onPlusButton(view);
        });

        return view;
    }

    @Override
    public void onCount(int count) { this.count.setText(""+count); }
}
```

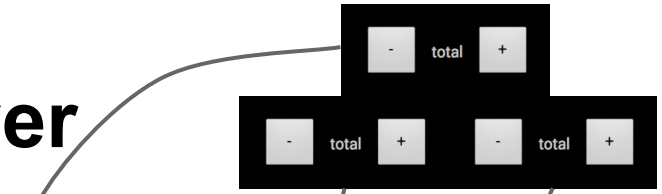
Local Broadcast Receiver

- Similar to the Observer Pattern
- Android takes care of delivering the messages.
- Any Object can participate, the just need to have a BroadcastReceiver.
- Our example is based on the Observer Pattern example.

```
<Button
    android:id="@+id/minusButton"
    android:layout_width="50dp"
    android:layout_height="wrap_content"
    android:text="-"
    />

<TextView
    android:id="@+id/count"
    android:layout_width="30dp"
    android:layout_height="wrap_content"
    android:layout_marginLeft="10dp"
    android:layout_marginRight="10dp"
    android:text="total"
    android:gravity="center_vertical|center_horizontal" />

<Button
    android:id="@+id/plusButton"
    android:layout_width="50dp"
    android:layout_height="wrap_content"
    android:text="+"
    />
```

Local Broadcast Receiver

- There are three sets of buttons.
- The two Fragments on the bottom also <include> the minus_text_plus layout.
- The fragment on the left is static and the one on the right is dynamic.




```
<include layout="@layout/minus_text_plus"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="1"
    />

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="1"
    android:orientation="horizontal"
    >
    <fragment android:name="andevcon14.FragmentCo
        android:id="@+id/left"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1"
        />
    <FrameLayout
        android:id="@+id/right"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1"
        />
</LinearLayout>
```

Local Broadcast Receiver - Business Rules

- Very simple business rules
- An integer counter that can be incremented and decremented.

```
public class Counter implements Serializable {
    private int count = 0;

    public int getCount() { return count; }

    public void update(boolean isPlus){
        if (isPlus)
            count++;
        else
            count--;
    }
}
```

Local Broadcast CounterBroadcastHelper

- We have two Actions: ACTION_REPORT and ACTION_UPDATE
- ACTION_REPORT has an EXTRA_COUNT that reports the current count.
- ACTION_UPDATE has an EXTRA_IS_PLUS that tells the receiver that the update is in the plus direction (or not).

```
public class CounterBroadcastHelper implements Serializable {
    private final static String ACTION_REPORT = "ACTION_REPORT";
    private final static String ACTION_UPDATE = "ACTION_UPDATE";
    public final static String EXTRA_COUNT = "EXTRA_COUNT";
    public final static String EXTRA_IS_PLUS = "EXTRA_IS_PLUS";

    private static CounterBroadcastHelper instance = new CounterBroadcastHelper();

    private CounterBroadcastHelper(){}

    public static CounterBroadcastHelper getInstance() { return instance; }

    public IntentFilter getReportIntentFilter() { return new IntentFilter(ACTION_REPORT); }

    public IntentFilter getUpdateIntentFilter() { return new IntentFilter(ACTION_UPDATE); }

    public void broadcastCount(Context context, int count){
        Intent intent = new Intent(ACTION_REPORT);
        intent.putExtra(EXTRA_COUNT, count);
        LocalBroadcastManager.getInstance(context)
                .sendBroadcast(intent);
    }

    public void broadcastUpdate(Context context, boolean isPlus){
        Intent intent = new Intent(ACTION_UPDATE);
        intent.putExtra(EXTRA_IS_PLUS, isPlus);
        LocalBroadcastManager.getInstance(context)
                .sendBroadcast(intent);
    }
}
```

Local Broadcast CounterBroadcastHelper

- CounterBroadcastHelper has two sets of functions; getXIntentFilter and broadcastX
- The getXIntentFilter functions return new Intent Filters ready to receive the selected type of Intent
- The broadcastX functions tell any listeners about the new info they're interested in.

```
public class CounterBroadcastHelper implements Serializable {
    private final static String ACTION_REPORT = "ACTION_REPORT";
    private final static String ACTION_UPDATE = "ACTION_UPDATE";
    public final static String EXTRA_COUNT = "EXTRA_COUNT";
    public final static String EXTRA_IS_PLUS = "EXTRA_IS_PLUS";

    private static CounterBroadcastHelper instance = new CounterBroadcastHelper();

    private CounterBroadcastHelper(){}

    public static CounterBroadcastHelper getInstance() { return instance; }

    public IntentFilter getReportIntentFilter() { return new IntentFilter(ACTION_REPORT); }

    public IntentFilter getUpdateIntentFilter() { return new IntentFilter(ACTION_UPDATE); }

    public void broadcastCount(Context context, int count){
        Intent intent = new Intent(ACTION_REPORT);
        intent.putExtra(EXTRA_COUNT, count);
        LocalBroadcastManager.getInstance(context)
                .sendBroadcast(intent);
    }

    public void broadcastUpdate(Context context, boolean isPlus){
        Intent intent = new Intent(ACTION_UPDATE);
        intent.putExtra(EXTRA_IS_PLUS, isPlus);
        LocalBroadcastManager.getInstance(context)
                .sendBroadcast(intent);
    }
}
```

LocalBroadcastActivity

- LocalBroadcastActivit has-a Counter and two BroastReceivers

- Counter is where our business logic lives. It's created in the onCreate function, and since it's Serializable, it's state is stored during a confiuration change.

- The BroadcastReceivers receive notice when the value of the Counter changes or when some other object wants to change the value of the Coutner.

```
public class LocalBroadcastActivity extends FragmentActivity {
    private static final String COUNTER_KEY = "COUNTER_KEY";
    private Counter counter = null;
    private BroadcastReceiver countReceiver, updateReceiver;
```

```
if (savedInstanceState != null) {
    counter = (Counter) savedInstanceState.getSerializable(COUNTER_KEY);
} else {
    counter = new Counter();
```

```
@Override
protected void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);
    outState.putSerializable(COUNTER_KEY, counter);
}
```

LocalBroadcastActivity

- The onCreate button sets up the minus and plus buttons. Like the Fragments, they just broadcast the update.

- Like the Fragment, there is a countReceiver that pulls the current count from transmitted by the update.

- the updateReceiver actually executes the business rule then transmits the result.

```
minusButton.setOnClickListener((view) -> {
        CounterBroadcastHelper.getInstance()
                .broadcastUpdate(LocalBroadcastActivity.this, false);
});

plusButton.setOnClickListener((view) -> {
        CounterBroadcastHelper.getInstance()
                .broadcastUpdate(LocalBroadcastActivity.this, true);
});

countReceiver = (BroadcastReceiver) (context, intent) -> {
        int newCount = intent.getIntExtra(CounterBroadcastHelper.EXTRA_COUNT, 0);
        count.setText("" + newCount);
};

updateReceiver = (BroadcastReceiver) (context, intent) -> {
        boolean isPlus = intent.getBooleanExtra(CounterBroadcastHelper.EXTRA_IS_PLUS,true);
        counter.update(isPlus);
        CounterBroadcastHelper.getInstance()
                .broadcastCount(LocalBroadcastActivity.this,counter.getCount());
};
```

- the updateReceiver above is the *only* place in the code where the business rule is called.

LocalBroadcastActivity

- Both BroadcastReceivers are registered in onResume and unregistered in onPause.
- onPostResume is called after all onResume functions in the Activity and all Fragments are called.
- It's a convenient place to send out the initial Broadcast of the current count so the UI starts with the correct information.

```
@Override
protected void onPause() {
    super.onPause();
    LocalBroadcastManager.getInstance(this)
            .unregisterReceiver(countReceiver);
    LocalBroadcastManager.getInstance(this)
            .unregisterReceiver(updateReceiver);
}

@Override
protected void onResume() {
    super.onResume();
    LocalBroadcastManager.getInstance(this)
            .registerReceiver(
                    countReceiver,
                    CounterBroadcastHelper.getInstance().getReportIntentFilter());
    LocalBroadcastManager.getInstance(this)
            .registerReceiver(
                    updateReceiver,
                    CounterBroadcastHelper.getInstance().getUpdateIntentFilter());
}

@Override
protected void onPostResume() {
    super.onPostResume();
    CounterBroadcastHelper.getInstance()
            .broadcastCount(this, counter.getCount());
}
```

LocalBroadcastFragment

- LocalBroadcastFragment has-a countReceiver
- It's created in the onCreateView function
- It's registerd in onResume
- It's unregistered in onPause

```
public class LocalBroadcastFragment extends Fragment {
    private BroadcastReceiver countReceiver;
```

```
    countReceiver = (BroadcastReceiver) (context, intent) -> {
            int newCount = intent.getIntExtra(CounterBroadcastHelper.EXTRA_COUNT, 0);
            count.setText(""+newCount);
    };
```

```
@Override
public void onPause() {
    super.onPause();
    LocalBroadcastManager.getInstance(getActivity())
            .unregisterReceiver(countReceiver);
}

@Override
public void onResume() {
    super.onResume();
    LocalBroadcastManager.getInstance(getActivity())
            .registerReceiver(
                    countReceiver,
                    CounterBroadcastHelper.getInstance().getReportIntentFilter());
}
```

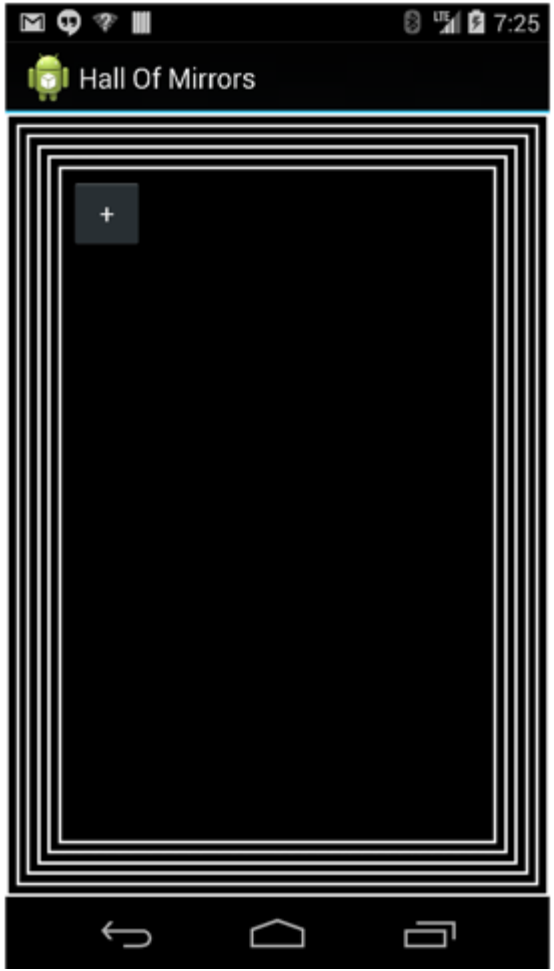
LocalBroadcastFragment

- onCreateView also sets up the minus and plus Buttons
- They use the broadcastUpdate to tell the owner of the Counter to update using the broadcastUpdate

```
minusButton.setOnClickListener((view) -> {
        CounterBroadcastHelper.getInstance()
                .broadcastUpdate(getActivity(), false);
});

plusButton.setOnClickListener((view) -> {
        CounterBroadcastHelper.getInstance()
                .broadcastUpdate(getActivity(), true);
});
```

ChildFragmentManager

- Fragments can be nested to about eight levels
- After that a StackOverflowError occurs
- When Activities add Fragments, use getFragmentManager / getSupportFragmentManager
- When Fragments add Fragments, use getChildFragmentManager / getSupportChildFragmentManager
- See the Hall of Mirrors example



Hall of Mirrors Activity

- Single Activity: HallOfMirrorsActivity
- onCreate: setContentView to the hom_activity layout
- uses getFragmentManager to insert a MirrorFragment into the R.id.frame FrameLayout

```
public class HallOfMirrorsActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.hom_activity);

        if (savedInstanceState == null) {
            FragmentTransaction ft = getFragmentManager().beginTransaction();
            MirrorFragment newFragment = MirrorFragment.newInstance();
            ft.add(R.id.frame, newFragment);
            ft.commit();
        }
    }
}
```

```
<FrameLayout
    android:id="@+id/frame"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    />
```

Hall of Mirrors Fragment

- Nesting fragment, MirrorFragment, has a static factory newInstance method.
- getChildFragmentManager used to add a new MirrorFragment into the R.id.frame FrameLayout

```
public class MirrorFragment extends Fragment {

    public static MirrorFragment newInstance(){
        return new MirrorFragment();
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle sis) {
        super.onCreateView(inflater, container, sis);
        final View view = inflater.inflate(R.layout.hom_fragment, container, false);
        final View plusButton = view.findViewById(R.id.plusButton);
        final View border = view.findViewById(R.id.border);
        border.setVisibility(View.GONE);
        plusButton.setOnClickListener((v) -> {
                plusButton.setVisibility(View.GONE);
                border.setVisibility(View.VISIBLE);
                FragmentTransaction ft = getChildFragmentManager().beginTransaction();
                MirrorFragment newFragment = MirrorFragment.newInstance();
                ft.add(R.id.frame, newFragment);
                ft.commit();
        });
        return view;
    }
}
```

Hall of Mirrors: Frag layout

- plusButton: pressed to make the nested border/frame visible.
- The LinearLayout named “border” is what puts the border around the frame.
- The thickness of the border is controlled by the layout_margin of the FrameLayout.
- The space between the borders is controlled by the padding of the FrameLayout.

```
<LinearLayout xmlns:android="http://schemas.android.
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    >

    <Button
        android:id="@+id/plusButton"
        android:layout_width="50dp"
        android:layout_height="wrap_content"
        android:text="+"
        />
    <LinearLayout
        android:id="@+id/border"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:background="#fff"
        >
        <FrameLayout
            android:id="@+id/frame"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_margin="2dp"
            android:padding="5dp"
            android:background="#000"
            />
    </LinearLayout>
</LinearLayout>
```

Native vs Support Library

Native API

- Benefits
  - Smaller App - Fragment code is part of the OS
  - Faster App load time
- Drawbacks
  - Unavailable before API 11
  - Fragment has bugs causing NPEs in API 11-13

Support Library

- Benefits
  - Available for (almost) all API Levels
  - Always latest features
  - Always latest bug fixes
- Drawbacks
  - Small increase in app size
  - Tiny increase in app load time

Conclusion: First choice is Support Library unless there is a really compelling reason not to

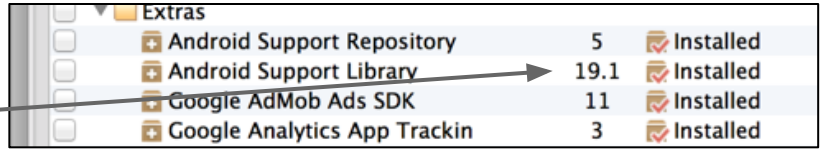
Converting from Native to Support Library

- The entire fragment-logged sample was converted to use the support library in about 15 minutes.

- The result is the fragment-logged-support project.

Converting from Native to Support Library

- Make sure the Android Support Library is installed in your SDK.
- Add the dependency to build.gradle
- Change the compileSdkVersion to match the Android API version you're targeting




```
apply plugin: 'android'

repositories {
    mavenCentral()
}

dependencies {
    compile 'com.android.support:support-v4:19.0.+'
}

android {
    compileSdkVersion 10
    buildToolsVersion "19.0.2"
}
```

Converting from Native to Support Library

- AndroidManifest.xml: Modify the minSdkVersion and targetSdkVersion to match the Android API level you're targeting

- Within your code, all instances of "import android.app.Activity" change to "import android.support.v4.app.FragmentActivity" for all Activities that host Fragments.

- Also, all "import android.app.Fragment" change to "import android.support.v4.app.Fragment"

- All calls to getFragmentManager change to getSupportFragmentManager. Also: getChildFragmentManager -> getSupportChildFragmentManager.

Recap

We looked at a LOT of code

- How a Fragment differs from an Activity.
- The Fragment Lifecycle.
- Communication between Activities & Fragments
- Nesting Fragments using the ChildFragmentManager
- Using the Support Library

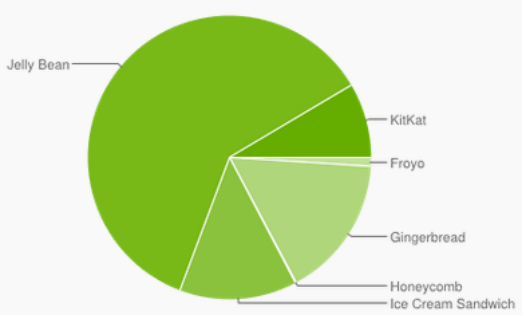
Questions

Supporting Slides

API Levels

Version	Codename	API	Distribution
2.2	Froyo	8	1.0%
2.3.3 - 2.3.7	Gingerbread	10	16.2%
3.2	Honeycomb	13	0.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	13.4%
4.1.x	Jelly Bean	16	33.5%
4.2.x		17	18.8%
4.3		18	8.5%
4.4	KitKat	19	8.5%

Data collected during a 7-day period ending on May 1, 2014.



Native Fragments work →

Native Fragments introduced →

Support Fragments →

Platform Version	API Level	VERSION_CODE
Android 4.4	19	KITKAT
Android 4.3	18	JELLY_BEAN_MR2
Android 4.2, 4.2.2	17	JELLY_BEAN_MR1
Android 4.1, 4.1.1	16	JELLY_BEAN
Android 4.0.3, 4.0.4	15	ICE_CREAM_SANDWICH_MR1
Android 4.0, 4.0.1, 4.0.2	14	ICE_CREAM_SANDWICH
Android 3.2	13	HONEYCOMB_MR2
Android 3.1.x	12	HONEYCOMB_MR1
Android 3.0.x	11	HONEYCOMB
Android 2.3.4 Android 2.3.3	10	GINGERBREAD_MR1
Android 2.3.2 Android 2.3.1 Android 2.3	9	GINGERBREAD
Android 2.2.x	8	FROYO
Android 2.1.x	7	ECLAIR_MR1
Android 2.0.1	6	ECLAIR_0_1
Android 2.0	5	ECLAIR
Android 1.6	4	DONUT
Android 1.5	3	CUPCAKE
Android 1.1	2	BASE_1_1
Android 1.0	1	BASE