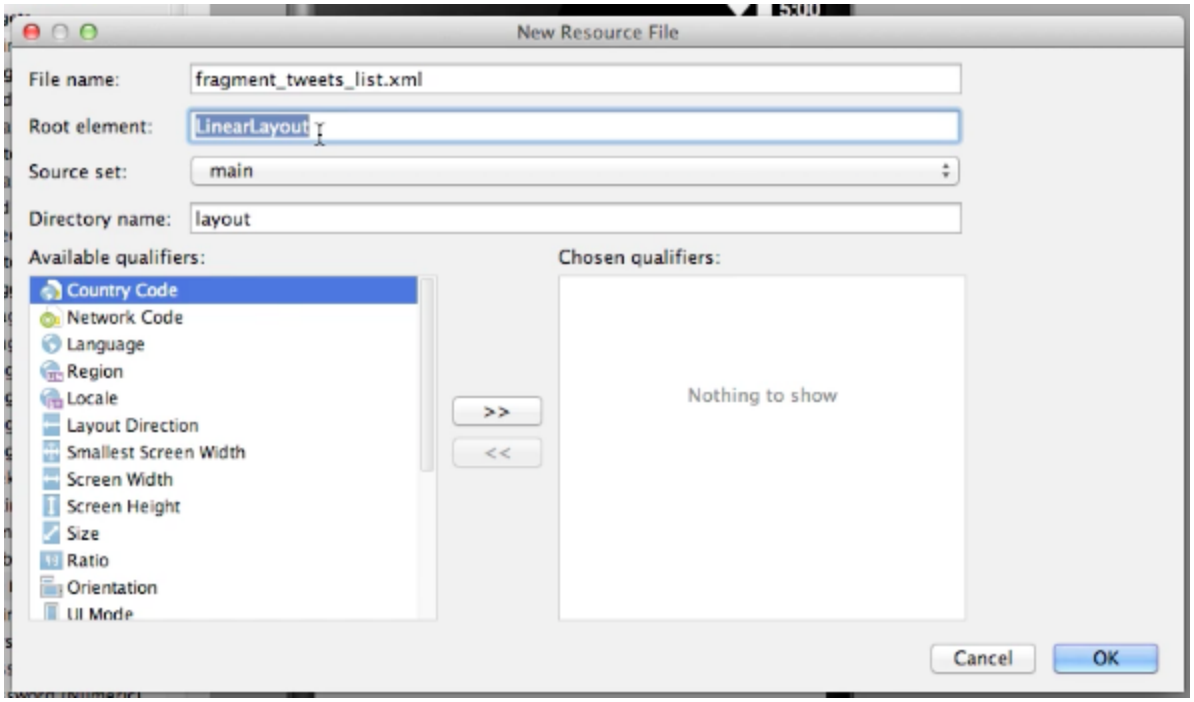
CPOG Week 4 Video Walkthrough Notes
CodePath Observer Group
Enhanced Twitter App.
By Chuck Huie, vidscmd@netscape.net
March 2015

Start with the app description.
00:01:42 - Fragments description - the ability to create components containing views and logic that act as part of an activity. Like a mini activity with it's own view, logic, life cycle, etc.
Activities can have multiple fragments and fragments can be reused by multiple activities.

00:03:34 - Move what we had before into fragments.
00:03:41 - Create a xml for the fragment.
Right click app/res/layout package. Select New/Layout resource file. File name: fragment_tweets_list.xml. Root element: LinearLayout



Now move what we had in the activity_timeline.xml into our fragment xml. Move the list view from the activity into the fragment, fragment_tweets_list.xml.

```
<ListView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/lvTweets"
    android:layout_alignParentTop="true"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:layout_alignParentBottom="true"
    android:layout_alignParentRight="true"
    android:layout_alignParentEnd="true" />
```

Remove some unnecessary properties and we end up with this:.

```
tweets_list.xml
activity_timeline.xml × fragment_tweets_list.xml ×
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      android:orientation="vertical" android:layout_width="match_parent"
4      android:layout_height="match_parent">
5
6      <ListView
7          android:layout_width="wrap_content"
8          android:layout_height="wrap_content"
9          android:id="@+id/lvTweets" />
10
11 </LinearLayout>
```

Next we want to create a new package for our fragment.
Right click on app/java/com… select New/Package, name it ‘fragments’
Next we create the fragment java class to contain the inflate and other logic for this layout.
Right click on fragments select New/Java Class, name it TweetsListFragment.
Then we want to extend the the class Fragments.

```
package com.codepath.apps.mysimpletweets.fragments;

public class TweetsListFragment extends Fr
}
    Fragment (android.support.v4.app)
    FractionBuilder (android.text.style.TtsSpan)
    Fragment (android.app)
    FragmentManager (android.app)
    FragmentTransaction (android.app)
```

Make sure to select the android.support.v4.app fragment.
The fragment requires at least 2 things.
The inflation logic
The creation life cycle event
Add the onCreateView

```
public class TweetsListFragment extends Fragment {
    // inflation logic
    oncreate
    public void onCreate(savedInstanceState) {...}                Fragment
    public View onCreateView(inflater, container, savedIn…        Fragment
    public Animation onCreateAnimation(transit, enter, ne…        Fragment
    public void onCreateContextMenu(menu, v, menuInfo) {…         Fragment
    public void onCreateOptionsMenu(menu, inflater) {...}         Fragment
    public void onActivityCreated(savedInstanceState) {…          Fragment
    public void onViewCreated(view, savedInstanceState) {…        Fragment
Press ⌘Space to see non-imported classes >>
```

The Creation is onCreate()

```
public class TweetsListFragment extends Fragment {
    // inflation logic

    @Override
    public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup container, @Nullable Bundle savedInstanceState) {
        return super.onCreateView(inflater, container, savedInstanceState);
    }


    // creation lifecycle event
    oncreate
    public void onCreate(savedInstanceState) {...}                Fragment
    public Animation onCreateAnimation(transit, enter, ne…        Fragment
    public void onCreateContextMenu(menu, v, menuInfo) {…         Fragment
    public void onCreateOptionsMenu(menu, inflater) {...}         Fragment
    public void onActivityCreated(savedInstanceState) {…          Fragment
    public void onViewCreated(view, savedInstanceState) {…        Fragment
Press ^. to choose the selected (or first) suggestion and insert a dot afterwards >>
```

???How did he get the super.onCreate statement to occur??? It was automatically (code completion) included when onCreate() was created.

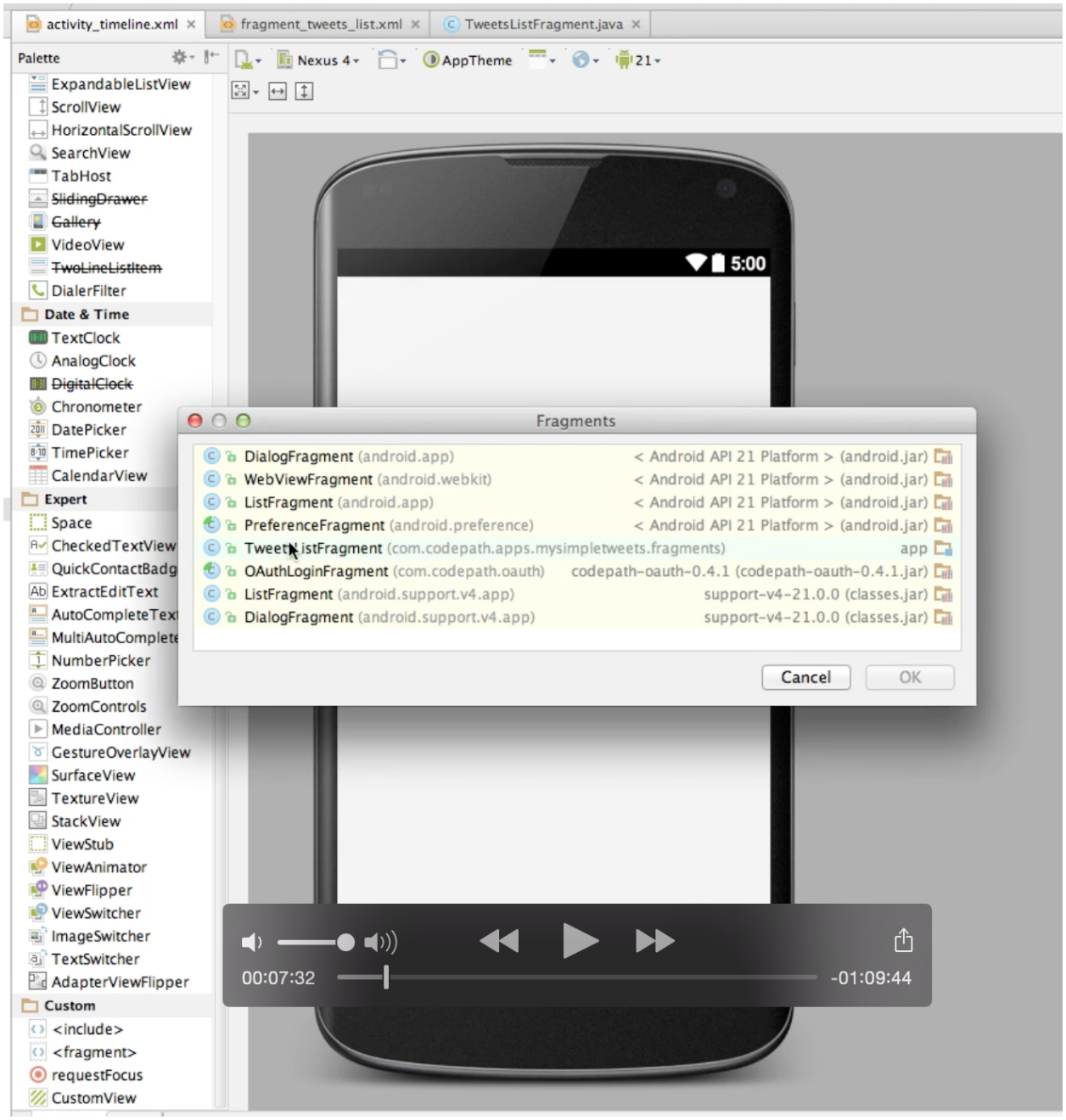
```
package com.codepath.apps.mysimpletweets.fragments;

import android.os.Bundle;
import android.support.annotation.Nullable;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class TweetsListFragment extends Fragment {
    // inflation logic

    @Override
    public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup container, @Nullable Bundle savedInstanceState) {
        return super.onCreateView(inflater, container, savedInstanceState);
    }


    // creation lifecycle event

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }
}
```

00:06:00 - Now to customize our inflation logic. Modify onCreateView().
Remove the return statement in the onCreateView.
Inflate the fragment using the inflater.

```
public class TweetsListFragment extends Fragment {
    // inflation logic

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


    // creation lifecycle event

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }
}
```

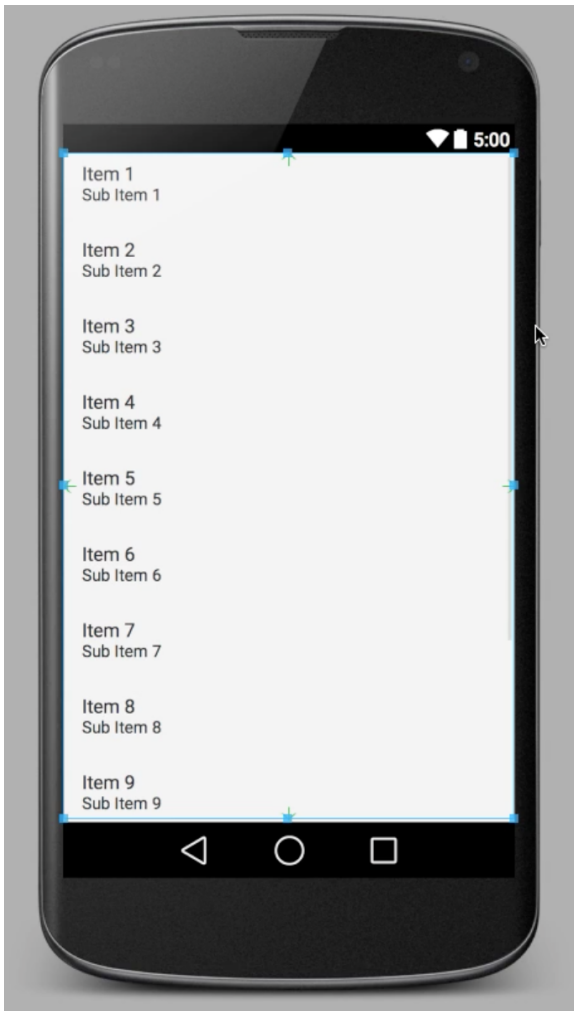
Where parent is our container and false so it doesn't have to add itself the view immediately.
Now add the fragment to our activity timeline statically for now. Before the timeline had the listview directly, now we will put the fragment in place of the listview.

00:07:00 - Modify the activity_timeline.xml
delete the listview
embed the fragment into the activity directly.
Go to the design tab for the activity_timeline.xml.
Scroll down to find <fragment> under custom. Click on fragment, select TweetListFragment, and then click



on (Use @layout/fragment_tweets_list, …) to populate the fragment view with the list.

To get this:



We want as much of the views and logic (code) into the fragment. Leave the activity to be as clean as possible so it just looks like a simple container of fragments.

00:09:00 Run to to show that we still get out list view.

00:09:24 We now want to move the code into the fragment. Note that the code is in TimelineActivity.java. We want to move it into TweetListFragment.java.

Move our variables from TimelineActivity.java:

```
public class TimelineActivity extends ActionBarActivity {

    private TwitterClient client;
    private ArrayList<Tweet> tweets;
    private TweetsArrayAdapter aTweets;
    private ListView lvTweets;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

Into TweetListFragment.java:

```
public class TweetsListFragment extends Fragment {
    private ArrayList<Tweet> tweets;
    private TweetsArrayAdapter aTweets;
    private ListView lvTweets;

    // inflation logic
    @Override
```

Move their usage as well, from TimelineActivity.java:t

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_timeline);
        // Find the listview
        lvTweets = (ListView) findViewById(R.id.lvTweets);
        // Create the arraylist (data source)
        tweets = new ArrayList<>();
        // Construct the adapter from data source
        aTweets = new TweetsArrayAdapter(this, tweets);
        // Connect adapter to list view
        lvTweets.setAdapter(aTweets);
        // Get the client
        client = TwitterApplication.getRestClient(); // single
        populateTimeline();
    }
```

Into TweetListFragment.java:
For variables that references the view:
Note that we are no longer in an activity, but a fragment here so we reference the View v to access the list view:

```
    // inflation logic
    @Override
    public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup parent, @Nullable Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_tweets_list, parent, false);
        // Find the listview
        lvTweets = (ListView) v.findViewById(R.id.lvTweets);
        // Connect adapter to list view
        lvTweets.setAdapter(aTweets);
        return v;
    }
```

For variables that do not need to reference, we move them into onCreate():
Note since we're not in an activity any longer, we cannot use 'this' in aTweets assignment, but use the function getActivity() to access the context:

```
// creation lifecycle event
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // Create the arraylist (data source)
    tweets = new ArrayList<>();
    // Construct the adapter from data source
    aTweets = new TweetsArrayAdapter(getActivity(), tweets);
}
```

00:11:23 We still want to make the network call from the activity, but do not have access to the adapter aTweets. So we want to expose the adapter for us here. But instead of adding a method in the fragment (TweetListFragment.java) to return the adapter to us for use in the TimelineActivity.java, a better idea is to talk to the fragment and have the fragment take care of inserting data into the list.
So let's create an addAll() method into our TweetListFragment.java file instead:

```
// creation lifecycle event
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // Create the arraylist (data source)
    tweets = new ArrayList<>();
    // Construct the adapter from data source
    aTweets = new TweetsArrayAdapter(getActivity(), tweets);
}

public void addAll(List<Tweet> tweets) {
    aTweets.addAll(tweets);
}
```

00:12:48 To call this new addAll method from TimelineActivity.java, we'll need access to the fragment. We create a new variable, fragmentTweetsList, to reference our fragment in TimelineActivity.java:

```
public class TimelineActivity extends ActionBarActivity {

    private TweetsListFragment fragmentTweetsList;
    private TwitterClient client;

    @Override
```

We'll want to access the fragment directly from the activity_timeline.xml, so we'll change the fragment from "@+id/fragment":

activity_timeline.xml | fragment_tweets_list.xml | TweetsListFragment.java | TimelineActivity.java

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


    <fragment
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:name="com.codepath.apps.mysimpletweets.fragments.TweetsListFragment"
        android:id="@+id/fragment"
        android:layout_alignParentTop="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        tools:layout="@layout/fragment_tweets_list"
        android:layout_alignParentBottom="true"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true" />

</RelativeLayout>
```

To "@+id/fragment_timeline":

```
<fragment
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:name="com.codepath.apps.mysimpletweets.fragments.TweetsListFragment"
    android:id="@+id/fragment_timeline"
    android:layout_alignParentTop="true"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    tools:layout="@layout/fragment_tweets_list"
    android:layout_alignParentBottom="true"
    android:layout_alignParentRight="true"
    android:layout_alignParentEnd="true" />
```

To access this particular fragment from our TimelineActivity.java, we add a line to onCreate():

Note that we don't use findViewbyID since we're not accessing a view, but a fragment which is a complicated object with its own set of controls. We use getSupportFragmentManager() to get the fragment.
This is how we statically pull a fragment from a layout.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_timeline);
    // Get the client
    client = TwitterApplication.getRestClient(); // singleton client
    populateTimeline();
    // access the fragment
    fragmentTweetsList = (TweetsListFragment) getSupportFragmentManager().findFragmentById(R.id.fragment_timeline);
}
```

But this is usually done inside a check for null like so:

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_timeline);
    // Get the client
    client = TwitterApplication.getRestClient(); // singleton client
    populateTimeline();
    if (savedInstanceState == null) {
        // access the fragment
        fragmentTweetsList = (TweetsListFragment) getSupportFragmentManager().findFragmentById(R.id.fragment_timeline);
    }
}
```

This is because if the saveInstanceState is not null, this activity has already been inflated in the past and the code has already been run and the fragment is already in memory.

Lastly, we want to change aTweets in line 51 of our populateTimeline() method to use fragmentTweetsList:

```
39      // Send an API request to get the timeline json
40      // Fill the listview by creating the tweet objects from the json
41      private void populateTimeline() {
42          client.getHomeTimeline(new JsonHttpResponseHandler() {
43              // SUCCESS
44              @Override
45              public void onSuccess(int statusCode, Header[] headers, JSONArray json) {
46                  Log.d("DEBUG", json.toString());
47                  // DESERIALIZE JSON
48                  // CREATE MODELS AND ADD THEM TO THE ADAPTER
49                  // LOAD THE MODEL DATA INTO LISTVIEW
50
51                  fragmentTweetsList.addAll(Tweet.fromJSONArray(json));
52              }
53
54              // FAILURE
55              @Override
56              public void onFailure(int statusCode, Header[] headers, Throwable throwable, JSONObject errorResponse) {
57                  Log.d("DEBUG", errorResponse.toString());
58              }
59          });
```

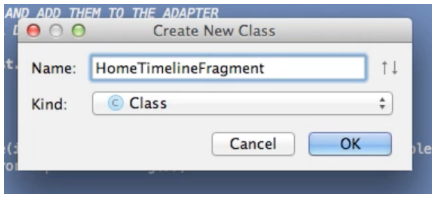
00:16:00 Summary for code change from TimelineActivity.java to TweetListFragment.java.

00:16:15 Run to see if we still get our timeline.

00:16:40 Noticed that the loading and the populate timeline were not moved to the fragment. We really don't want to move them to our TweetsListFragment as we want to use it as our base list fragment so we can also use it later with our mentions fragment. So we'll create a new fragment that builds on top of the TweetsListFragment.

00:17:38 Let's create a HomeTimelineFragment for use with our existing home list. Right click on the fragment folder and select New/Java Class/

Name the new class, ‘HomeTimelineFragment’.



Edit HomeTimelineFragment.java so that the new class extends the TweetsListFragment instead of the standard Fragment.

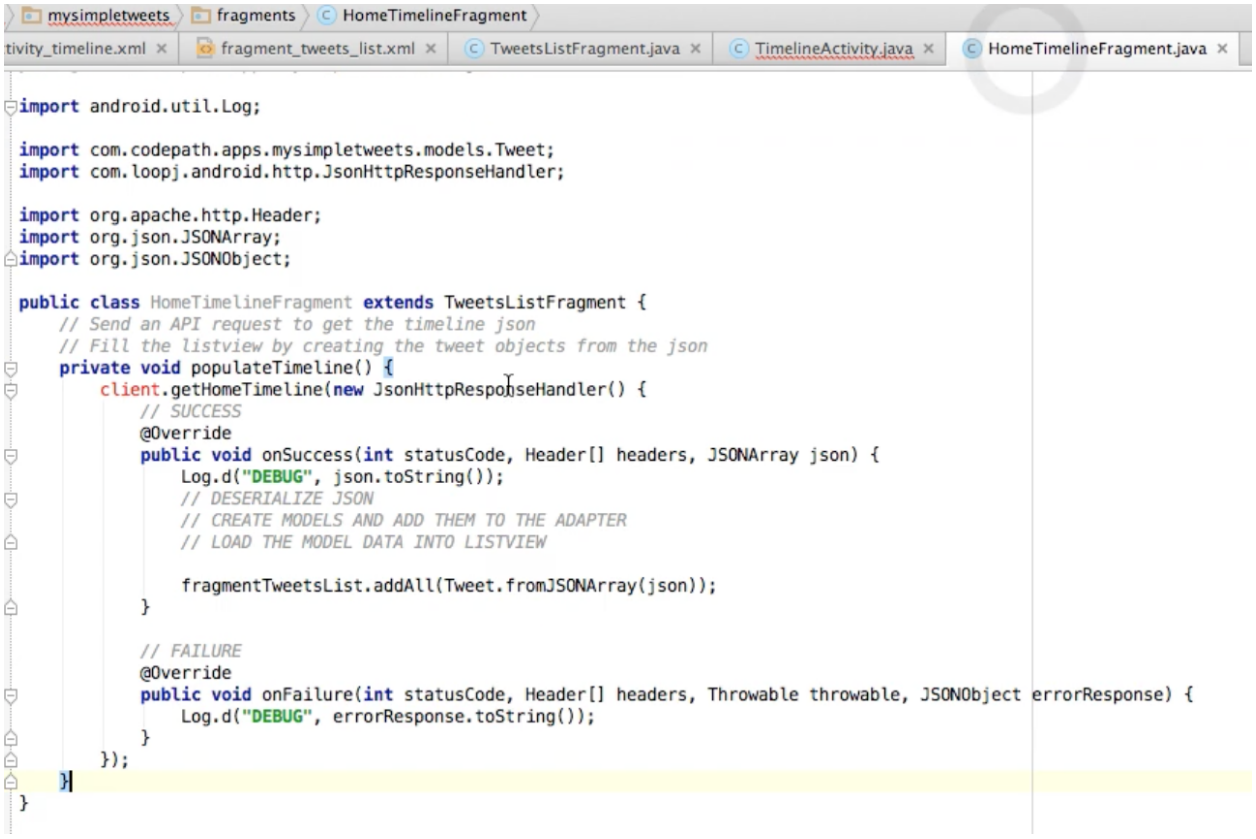
```
package com.codepath.apps.mysimpletweets.fragments;

public class HomeTimelineFragment extends TweetsListFragment {

}
```

Now we want to move the rest of the display list code from TimelineActivity to this new fragment, HomeTimelineFragment.

18:30:00 Move over the populateTimeline method from TimelineActivity:

```
    // Send an API request to get the timeline json
    // Fill the listview by creating the tweet objects from the json
    private void populateTimeline() {
        client.getHomeTimeline(new JsonHttpResponseHandler() {
            // SUCCESS
            @Override
            public void onSuccess(int statusCode, Header[] headers, JSONArray json) {
                Log.d("DEBUG", json.toString());
                // DESERIALIZE JSON
                // CREATE MODELS AND ADD THEM TO THE ADAPTER
                // LOAD THE MODEL DATA INTO LISTVIEW

                fragmentTweetsList.addAll(Tweet.fromJSONArray(json));
            }

            // FAILURE
            @Override
            public void onFailure(int statusCode, Header[] headers, Throwable throwable, JSONObject errorResponse) {
                Log.d("DEBUG", errorResponse.toString());
            }
        });
    }
```

To the HomeTimelineFragment:




```java
import android.util.Log;

import com.codepath.apps.mysimpletweets.models.Tweet;
import com.loopj.android.http.JsonHttpResponseHandler;

import org.apache.http.Header;
import org.json.JSONArray;
import org.json.JSONObject;

public class HomeTimelineFragment extends TweetsListFragment {
    // Send an API request to get the timeline json
    // Fill the listview by creating the tweet objects from the json
    private void populateTimeline() {
        client.getHomeTimeline(new JsonHttpResponseHandler() {
            // SUCCESS
            @Override
            public void onSuccess(int statusCode, Header[] headers, JSONArray json) {
                Log.d("DEBUG", json.toString());
                // DESERIALIZE JSON
                // CREATE MODELS AND ADD THEM TO THE ADAPTER
                // LOAD THE MODEL DATA INTO LISTVIEW

                fragmentTweetsList.addAll(Tweet.fromJSONArray(json));
            }

            // FAILURE
            @Override
            public void onFailure(int statusCode, Header[] headers, Throwable throwable, JSONObject errorResponse) {
                Log.d("DEBUG", errorResponse.toString());
            }
        });
    }
}
```

We also need to move over the client from the TimelineActivity:

```java
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_timeline);
    // Get the client
    client = TwitterApplication.getRestClient(); // singleton client
    populateTimeline();
    if (savedInstanceState == null) {
        // access the fragment
        fragmentTweetsList = (TweetsListFragment) getSupportFragmentManager(
```

To our HomeTimelineFragment:
We'll need a onCreate for this and must add the super.onCreate much like we did for our activity.
Just type onCreate and super.onCreate and let code completion do the rest. Clean up code as necessary.
Note we also need to include the private variable 'client' delcaration:

```java
public class HomeTimelineFragment extends TweetsListFragment {
    private TwitterClient client;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // Get the client
        client = TwitterApplication.getRestClient(); // singleton client
        populateTimeline();
    }
```

We need to clean up the populateTimeline method then to just call addAll() on ourselves instead of fragmentTweetsList.addAll().

```
// Send an API request to get the timeline json
// Fill the listview by creating the tweet objects from the json
private void populateTimeline() {
    client.getHomeTimeline(new JsonHttpResponseHandler() {
        // SUCCESS
        @Override
        public void onSuccess(int statusCode, Header[] headers, JSONArray json) {
            Log.d("DEBUG", json.toString());
            // DESERIALIZE JSON
            // CREATE MODELS AND ADD THEM TO THE ADAPTER
            // LOAD THE MODEL DATA INTO LISTVIEW
            addAll(Tweet.fromJSONArray(json));
        }

        // FAILURE
        @Override
        public void onFailure(int statusCode, Header[] headers, Throwable throwable, JSONObject errorResponse) {
            Log.d("DEBUG", errorResponse.toString());
        }
    });
}
```

To further clean up the activity, we don't need access to the fragment in our TimelineActivity:
We can delete the following lines of code to make our activity truly empty container in which we insert a fragment:

```
public class TimelineActivity extends ActionBarActivity {

    private TweetsListFragment fragmentTweetsList;

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

        if (savedInstanceState == null) {
            // access the fragment
            fragmentTweetsList = (TweetsListFragment) getSupportFragmentManager().findFragmentById(R.id.fragment_timeline);
        }
    }
```

Let's delete this line also:

```
public class TimelineActivity extends ActionBarActivity {

    private TweetsListFragment fragmentTweetsList;
```

00:20:20 Summary of our new HomeTimelineFragment.java, extends from TweetsListFragment. Recall that we also want to use the TweetsListFragment for our mentions fragment, MentionsFragment.java.

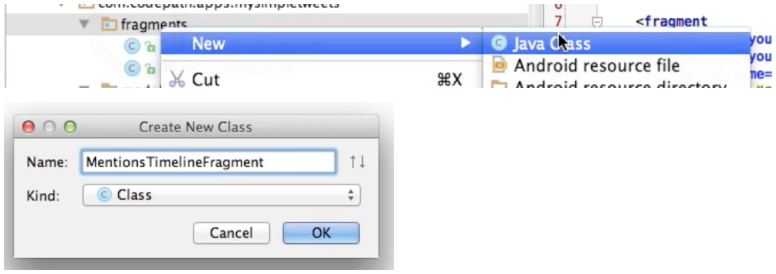
00:20:59 Before we can run and verify, we need to switch from the TweetsListFragment to the HomeTimeLineFragment. So we need to trigger the HomeTimelineFragmeent manually, as it is , the app will run the the incorrect fragment. We need to modify the activity_timeline.xml and change the name from .TweetsListFragment to .HomeTimelineFragment. We want to inflate the HomeTimelineFragment so it can be inserted into the timeline activity. The HomeTimelineFragment will trigger the network request and add the items into the base TweetsListFragment adapter.

```
<fragment
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:name="com.codepath.apps.mysimpletweets.fragments.TweetsListFragment"
    android:id="@+id/fragment_timeline"
    android:layout_alignParentTop="true"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    tools:layout="@layout/fragment_tweets_list"
    android:layout_alignParentBottom="true"
    android:layout_alignParentRight="true"
    android:layout_alignParentEnd="true" />
```

```
<fragment
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:name="com.codepath.apps.mysimpletweets.fragments.HomeTimelineFragment"
    android:id="@+id/fragment_timeline"
    android:layout_alignParentTop="true"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    tools:layout="@layout/fragment_tweets_list"
    android:layout_alignParentBottom="true"
    android:layout_alignParentRight="true"
    android:layout_alignParentEnd="true" />
```

00:21:46 **Run** and verify.

00:22:10 Building the Mentions Timeline.
00:22:17



00:22:42 Copy the code from the HomeTimeLine to the MentionsTimeline:

```
public class HomeTimelineFragment extends TweetsListFragment {
    private TwitterClient client;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // Get the client
        client = TwitterApplication.getRestClient(); // singleton client
        populateTimeline();
    }

    // Send an API request to get the timeline json
    // Fill the listview by creating the tweet objects from the json
    private void populateTimeline() {
        client.getHomeTimeline(new JsonHttpResponseHandler() {
            // SUCCESS
            @Override
            public void onSuccess(int statusCode, Header[] headers, JSONArray json) {
                // DESERIALIZE JSON
                // CREATE MODELS AND ADD THEM TO THE ADAPTER
                // LOAD THE MODEL DATA INTO LISTVIEW
                addAll(Tweet.fromJSONArray(json));
            }

            // FAILURE
            @Override
            public void onFailure(int statusCode, Header[] headers, Throwable throwable, JSONObject errorResponse) {
                Log.d("DEBUG", errorResponse.toString());
            }
        });
    }
}
```

To the MentionsTimelineFragment:

```
public class MentionsTimelineFragment {
    private TwitterClient client;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // Get the client
        client = TwitterApplication.getRestClient(); // singleton client
        populateTimeline();
    }

    // Send an API request to get the timeline json
    // Fill the listview by creating the tweet objects from the json
    private void populateTimeline() {
        client.getHomeTimeline(new JsonHttpResponseHandler() {
            // SUCCESS
            @Override
            public void onSuccess(int statusCode, Header[] headers, JSONArray json) {
                // DESERIALIZE JSON
                // CREATE MODELS AND ADD THEM TO THE ADAPTER
                // LOAD THE MODEL DATA INTO LISTVIEW
                addAll(Tweet.fromJSONArray(json));
            }

            // FAILURE
            @Override
            public void onFailure(int statusCode, Header[] headers, Throwable throwable, JSONObject errorResponse) {
                Log.d("DEBUG", errorResponse.toString());
            }
        });
    }
}
```

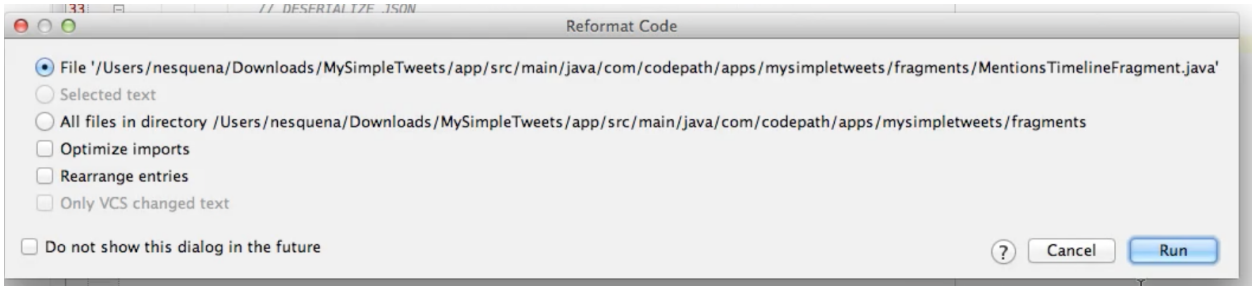
Of course we want to extend this from the TweetListFragment:

```
public class MentionsTimelineFragment extends TweetsListFragment {
    private TwitterClient client;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

We also want to call the getMentionsTimeline now, not the getHomeTimeline:

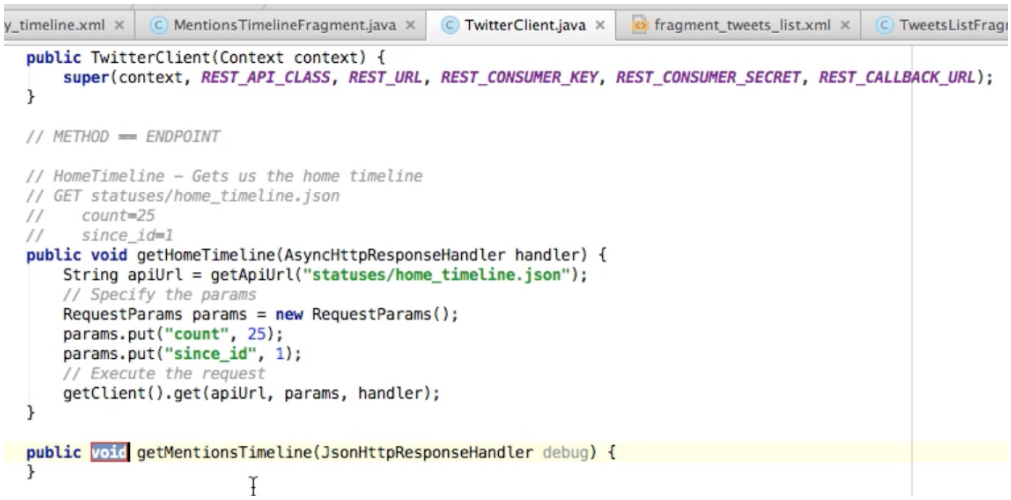
```
public class MentionsTimelineFragment extends TweetsListFragment {
    private TwitterClient client;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // Get the client
        client = TwitterApplication.getRestClient(); // singleton client
        populateTimeline();
    }

    // Send an API request to get the timeline json
    // Fill the listview by creating the tweet objects from the json
    private void populateTimeline() {
        client.getMentionsTimeline(new JsonHttpResponseHandler() {
            // SUCCESS
            @Override
            public void onSuccess(int statusCode, Header[] headers, JSONArray json) {
                // DESERIALIZE JSON
                // CREATE MODELS AND ADD THEM TO THE ADAPTER
```

We want to reformat the code:
From the menu => Code/Reformat Code:

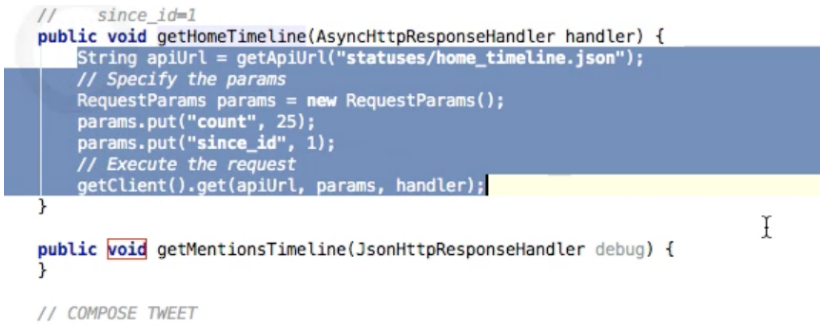


00:23:35 We need to add getMentionsTimeline to the client:
Select getMentionsTimeline and press Alt-Enter and select the Create Method option:

Which should take us here:



00:23:47 Copy the code from the getHomeTimeline to the getMentionsTimeline:



To the getMentionsTimeline:
Note the corrected parms, 'AsyncHttpResponseHandler handler':
Note the corrected endpoint 'statuses/mentions_timeline.json':
Note the removal of the unneeded put for the 'since_id':

```
public void getMentionsTimeline(AsyncHttpResponseHandler handler) {
    String apiUrl = getApiUrl("statuses/mentions_timeline.json");
    // Specify the params
    RequestParams params = new RequestParams();
    params.put("count", 25);
    // Execute the request
    getClient().get(apiUrl, params, handler);
}
```

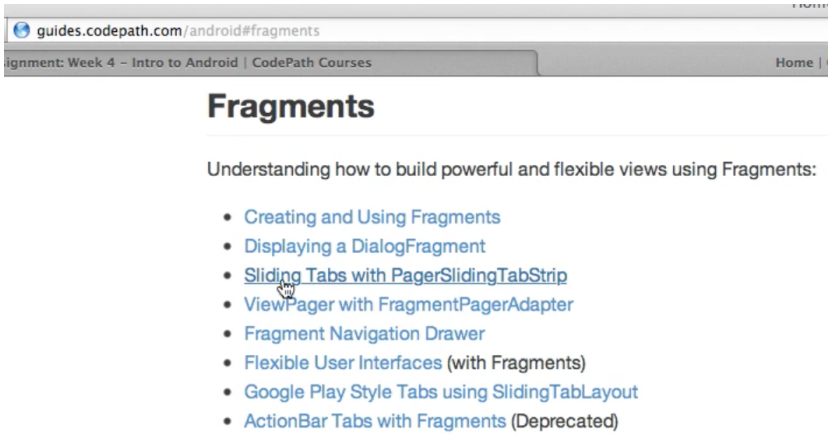
00:24:31 To test this mentions timeline, we edit the activities_timeline.xml to call our new fragment:
Change the name from ...HomeTimelineFragment to ...MentionsTimelineFragment:

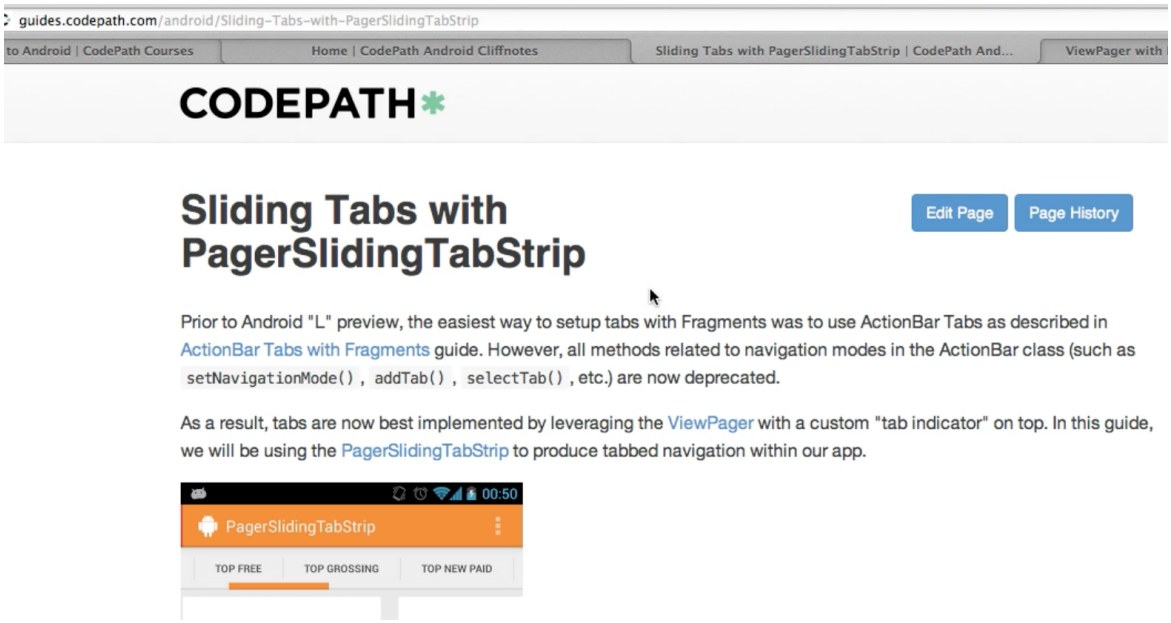
activity_timeline.xml | MentionsTimelineFragment.java | TwitterClient.java | fragment_tweets_list.xml

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


    <fragment
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:name="com.codepath.apps.mysimpletweets.fragments.MentionsTimelineFragment"
        android:id="@+id/fragment_timeline"
        android:layout_alignParentTop="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        tools:layout="@layout/fragment_tweets_list"
        android:layout_alignParentBottom="true"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true" />

</RelativeLayout>
```

00:24:48 **Run** and see the mentions timeline.

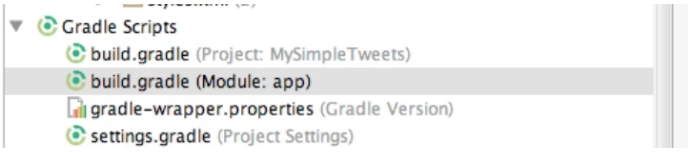
00:25:54 Learning how to use ViewPager to set up tabs. We'll use this 3rd party library to access our fragments.

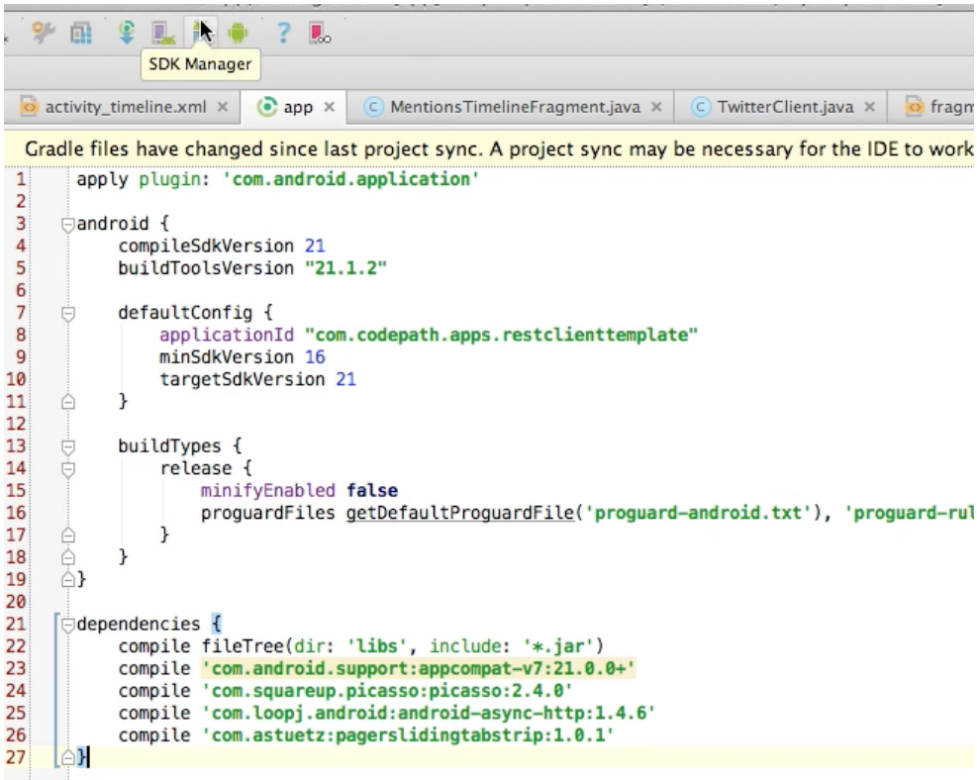




00:26:00 ViewPager library is a simple way to page or move between fragments. Allows tabbing or swiping between fragments fragments within an activity.

00:27:35 Modify the build.gradle (Module:app):
Add the compile line for the pagerslidingtabstrip module and click on sync with Gradle:





Reference: https://guides.codepath.com/android/Sliding-Tabs-with-PagerSlidingTabStrip

Install PagerSlidingTabStrip

First, we need to add `PagerSlidingTabStrip` to our application by adding the following to our `app/build.gradle` file:

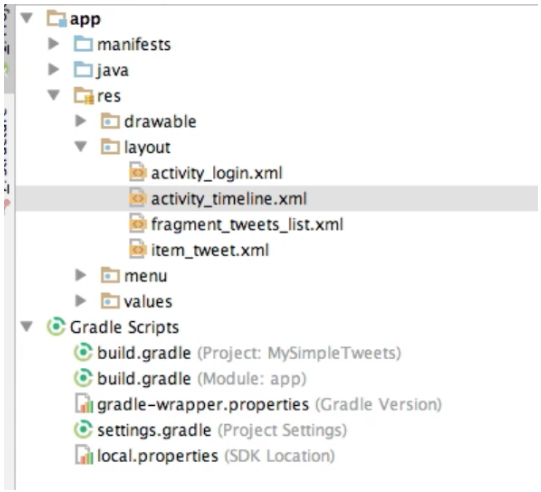
```
dependencies {
    compile 'com.astuetz:pagerslidingtabstrip:1.0.1'
}
```

Once you have included the library and synced with Gradle, we can use the `SlidingTabLayout` in our layout file to display tabs. Your layout file will have tabs on the

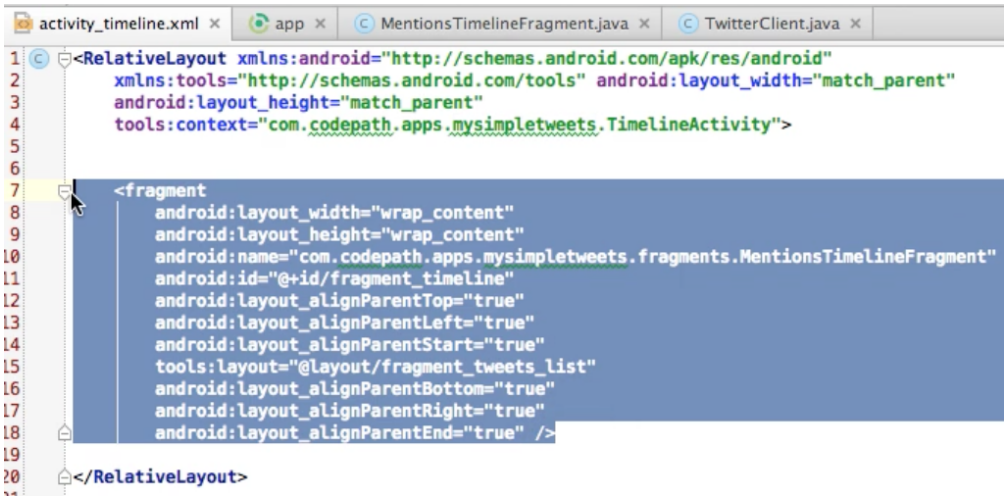
.

00:28:00 Once we've included the pagerslidingtabstrip library and sync with gradle, we have access to it in our application and can start using it.

00:28:36 Instead of embedding the fragment statically in activity_timeline.xml, we use the ViewPager to show our fragments.



00:26:38 Remove the old fragment definition in activity_timeline.xml:



00:28:54 Paste the PagerSlidingTabStrip and ViewPager definitions from the ViewPager website into activity_timeline.xml.

https://github.com/codepath/android_guides/wiki/Sliding-Tabs-with-PagerSlidingTabStrip

Once you have included the library and synced with Gradle, we can use the `SlidingTabLayout` in our layout file to display tabs. Your layout file will have tabs on the top and a `ViewPager` on the bottom as shown in the code snippet below:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <com.astuetz.PagerSlidingTabStrip
        android:id="@+id/tabs"
        app:pstsShouldExpand="true"
        app:pstsTextAllCaps="true"
        android:layout_width="match_parent"
        android:layout_height="48dp">
    </com.astuetz.PagerSlidingTabStrip>

    <android.support.v4.view.ViewPager
        android:id="@+id/viewpager"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="@android:color/white" />

</LinearLayout>
```

Make sure to **add the `xmlns:app` namespace** as shown above to your root layout.

00:29:02 Paste it into activity_timeline.xml:

activity_timeline.xml × app × MentionsTimelineFragment.java × TwitterClient.java ×

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

    <com.astuetz.PagerSlidingTabStrip
        android:id="@+id/tabs"
        app:pstsShouldExpand="true"
        app:pstsTextAllCaps="true"
        android:layout_width="match_parent"
        android:layout_height="48dp">
    </com.astuetz.PagerSlidingTabStrip>

    <android.support.v4.view.ViewPager
        android:id="@+id/viewpager"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="@android:color/white" />

</RelativeLayout>
```

Notice that we need to include the prefix definition. We get this from the ViewPager website.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
```

00:29:12 Paste it into activity_timeline.xml:

activity_timeline.xml | app | MentionsTimelineFragment.java | TwitterClient.java

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"
    android:layout_height="match_parent"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    tools:context="com.codepath.apps.mysimpletweets.TimelineActivity">

    <com.astuetz.PagerSlidingTabStrip
        android:id="@+id/tabs"
```

Because we're using relative layout, we need to add the layout_below tag to ViewPager:

```
    <android.support.v4.view.ViewPager
        android:id="@+id/viewpager"
        android:layout_below="@+id/tabs"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="@android:color/white" />

</RelativeLayout>
```

Then we get the the start of our tabbed layout:

With the tabstrip on top and the viewpager (fragment container) below it.

We need to somehow tell the ViewPager which fragment to display.
We need to create a ViewPager adapter.
00:30:31 Add the adapter code to TimelineActivity.java that will return the fragments to display and in what order.

We'll need a constructor.
Need to tell the adapter what the fragment manager is. Passed in from the activity. Need to pass it thru to the base class via super.
00:32:29 Make sure all FragmentManger modules come from android.support.v4.app:

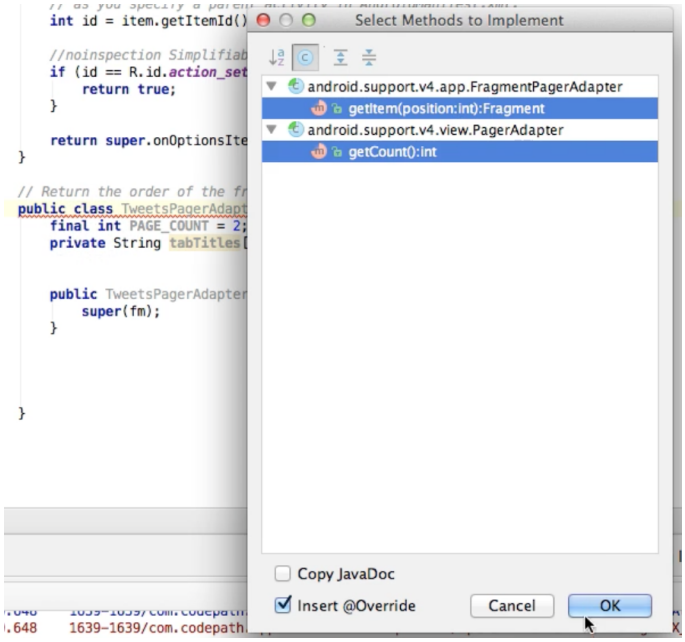
activity_timeline.xml | app | MentionsTimelineFragment.java

```
package com.codepath.apps.mysimpletweets;

import android.support.v4.app.FragmentManager;
import android.support.v4.app.FragmentPagerAdapter;
import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.util.Log;
```

00:32:33

```
    // Return the order of the fragments in the view pager
    public class TweetsPagerAdapter extends FragmentPagerAdapter {
        final int PAGE_COUNT = 2;
        private String tabTitles[] = { "Home", "Mentions" };


        public TweetsPagerAdapter(FragmentManager fm) {
            super(fm);
        }

    }
}
```

We still have an error. Select FragmentPagerAdapter and hit <alt><enter> and select implement methods. We're going to implement getItem and getCount:




```
// Return the order of the fragments in the view pager
public class TweetsPagerAdapter extends FragmentPagerAdapter {
    final int PAGE_COUNT = 2;
    private String tabTitles[] = { "Home", "Mentions" };


    public TweetsPagerAdapter(FragmentManager fm) {
        super(fm);
    }

    @Override
    public Fragment getItem(int position) {
        return null;
    }


    @Override
    public int getCount() {
        return 0;
    }
}
```

Don’t need page count, use tabTitles.length instead.
Complete getItem to specify which fragment to display.
Complete getPageTitle for tab names.
This completes our pager adapter.

```
// Return the order of the fragments in the view pager
public class TweetsPagerAdapter extends FragmentPagerAdapter {
    private String tabTitles[] = { "Home", "Mentions" };

    // Adapter gets the manager insert or remove fragment from activity
    public TweetsPagerAdapter(FragmentManager fm) {
        super(fm);
    }

    // The order and creation of fragments within the pager
    @Override
    public Fragment getItem(int position) {
        if (position == 0) {
            return new HomeTimelineFragment();
        } else if (position == 1) {
            return new MentionsTimelineFragment();
        } else {
            return null;
        }
    }

    // Return the tab title
    @Override
    public CharSequence getPageTitle(int position) {
        return tabTitles[position];
    }

    // How many fragments there are to swipe between?
    @Override
    public int getCount() {
        return tabTitles.length;
    }
}
}
```

00:35:20
Like our listview, to plug in the listview, we look up the listview, plug it into the adapter, then it binds and plugs in data.
00:36:07 So this is what we need to do:

```
// Get the viewpager
// Set the viewpager adapter for the pager
// Find the sliding tabstrip
// Attach the tabstrip to the viewpager
```

00:36:16 ViewPager vs. ViewPager Indicator (displays which page you’re on).
Add code to TimelineActivity.java to find the ViewPager and set the adapter so we can view our fragments.

```
public class TimelineActivity extends ActionBarActivity {

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

        // Get the viewpager
        ViewPager vpPager = (ViewPager) findViewById(R.id.viewpager);
        // Set the viewpager adapter for the pager
        vpPager.setAdapter(new TweetsPagerAdapter(getSupportFragmentManager()));
        // Find the sliding tabstrip
        // Attach the tabstrip to the viewpager
    }
```

00:37:34 Run to show ViewPager with the two fragments. Swipe to go between fragments.

00:40:21 Show customization from guides. i.e. using icons instead of names for tabs.
00:40:40 ViewPager addition summary.

Manifest file from MySimpleTweets-Master:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.codepath.apps.mysimpletweets"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="16"
        android:targetSdkVersion="21" />

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

    <android:uses-permission android:name="android.permission.READ_PHONE_STATE" />
    <android:uses-permission
        android:name="android.permission.READ_EXTERNAL_STORAGE"
        android:maxSdkVersion="18" />

    <application
        android:name=".TwitterApplication"
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <meta-data
            android:name="AA_DB_NAME"
            android:value="RestClient.db" />
        <meta-data
            android:name="AA_DB_VERSION"
            android:value="1" />

        <activity
            android:name=".LoginActivity"
            android:label="@string/app_name"
            android:theme="@style/AppTheme" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
            <intent-filter>
                <action android:name="android.intent.action.VIEW" />

                <category android:name="android.intent.category.DEFAULT" />
                <category android:name="android.intent.category.BROWSABLE" />
```

```
                <data
                    android:host="cpsimpletweets"
                    android:scheme="oauth" />
            </intent-filter>
        </activity>
        <activity
            android:name=".TimelineActivity"
            android:label="@string/title_activity_timeline" >
        </activity>
    </application>

</manifest>
```