

Mobile Computing
Professor Pushendra Singh
Indraprastha Institute of Information Technology Delhi
Lecture 24
Fragments

Hello, let us continue our discussion on fragments. In last lecture we discussed what fragments are, normally you should assume fragment as a sub-activity which an activity can call so that is an activity can call multiple fragments put the layout of the fragments in the activities book load. So you can see multiple fragments inside a single activity and you can create dynamic and multi-plane UI that you may need if you want to develop your applications for devices such as tablets which have a large screen size as well as performance which have smaller size. So today we will learn a little bit more about fragments.

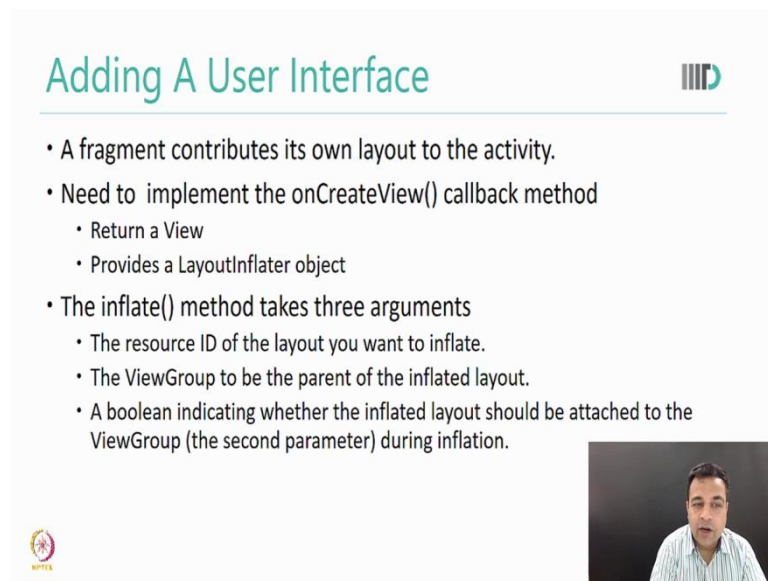
(Refer Slide Time: 1:07)



The slide is titled "Fragment Development" in a teal font. Below the title, there is a list of "Important subclasses" with three items: DialogFragment, ListFragment, and PreferenceFragment. In the bottom right corner of the slide, there is a small video inset showing a man in a light blue shirt. The slide also features a logo in the top right corner and a small circular logo in the bottom left corner.



So when you are doing fragment development you can extend fragments from 3 important subclasses. These subclasses are DialogFragment, ListFragment and PreferenceFragment. Later on we will learn a lot more about dialogue and what are the other ways to handle dialogs, but DialogueFragment gives us a fragment which can act as a dialogue we will see later in the program how to do that. Similarly ListFragment shows us a list of items and the PreferenceFragment shows us the list of items in a (()) (01:47) or the most general way is to just implement directly from the fragment class and then implement the functionality where you want to. Now how do we create a fragment? Well, for creating a fragment we extend the fragment class and we override the key lifecycle methods.

(Refer Slide Time: 2:23)



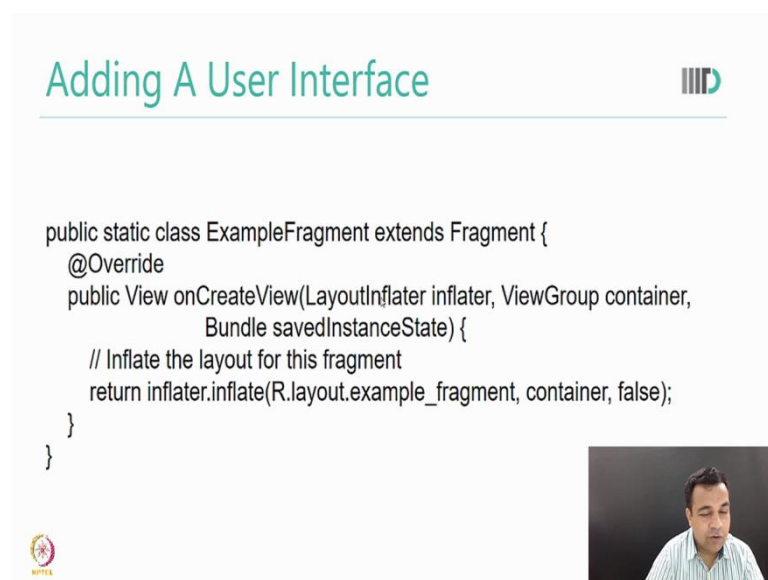
Adding A User Interface

- A fragment contributes its own layout to the activity.
- Need to implement the onCreateView() callback method
 - Return a View
 - Provides a LayoutInflater object
- The inflate() method takes three arguments
 - The resource ID of the layout you want to inflate.
 - The ViewGroup to be the parent of the inflated layout.
 - A boolean indicating whether the inflated layout should be attached to the ViewGroup (the second parameter) during inflation.



One method which we definitely use is the onCreateView() method. When you create a fragment, a fragment contributes its own layout to the activity. So you implement the onCreateView() callback method you return a view the view that contains the layout of the fragment and that layout is then inserted into the place holder inside the activity. In order to do that the fragment provides you a layout inflater object and this inflate method takes 3 arguments.

(Refer Slide Time: 3:03)



Adding A User Interface

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

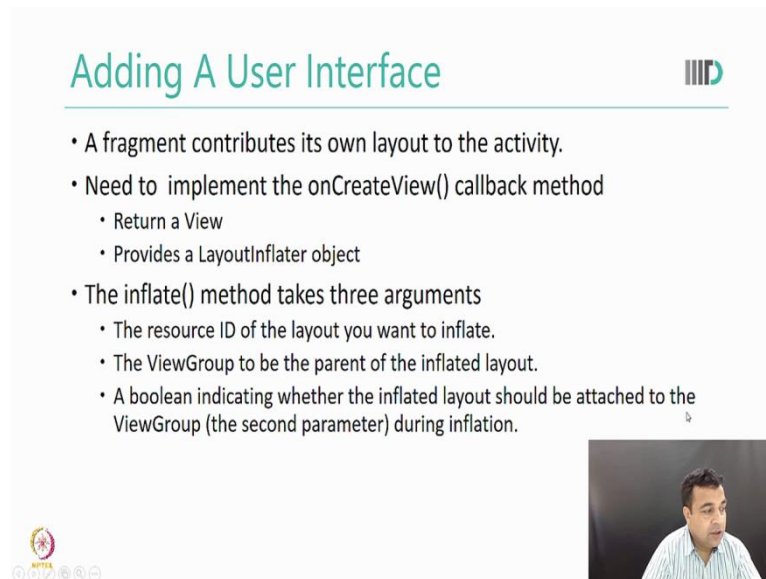
 

Lets first have a look at the method and then come back, so here is an example of the Layout Inflater so yes you can see that I am having a class which is extending a fragment inside that the only method I am overriding is the onCreateView LayoutInflater, this method is taking

three parameters one is the `LayoutInflater` another is the `ViewGroup` and third is a `Bundle`. The `Bundle` is same as the `Bundle` that we have earlier in our activity.

The `ViewGroup` is also same as we know already, the interesting new element is this layout and the `LayoutInflater` actually calls a method called `inflate` which again takes 3 parameters and these parameters are what we are investigating. So let us go back, first argument is the resource idea of the layout, so this is the resource ideas you see `R.layout.example_fragment`. This is the resource layout of the example fragment so this is the idea of this.

(Refer Slide Time: 4:41)

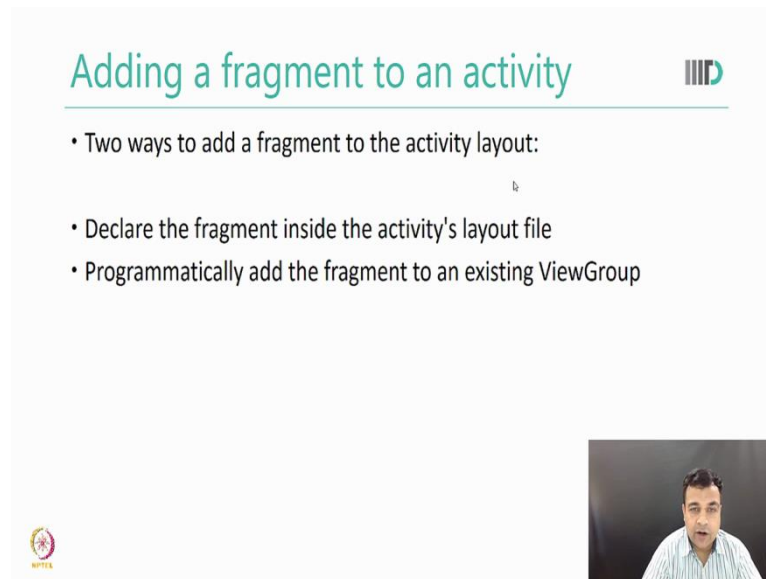


The slide is titled "Adding A User Interface" and features a list of bullet points. The first bullet point states that a fragment contributes its own layout to the activity. The second bullet point discusses the need to implement the `onCreateView()` callback method, which involves returning a `View` and providing a `LayoutInflater` object. The third bullet point details the `inflate()` method, which takes three arguments: the resource ID of the layout, the `ViewGroup` parent, and a boolean indicating whether the layout should be attached to the `ViewGroup`. A small video inset in the bottom right corner shows a man speaking.

- A fragment contributes its own layout to the activity.
- Need to implement the `onCreateView()` callback method
 - Return a `View`
 - Provides a `LayoutInflater` object
- The `inflate()` method takes three arguments
 - The resource ID of the layout you want to inflate.
 - The `ViewGroup` to be the parent of the inflated layout.
 - A boolean indicating whether the inflated layout should be attached to the `ViewGroup` (the second parameter) during inflation.

The second argument is the `ViewGroup` which will be the parent of this `inflate` layout, so this will be set fit into this container, this container will information will most likely come from the activity class that is the parent class of this fragment, by parent class I mean the activity class that is using this fragment and the third is an argument which is a `Boolean` which indicates whether the inflated layout should be attached to the `Viewgroup` or not, so we do not so we will sent this value to `false` correct. You may want to experiment by setting a `true` value and seen how does your application look.

(Refer Slide Time: 5:01)



The slide is titled "Adding a fragment to an activity" in a teal font. In the top right corner, there is a logo consisting of three vertical bars of varying heights. Below the title, there is a list of two bullet points:

- Two ways to add a fragment to the activity layout:
- Declare the fragment inside the activity's layout file
- Programmatically add the fragment to an existing ViewGroup

In the bottom right corner of the slide, there is a small video inset showing a man with dark hair and a light blue shirt speaking. In the bottom left corner, there is a small circular logo with the word "NPTX" below it.

now we know how to initialize a fragment how to create a fragment that is extending the right class and then implementing the `onCreateView` method. Now let us see that how do we add a fragment to an activity, so there are 2 ways to add a fragment to an activity and as you may also guess that the one way is to add it in the external file (`res/layout/`) (05:23) and the another way is to add it through the (`FragmentManager`) (05:26). If you remember in the very beginning they talked about that for each activity we have a layout external file but we had also said that at that time that you know we can create all this using the code with fragment it is the same thing, you can either add a fragment inside the activities layout file or we can programmatically add the fragment to an existing `ViewGroup` using the code.

What do you think is the better approach and in what condition? Think about it. So as you may imagine if your fragment is not going to be too dynamic then maybe you will consider the first approach where you directly attach it into the layout. However if your program is such that you move fragments you change fragments you change between different fragments then the second approach is the most suitable. We will learn both of the approaches in next few lectures.

(Refer Slide Time: 6:28)

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



Now let us look at the first approach for the time. How do we do it, number 1 specify layout properties for the fragment as if it was just another view. So now we already know how to add a view for example, we added buttons in our previous example which were a view. So the same thing that we have to specify layout properties for fragment as if it were a view. Now let us see an example, so this is the example that is given here let me try to increase it so that you can read more clearly. You see that we start with a linear layout this is the layout of the activity and this activity contains two fragments, which is starts with the name fragment. So the value of the fragment is only from this fragment till this angle. Now just like another view it has the elements like weight, width, height, etc but it also has something else.

We have seen id, weight width, and height earlier, but it also has something called android name and that name was actually giving what looks like a path to a class and actually this is what it is. From the name the android system will find out the fragment that you are talking about. It will inflate the layout of that fragment and with these properties it will place that layout inside this ViewGroup. So let me come back again, a fragment has android name which will refer to the class file of the fragment that you are referring to. From that end with these properties the android system will decide how to put the fragment as a view into the ViewGroup of the activity.

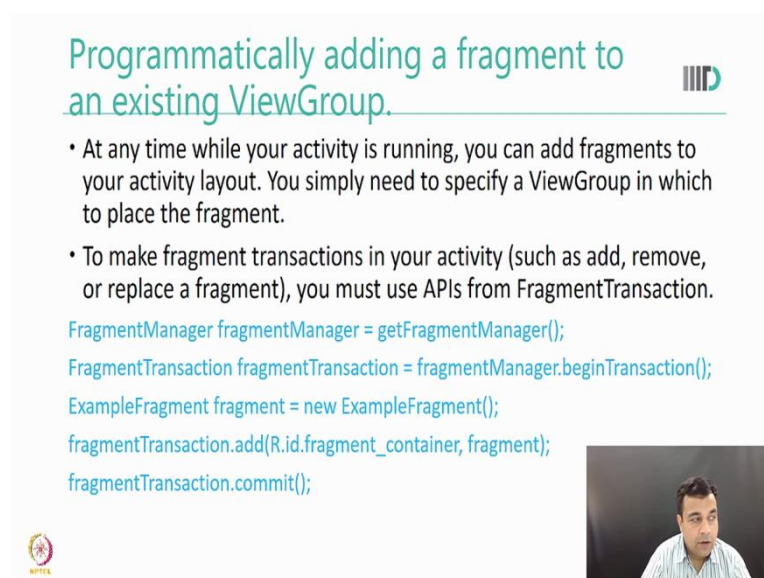
So in this class if you see this is a linear layout and in this class there are two fragments and they are these two fragments so most likely this is an interface for I would say a tablet, where the screen is divided into two parts something which we discussed earlier. Now one of the fragment may be displaying a list looks like by the name, while the another fragment

may be displaying details of a particular item list. Now let us go back so here is what we have discussed the android name attribute, the android name attributes specify the fragment class to instantiate in the layout. So in our previous example if we go few slides back we were giving the example of this ExampleFragment and then we were saying that the ExampleFragment will have inflated a layout here.

Now if I have used ExampleFragment here, then android system will seek ok it is ExampleFragment will go inflate the layout and put into the ViewGroup of the activity that is how it works. Now, each fragment that you add requires a unique identifier that a system can use. As you will see that because we are using our fragments dynamically in the program we need an identifier to refer to it and the same identifier we use to capture the fragment to perform transactions, for example “remove it”.

Now there are 3 ways to provide an identifier, the first method is to supply an attribute for android:ID as you can see in this example we are using this, this is android:ID again I will enlarge. Android:D and then the second way is supply the android tag attribute with a unique string. We are not doing that because we are using ID, you can use either of it and then the third is that if we do not do ID or tag then android system can always use the ID of the container, but I will advise you either ID or tag to identify your fragment in your program.

(Refer Slide Time: 11:49)



The slide is titled "Programmatically adding a fragment to an existing ViewGroup." and features a small "IIIID" logo in the top right corner. It contains two bullet points: "At any time while your activity is running, you can add fragments to your activity layout. You simply need to specify a ViewGroup in which to place the fragment." and "To make fragment transactions in your activity (such as add, remove, or replace a fragment), you must use APIs from FragmentTransaction." Below the text is a code block with the following Java code:

```
FragmentManager fragmentManager = getFragmentManager();
FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
ExampleFragment fragment = new ExampleFragment();
fragmentTransaction.add(R.id.fragment_container, fragment);
fragmentTransaction.commit();
```

 In the bottom right corner of the slide, there is a small video inset showing a man speaking.

Now that completes our discussion on how to add up fragment in the xml file of the activity layout file, now let us move on to this second way of adding a fragment that is programmatically adding a fragment in our activity. So if you want to add programmatically

over fragment then it anytime while our activity is running, so earlier if you remember we had just discussed that activity after the on resume method is considered in the state of running, so in that state we can add fragments to our activity. What we need to do is we will simply have to specify the ViewGroup in which to place the fragment. So the requirements of specifying the ViewGroup is same, in the xml method we were directly adding it into the ViewGroup, here we need to specify the group.

Now let us look at the code that is given, we obtain a fragment manager we obtain a fragment transaction, then this is our fragment example fragment. We create the object of it then using our fragment transaction we add our fragment to the view group were we want to add, so this is the ViewGroup R.ID.fragment_container. If we go back then we will find this container in this so this is the value that will be supplied, so here it is coming doubt here so this is the container of the activity. So this is the ViewGroup in which we are adding our own fragment and once we have done it we do a commit. So all this is a called a complete fragment transaction and for doing this transaction we use the API from the fragment transaction.

So think of the transaction of fragments similar to the transactions that you have learned in data, where you do a series of operations which you call as a transaction and after each transaction you do work up, same thing here we do a series of operation and then we commit.

(Refer Slide Time: 14:24)



The slide is titled "Managing Fragments" and features a list of methods for the FragmentManager class. The methods listed are:

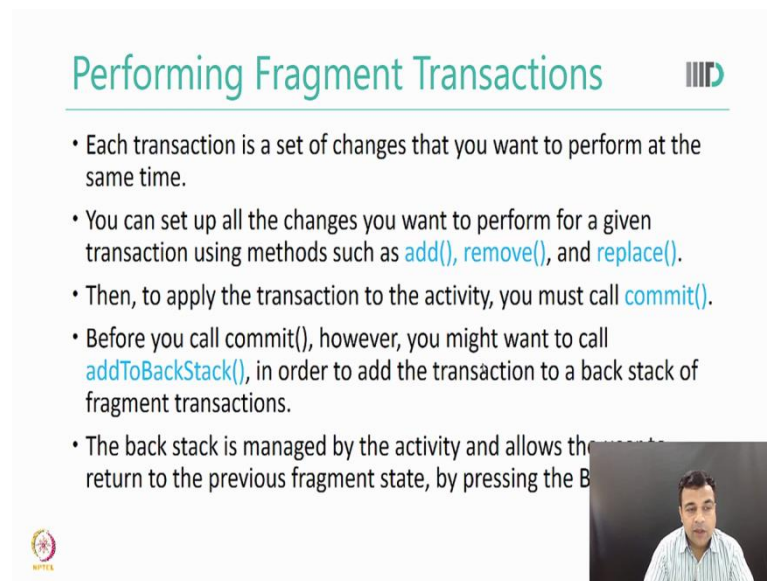
- findFragmentById() or findFragmentByTag()
 - Get fragments that exist in the activity
- popBackStack()
 - Pop fragments off the back stack
- addOnBackStackChangeListener()
 - Register a listener for changes to the back stack

In the bottom right corner of the slide, there is a small video inset showing a man speaking.

Now let us come next on discussing how to manage our fragments. So we can manage our fragments using the fragment manager this is the same fragment manager that we are talking about. In the fragment manager first you will have to obtain a reference to our fragment and

that reference we can find by either using `findFragmentById` or `findFragment by Tag`. So this will return us the fragment that exists in our activity. Now earlier we had discussed about back stack which was something maintained by the activity to manage your fragment transactions. So the `pop back stack` method will pop the latest transactions from the stack and we can also add a listener to it this is called `addOnBackStackChanged` Listener, so if there is a change in the back stack then this listener method will be called. Now let us move forward and learn about how we perform fragment transactions.

(Refer Slide Time: 15:18)



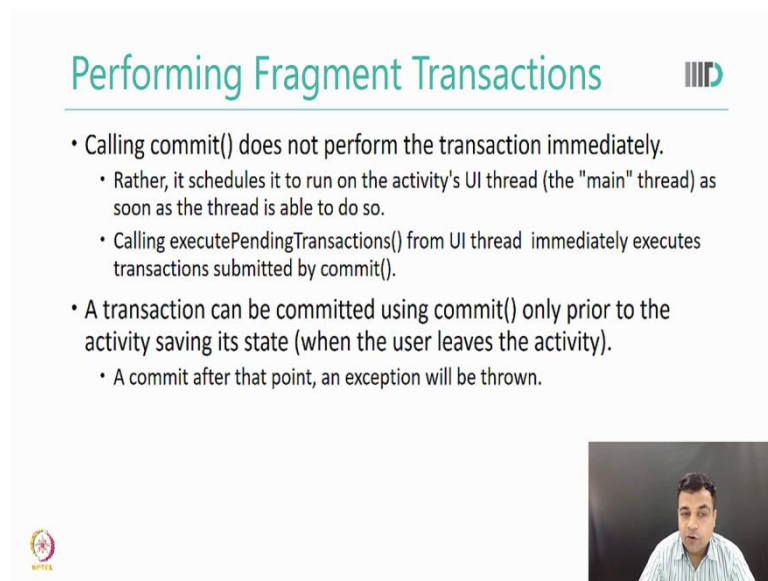
Performing Fragment Transactions

- Each transaction is a set of changes that you want to perform at the same time.
- You can set up all the changes you want to perform for a given transaction using methods such as `add()`, `remove()`, and `replace()`.
- Then, to apply the transaction to the activity, you must call `commit()`.
- Before you call `commit()`, however, you might want to call `addToBackStack()`, in order to add the transaction to a back stack of fragment transactions.
- The back stack is managed by the activity and allows the user to return to the previous fragment state, by pressing the B

NITKA

So each transactions as you know from your data base audits is a set of changes that you want to perform at the same time, you can set up all the changes you want to perform for a given transaction using methods such as `add`, `remove` and `replace` and then to apply the transactions to the activity you must call `commit`. However, if you want to call if you want to add the transaction to the back stack, then you may also want to call a method called `add to back stack` before you call. This back stack as told to you earlier is managed by the activity and allows the user to return to the previous fragment state by pressing the back.


(Refer Slide Time: 16:14)



Performing Fragment Transactions

- Calling `commit()` does not perform the transaction immediately.
 - Rather, it schedules it to run on the activity's UI thread (the "main" thread) as soon as the thread is able to do so.
 - Calling `executePendingTransactions()` from UI thread immediately executes transactions submitted by `commit()`.
- A transaction can be committed using `commit()` only prior to the activity saving its state (when the user leaves the activity).
 - A commit after that point, an exception will be thrown.

NPTEL



Few more details, when you call `commit` the action does not perform immediately, what `commit` does is that it is scheduled to run on activities UI thread which is the main thread and then it leaves it to that side and as soon as that side is able to do the `commit` it does the `commit`. That is that all the transactions which we mentioned in the `commit` will be executed, but it depends on the activities UI. However if we want to do an immediate execution then we call `execute pending transaction` from UI tag which then immediately executes transactions submitted by it. Another important detail is that a transaction can be committed using `comment` only prior to the activity saving its state that is when the user leaves the activity.

If you remember when we were discussing activity and fragment and we were discussing their methods like `onPause`, `onStop`, `onDestroy`, at times we were talking about committing the transactions committing the transactions, so this is the point so we can only `commit` only prior to activity saving its state if a `commit` after that point an exception is thrown. Now let us look at some code that performs the fragment transaction.

(Refer Slide Time: 17:49)

Performing Fragment Transactions

```
// Create new fragment and transaction

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

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

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

// Commit the transaction

transaction.commit();
```



So here is an example code, first you create your new fragment then you create a new transaction, which will get a `FragmentManager` and start the transaction. The transaction that you are doing is actually replacing the existing fragment at this fragment container with our new fragment. We are doing this replacement using the call `replace`. Earlier you had seen the call to add there is also similar call to `remove`. We want this transaction to be available on the back stack so we are doing the `transaction.addToBackStack` before we do the `commit`. So far we have talked about fragments which have a UI but a fragment could also be there without a UI.

(Refer Slide Time: 18:36)

Performing Fragment Transactions

```
// Create new fragment and transaction


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

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

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

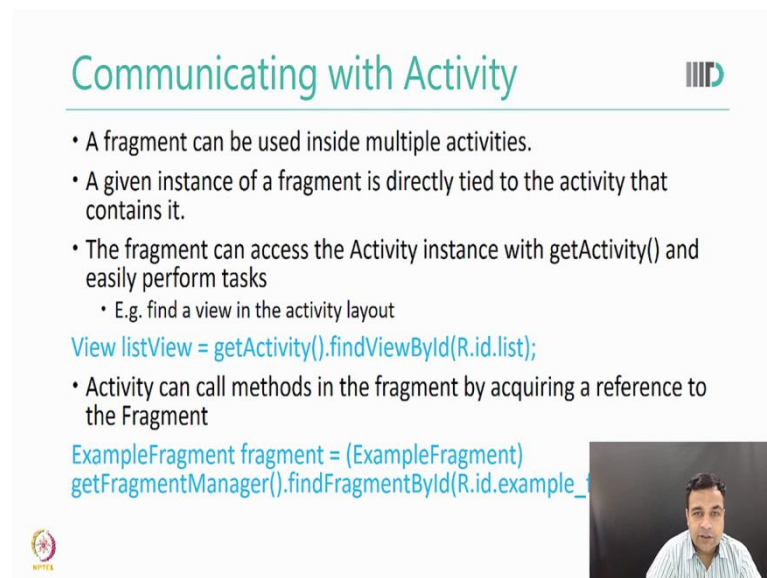
// Commit the transaction

transaction.commit();
```



Now you may want to wonder that what is the use of a fragment without a UI. Well, one simple example is you want to do something in the background, there you can have a fragment which does not have a UI. Now let us see that how do we add a fragment without a UI. It is again very easy we can use the same add, so here we are giving a string a not a view ID. It will add the fragment but will not call onCreateView because it need not create a UI and then findFragmentByTag() return. So here we have learned that how to do basic functionality within the fragments, how to perform transactions, how to add fragments etc. Now let us come to another very important topic that how do the fragment communicates with a activity.

(Refer Slide Time: 19:36)



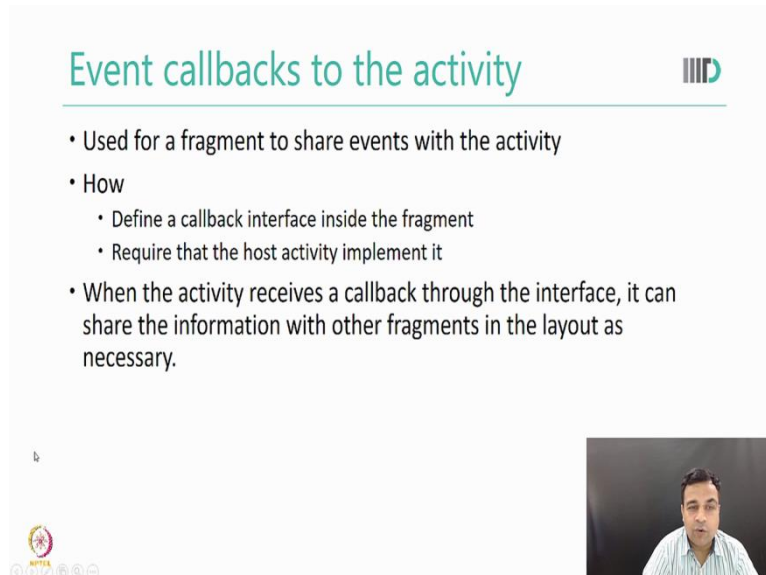
The slide is titled "Communicating with Activity" and features a list of bullet points and two code snippets. The first bullet point states that a fragment can be used inside multiple activities. The second bullet point explains that a given instance of a fragment is directly tied to the activity that contains it. The third bullet point notes that the fragment can access the Activity instance with getActivity() and easily perform tasks, with an example of finding a view in the activity layout. The first code snippet shows how to find a view: `View listView = getActivity().findViewById(R.id.list);`. The second bullet point states that an activity can call methods in the fragment by acquiring a reference to the fragment, with a corresponding code snippet: `ExampleFragment fragment = (ExampleFragment) fragmentManager().findFragmentById(R.id.example_f`. The slide also includes a small video inset of a man speaking in the bottom right corner and a logo in the bottom left corner.

So let us see, first a small recall as you as I told earlier that fragment can be used inside multiple activities and in fact your advice to develop a fragment so that it is an independent module which can be used inside multiple activities, but a given instance of a fragment is directly tied to the activity that contains. So while there could be multiple activities in your application, the activities that contains the fragment at a given point of time your fragment is directly tied to the activity. So how does the fragment gets the reference of the activity, well there is a call called getActivity. When a fragment makes a call to getActivity it gets a reference of the activities in which it is currently run. So by during this then it can do other performance for example, a fragment in this example in this example your fragment wants to get a view in the activity layout.

So it makes a call to getActivity using that reference which was findViewById. Just like fragment can access activity, activity can also access fragments, so the activity can call



methods in the fragment by acquiring a reference to the frag, how does activity gets a reference, by calling this method called `getFragmentManager` and `findFragmentById`. So now my activity has got a reference `()` (21:18) So you see that both activity can get a reference to the fragment and fragment can get a reference to the activity.

(Refer Slide Time: 21:32)



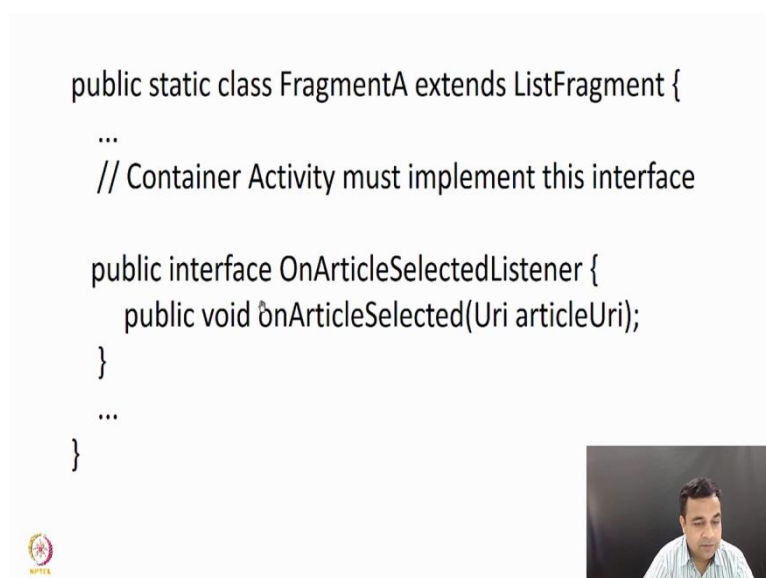
Event callbacks to the activity

- Used for a fragment to share events with the activity
- How
 - Define a callback interface inside the fragment
 - Require that the host activity implement it
- When the activity receives a callback through the interface, it can share the information with other fragments in the layout as necessary.



 

Now let us discuss about how to have callbacks which also allow the interaction between the fragment and `()` (21:37) so we use call back to share some events with activity. How do we do it, define a callback interface inside the fragment and ask the activity to implement that. Now when the activity receives the callback to the interface it can share the information with other fragments in the layout as necessary.

(Refer Slide Time: 22:19)

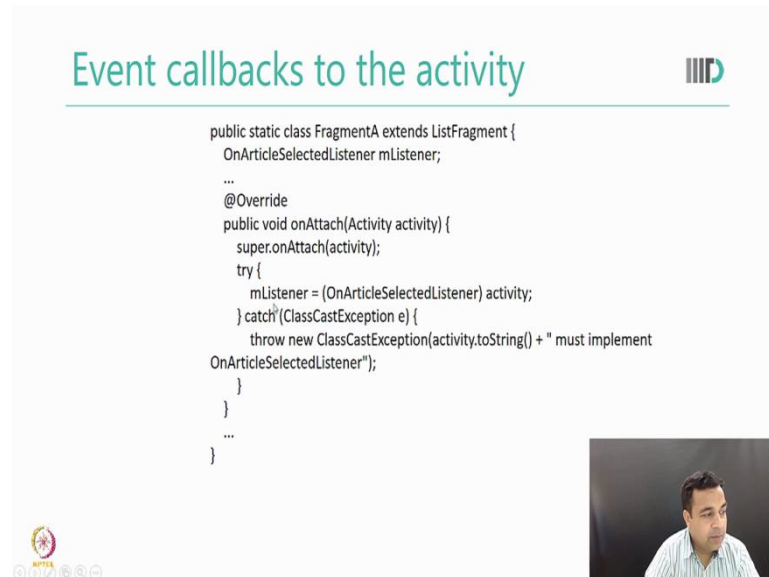


```
public static class FragmentA extends ListFragment {  
    ...  
    // Container Activity must implement this interface  
  
    public interface OnArticleSelectedListener {  
        public void onArticleSelected(Uri articleUri);  
    }  
    ...  
}
```

So let us quickly look into a very brief example again let me increase the font here you see that there is a fragment and this is the interface. Now let us go to second screen here we are doing little bit more than that.

(Refer Slide Time: 22:41)



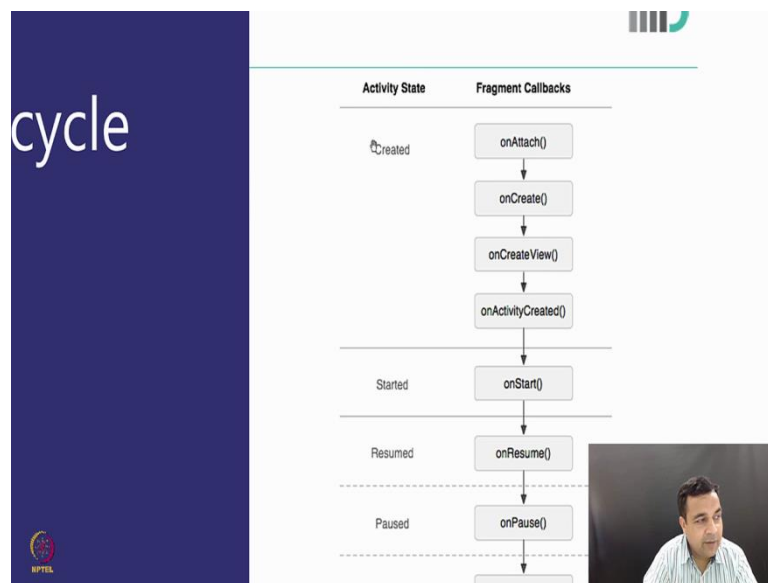
The slide is titled "Event callbacks to the activity" and features the Android logo in the top right corner. It displays the following Java code snippet:

```
public static class FragmentA extends ListFragment {
    OnItemSelectedListener mListener;
    ...
    @Override
    public void onAttach(Activity activity) {
        super.onAttach(activity);
        try {
            mListener = (OnItemSelectedListener) activity;
        } catch (ClassCastException e) {
            throw new ClassCastException(activity.toString() + " must implement
OnItemSelectedListener");
        }
    }
    ...
}
```

In the bottom right corner of the slide, there is a small video inset showing a man in a light blue shirt speaking.

So earlier we had this, onArticleSelectedListner now you can see that it is coming here. Going to mListner and then in the 3rd screen you can see that from the mListner we are (()) (22:46). So later on we will learn a program and there we will see that how to implement a callback but this gives me a basic idea which was to define a callback interface and require the host activity to implement it. In later videos we will write a program which is equally (()) (23:07). Now let us discuss a little bit more about the fragment lifecycle, we discussed it briefly in the previous lecture and this lecture we are going to discuss it in more detail. Number one, let us see how does the fragment lifecycle and activity lifecycle work together.

(Refer Slide Time: 23:40)

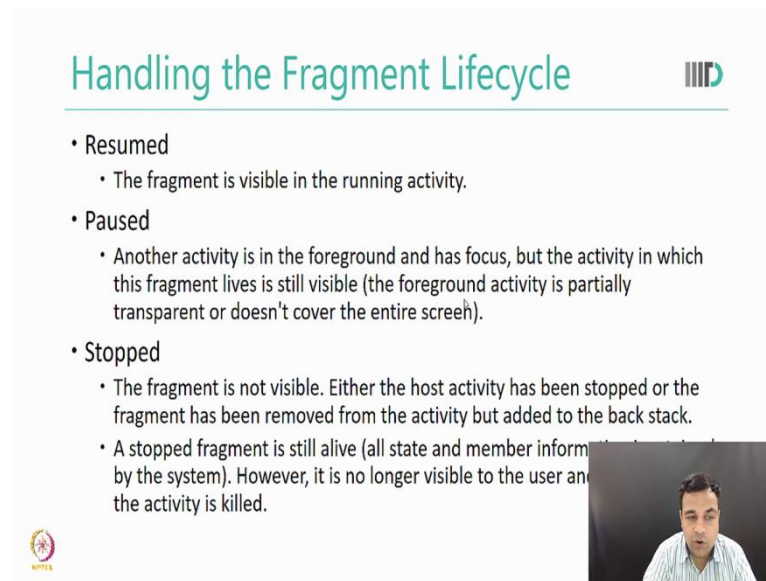


So let me also make a bigger image so that you can see clearly on the left side I am mentioning activity states and on the right side I am mentioning fragment callbacks. When the activity state is created that is the very first activities created. The fragment callback four fragment callback happen at the same time, onAttach, onCreate, onCreateView and onActivityCreated.

When the activity state moves to started onStart callback, when the activity state moves to resumed onResume callback and as you know from your knowledge of activity that the activity stays in a longer duration only in the resume state and not in the created and start state. So this is the main or the major state in which an activity is and fragment callback is the onResume, then in paused and stopped also cycle an activity stays for a longer time even that also results into the corresponding pause sometime, but when an activity is destroyed this again requires a multiple number of callbacks to be called in the fragments onDestroyView, onDestroy and onDetach.

What my advice to you is to override all these methods in your fragment example and put appropriate log nest is there similar to what we did when we wanted to understand the activity example. Now once you have placed the log messages then you run your program and you do various things. You go you press the back button you press the home button you start another application and try to see that what is happening to your fragments when your activities is changing its lifecycle. Let us come back to the normal point move to the next slide.

(Refer Slide Time: 25:52)

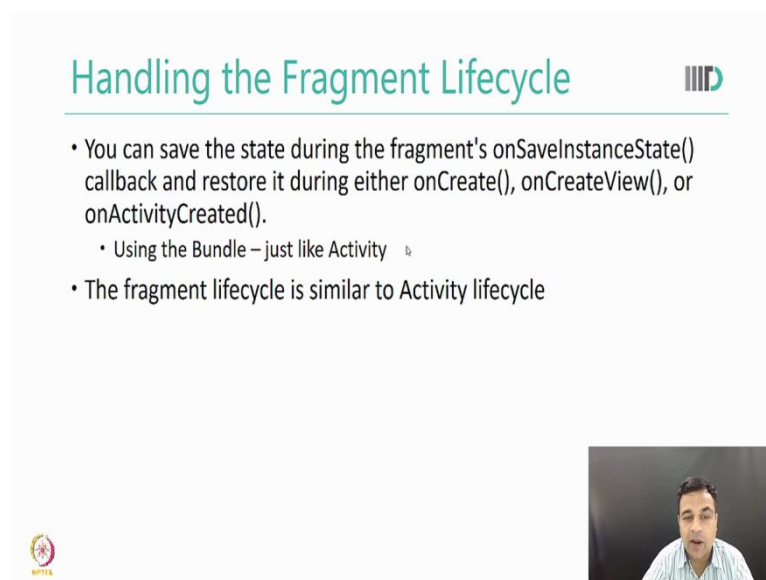


Handling the Fragment Lifecycle

- Resumed
 - The fragment is visible in the running activity.
- Paused
 - Another activity is in the foreground and has focus, but the activity in which this fragment lives is still visible (the foreground activity is partially transparent or doesn't cover the entire screen).
- Stopped
 - The fragment is not visible. Either the host activity has been stopped or the fragment has been removed from the activity but added to the back stack.
 - A stopped fragment is still alive (all state and member information is maintained by the system). However, it is no longer visible to the user and the activity is killed.

So let us go again into the resume pause and stop because these are the three major states in which activity also lasts for longer time and fragment. So in the resume state the fragment is resumed, in the paused state there is another activity in the foreground and has focus, but the activity in which this fragment lives is still visible something like a dialogue box right and then in the stopped state the fragment is not visible either the host activity has been stopped or the fragment has been removed from the activity.

(Refer Slide Time: 27:06)



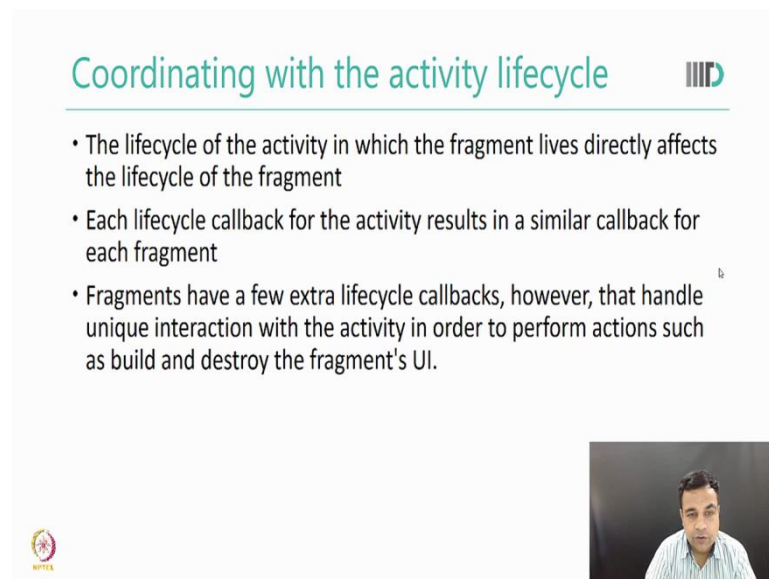
Handling the Fragment Lifecycle

- You can save the state during the fragment's `onSaveInstanceState()` callback and restore it during either `onCreate()`, `onCreateView()`, or `onActivityCreated()`.
 - Using the Bundle – just like Activity
- The fragment lifecycle is similar to Activity lifecycle

Just like the activity has stopped fragment is still alive and it is just no longer visible, So as you see that there are lots of similarities between fragments and activities and if you understand the activity you should have little problem in understanding the fragment. Now


let us discuss the lifecycle a bit more. Just like what we did in our activity we can also do in fragment that is you can save the state during the fragments onSaveInstanceState callback and restore it during the onCreate or onCreateView or onActivityCreated. This functionality is also very simple as we did in the activity using the using the Bundle. So effectively a fragment lifecycle is very very similar to the activity lifecycle.

(Refer Slide Time: 27:34)



Coordinating with the activity lifecycle

- The lifecycle of the activity in which the fragment lives directly affects the lifecycle of the fragment
- Each lifecycle callback for the activity results in a similar callback for each fragment
- Fragments have a few extra lifecycle callbacks, however, that handle unique interaction with the activity in order to perform actions such as build and destroy the fragment's UI.





Now let us see that how these two lifecycle co-ordinate. So the lifecycle of activity in which the fragment lives directly affects the life cycle of the fragment, so activity lifecycle is controlled by let us say android system and fragment lifecycle is pretty much controlled by the activity. Each lifecycle callback for the activity results in a similar callback for each activity. So if in a lifecycle onResume is called then in the (()) (20:00). The fragment has some few extra life cycle callbacks that will handle the unique interactions with the activity in order to perform some actions for example, building and destroying fragments UI.

(Refer Slide Time: 28:25)

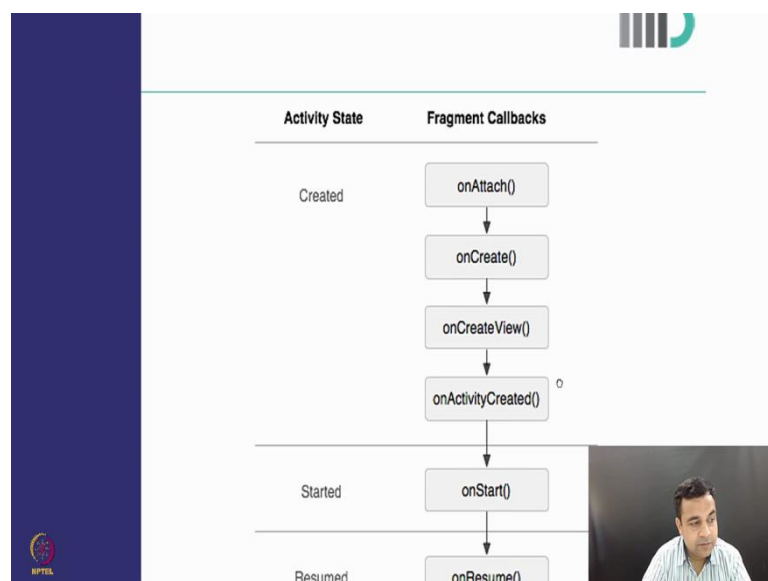
Coordinating with the activity lifecycle

- `onAttach()`
 - Called when the fragment has been associated with the activity (the Activity is passed in here).
- `onCreateView()`
 - Called to create the view hierarchy associated with the fragment.
- `onActivityCreated()`
 - Called when the activity's `onCreate()` method has returned.
- `onDestroyView()`
 - Called when the view hierarchy associated with the fragment is being removed.
- `onDetach()`
 - Called when the fragment is being disassociated from the activity.



So again some details what happens when the methods other than what is exactly common with the activity happen. When `onAttach` is called, when the fragment has been associated with the activity `onCreateView` is called to create the view hierarchy associated with the fragment that we have already seen and `onActivityCreated` is called then the activities `onCreate` method has returned. Similarly, `onDestroyView` is called when the view hierarchy associated with the fragments is been removed and `onDetach` is called when the fragment is being disassociated from the activity. The best way to learn about all these skills to write a program override the methods add the log messages and then play with the program to see what is happening, so one more time I will display the same image that I displayed earlier.

(Refer Slide Time: 29:34)



You can see, `onAttached`, `onCreateView`, `OnactivityCreated` just we discussed rest of the matters are same and `onDestroyView` and `onDetach` that we just discussed. So with this we complete our discussion on fragments and in next lecture we will work through a programming example to see some of these concepts in it.

Thank you