Parul® University
Vadodara, Gujarat
NAAC GRADE A++

Mobile App Development

WORKING WITH VIEWS & FRAGMENT

Prof. Nilesh B. Ghavate
Assistant Professor
CSE IEP Department

Outline

- GridView
- WebView
- ScrollView
- ListView
- RecyclerView
- CardView
- Fragment : Introduction
- life Cycle
- Implementation

Gridview

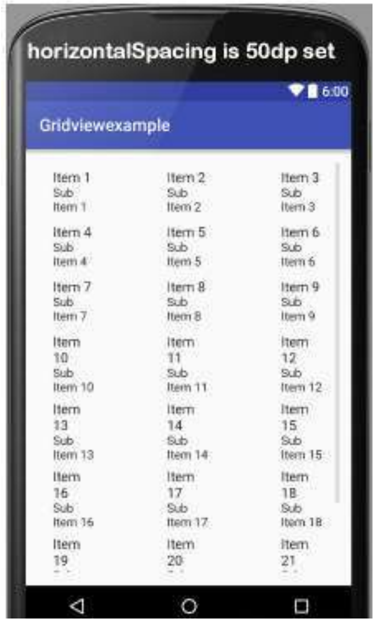
- In android GridView is a view group that display items in two dimensional scrolling grid (rows and columns), the grid items are not necessarily predetermined but they are automatically inserted to the layout using a ListAdapter.
- Users can then select any grid item by clicking on it. GridView is default scrollable so we don't need to use ScrollView or anything else with GridView.
- GridView is widely used in android applications.
- An example of GridView is your default Gallery, where you have number of images displayed using grid.

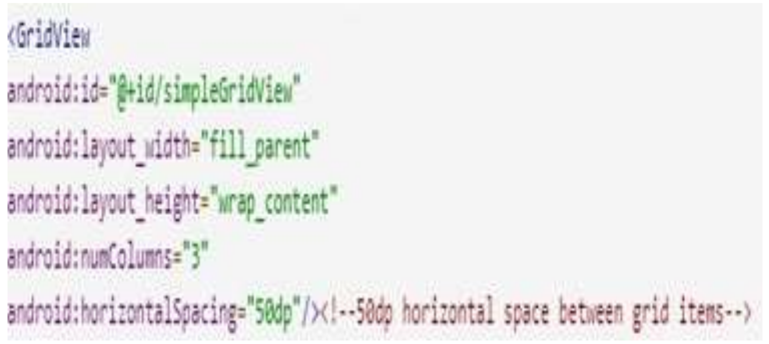
Gridview - attributes

- **1.id:** id is used to uniquely identify a GridView.

- **2.numColumns:** num Column define how many columns to show. It may be a integer value, such as “5” or auto_fit.

- **3. horizontal Spacing:** horizontal Spacing property is used to define the default horizontal spacing between columns. This could be in pixel(px),density pixel(dp) or scale independent pixel(sp).



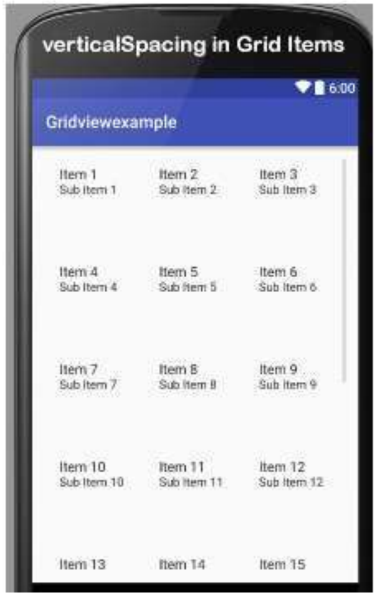


```
<GridView
android:id="@+id/simpleGridView"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:numColumns="3"
android:horizontalSpacing="50dp"/><!--50dp horizontal space between grid items-->
```

Gridview - attributes

•**4.verticalSpacing:** verticalSpacing property used to define the default vertical spacing between rows. This should be in px, dp or sp**.**

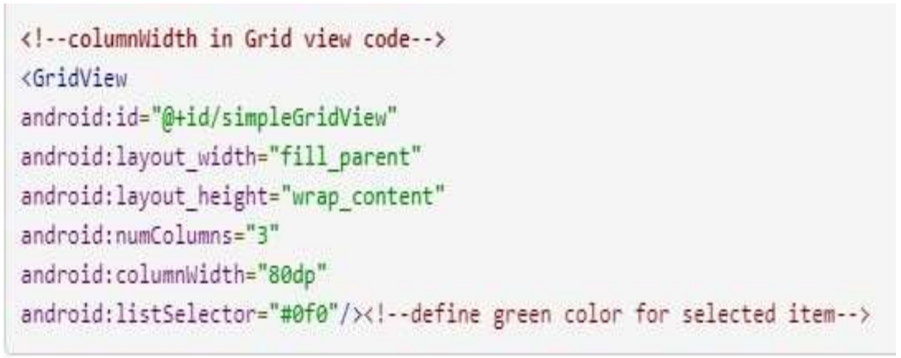
```
<!-- Vertical space between grid items code -->
<GridView
android:id="@+id/simpleGridView"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:numColumns="3"
android:verticalSpacing="50dp"/><!--50dp vertical space set between grid items-->
```



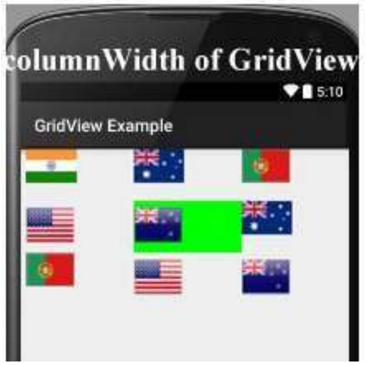
Gridview - attributes

5. columnWidth: columnWidth property specifies the fixed width of each column. This could be in px, dp or sp.

- Below is the columnWidth example code. Here column width is 80dp and selected item's background color is green which shows the actual width of a grid item.



```
<!--columnWidth in Grid view code-->
<GridView
android:id="@+id/simpleGridView"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:numColumns="3"
android:columnWidth="80dp"
android:listSelector="#0f0"/><!--define green color for selected item-->
```



Gridview – adapter class

❑An adapter is a bridge between UI component and data source that helps us to fill data in UI component. It holds the data and sends the data to adapter view, then view can takes the data from the adapter view and shows the data on different views like as list view, grid view, spinner etc.

❑GridView and ListView both are subclasses of AdapterView and it can be populated by binding to an Adapter, which retrieves the data from an external source and creates a View that represents each data entry. In android commonly used adapters which fill data in GridVieware:

- 1. Array Adapter
- 2. Base Adapter
- 3. Custom Array Adapter

Gridview – adapter class

- **1. Avoid Array Adapter To Fill Data In GridView**:
- Whenever you have a list of single items which is backed by an array, you can use Array Adapter. For instance, list of phone contacts, countries or names.
- By default, Array Adapter expects a Layout with a single TextView, If you want to use more complex views means more customization in grid items, please avoid Array Adapter and use custom adapters.
- Array Adapter adapter = new Array Adapter<String>(this,R.layout.ListView,R.id.textView,StringArray);

Gridview – adapter class

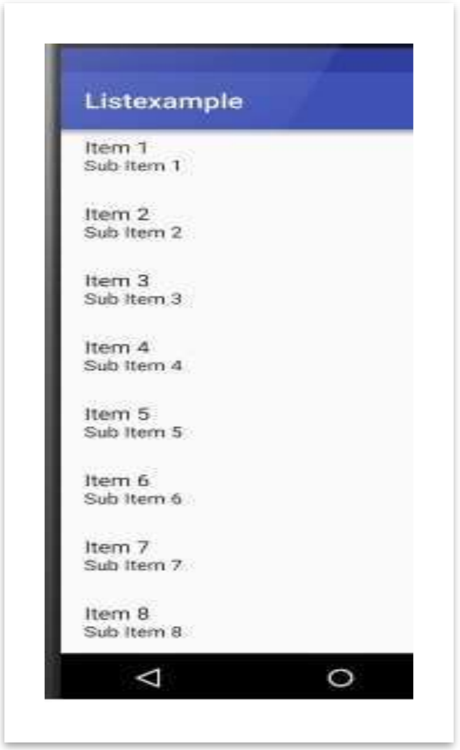
- **2. GridView Using Base Adapter In Android:**
- Base Adapter is a common base class of a general implementation of an Adapter that can be used in GridView. Whenever you need a customized grid view you create your own adapter and extend base adapter in that. Base Adapter can be extended to create a custom Adapter for displaying custom grid items. ArrayAdapter is also an implementation of BaseAdapter.
- **Example of GridView using Base Adapter in Android Studio:** Below is the example of GridView in Android, in which we show the Android logo's in the form of Grids. In this example firstly we create an int type array for logo images and then call the Adapter to set the data in the GridView. In this we create a CustomAdapter by extending BaseAdapter in it. At Last we implement setOnItemClickListener event on GridView and on click of any item we send that item to another Activity and show the logo image in full size.
- ..\Gridview_BaseAdapter.docx

Gridview – adapter class

- **3. GridView Example Using Custom ArrayAdapter In Android Studio:**
- ArrayAdapter is also an implementation of BaseAdapter so if we want more customization then we create a custom adapter and extend ArrayAdapter in that. Here we are creating GridView using custom array adapter.
- **Example of GridView using Custom Adapter :** Example of Grid View using custom arrayadapter to show birds in the form of grids. Below is the code and final output:
- Below you can download code, see final output and step by step explanation of the topic.
- ..\Gridview_ArrayAdapter.docx

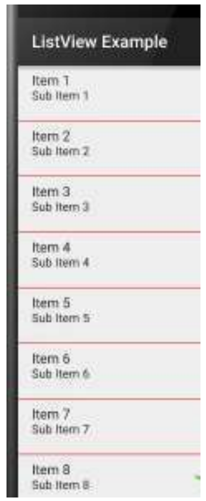
Listview

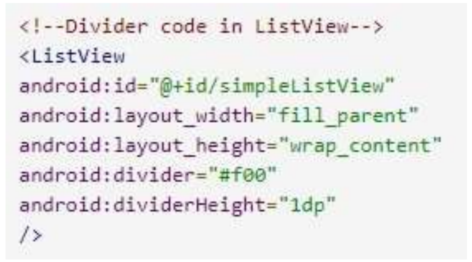
- List of scrollable items can be displayed in Android using ListView. It helps you to displaying the data in the form of a scrollable list.
- Users can then select any list item by clicking on it. ListView is default scrollable so we do not need to use scroll View or anything else with ListView.
- ListView is widely used in android applications. A very common example of ListView is your phone contact book, where you have a list of your contacts displayed in a ListView and if you click on it then user information is displayed.



Listview

1. **id**: id is used to uniquely identify a ListView.
2. **divider:** This is a drawable or color to draw between different list items.



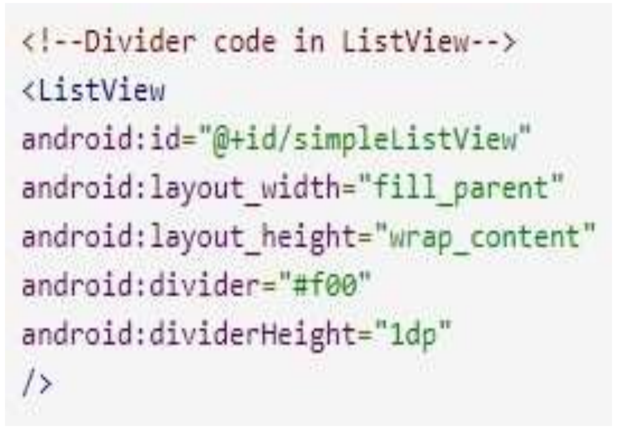


```
<!--Divider code in ListView-->
<ListView
android:id="@+id/simpleListView"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:divider="#f00"
android:dividerHeight="1dp"
/>
```

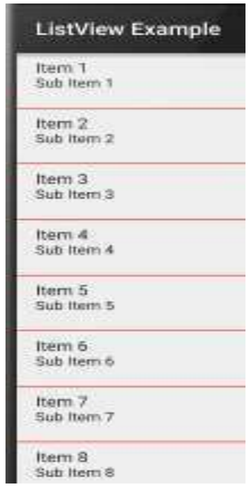
Listview

- **3. dividerHeight:**

This specify the height of the divider between list items. This could be in dp(density pixel),sp(scale independent pixel) or px(pixel).



```
<!--Divider code in ListView-->
<ListView
android:id="@+id/simpleListView"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:divider="#f00"
android:dividerHeight="1dp"
/>
```

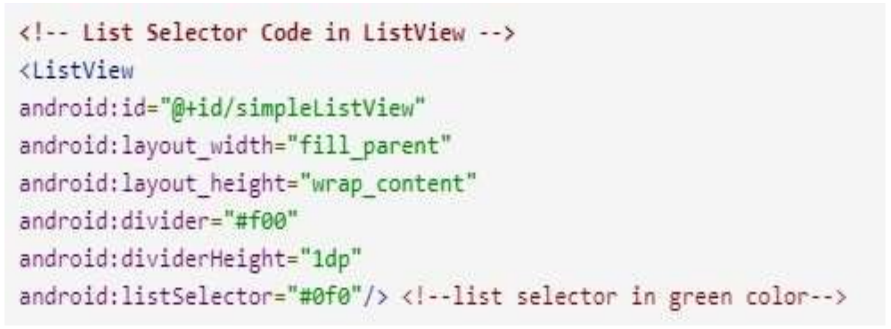


ListView Example
Item 1
Sub Item 1
Item 2
Sub Item 2
Item 3
Sub Item 3
Item 4
Sub Item 4
Item 5
Sub Item 5
Item 6
Sub Item 6
Item 7
Sub Item 7
Item 8
Sub Item 8

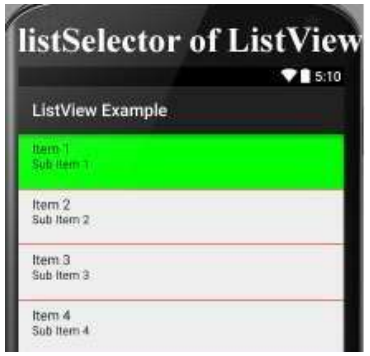
Listview

- **4. listSelector:**

listSelector property is used to set the selector of the listView. It is generally orange or Sky blue color mostly but you can also define your custom color or an image as a list selector as per your design.



```
<!-- List Selector Code in ListView -->
<ListView
android:id="@+id/simpleListView"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:divider="#f00"
android:dividerHeight="1dp"
android:listSelector="#0f0"/> <!--list selector in green color-->
```



Listview

1.Array Adapter:

- Whenever you have a list of single items which is backed by an array, you can use ArrayAdapter. For instance, list of phone contacts, countries or names.
- Important Note: By default, ArrayAdapter expects a Layout with a single TextView, If you want to use more complex views means more customization in list items, please avoid ArrayAdapter and use custom adapters.
- Below is Array Adapter code:
- ArrayAdapter adapter = new ArrayAdapter<String>(this,R.layout.ListView,R.id.textView,StringArray);
- ..\ListView_ArrayAdapter.docx

Listview

2.Base Adapter:

- BaseAdapter is a common base class of a general implementation of an Adapter that can be used in ListView. Whenever you need a customized list you create your own adapter and extend base adapter in that. Base Adapter can be extended to create a custom Adapter for displaying a custom list item. ArrayAdapter is also an implementation of BaseAdapter.

- Example of list view using Custom adapter(Base adapter):
- In this example we display a list of countries with flags. For this, we have to use custom adapter as shown in example:
- ..\ListView_BaseAdapter.docx

Scroll view

•Horizontal ScrollView:

•In android, You can scroll the elements or views in both vertical and horizontal directions. To scroll in Vertical we simply use ScrollView and to scroll in horizontal direction we need to use HorizontalScrollview.

```
•<HorizontalScrollView
 android:id="@+id/horizontalscrollView"
 android:layout_width="fill_parent"
 android:layout_height="fill_parent">

•<-- add child view's here -->

•</HorizontalScrollView >
```

Scroll view

- **Attributes Of Scroll View:**
- ScrollView and HorizontalScrollView has same attributes, the only difference is scrollView scroll the child items in vertical direction while horizontal scroll view scroll the child items in horizontal direction.
- Now let's we discuss about the attributes that helps us to configure a ScrollView and its child controls. Some of the most important attributes you will use with ScrollView include:

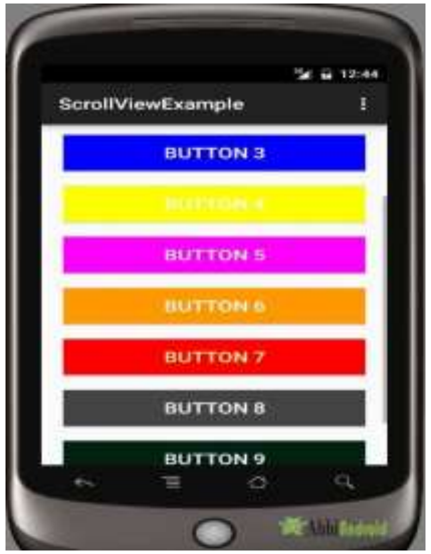
Scroll view

- **1. id:** In android, id attribute is used to uniquely identify a ScrollView.

- Below is id attribute's example code with explanation included.
- **2. scrollbars:** In android, scrollbars attribute is used to show the scrollbars in horizontal or vertical direction. The possible Value of scrollbars is vertical, horizontal or none. By default scrollbars is shown in vertical direction in scrollView and in horizontal direction in HorizontalScrollView.

- Below is scrollbars attribute's example code in which we set the scrollbars in vertical direction.

Scroll view



- < HorizontalScrollView
- android:layout_width="fill_parent"
- android:layout_height="fill_parent“
- android:scrollbars="vertical"/>
- **Example of ScrollView In Android Studio:**
- **Example 1:** In this example we will use 10 button and scroll them using ScrollView
- in vertical direction. Below is the code and final Output we will create:

Scroll view

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"      xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
"tools:context=".MainActivity"
<ScrollView
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:scrollbars="vertical">

<LinearLayout
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:layout_margin="20dp"
android:orientation="vertical">

<Button
android:layout_width="fill_parent"
android:layout_height="wrap_content
"       android:layout_gravity="center"
android:background="#f00"
android:text="Button                    1"
android:textColor="#fff"
android:textSize="20sp" />

<Button
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:layout_gravity="center"
android:layout_marginTop="20dp"
android:background="#0f0"
android:text="Button                    2"
android:textColor="#fff"
android:textSize="20sp" />

<Button
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:layout_gravity="center"
android:layout_marginTop="20dp"
android:background="#0f0"
android:text="Button                    10"
android:textColor="#fff"
android:textSize="20sp" />
</LinearLayout>
</ScrollView>
</RelativeLayout>
```

Recycler view

- RecyclerView makes it easy to efficiently display large sets of data. You supply the data and define how each item looks, and the RecyclerView library dynamically creates the elements when they're needed
- As the name implies, RecyclerView *recycles* those individual elements. When an item scrolls off the screen, RecyclerView doesn't destroy its view. Instead, RecyclerView reuses the view for new items that have scrolled onscreen. RecyclerView improves performance and your app's responsiveness, and it reduces power consumption.

Recycler view

- RecyclerView is the ViewGroup that contains the views corresponding to your data.
- It's a view itself, so you add RecyclerView to your layout the way you would add any other UI element.

Cardview

- **CardView** is a new widget in Android that can be used to display any sort of data by providing a rounded corner layout along with a specific elevation.
- CardView is the view that can display views on top of each other. The main usage of CardView is that it helps to give a rich feel and look to the UI design.
- This widget can be easily seen in many different Android Apps.
- CardView can be used for creating items in listview or inside RecyclerView.
- The best part about CardView is that it extends Framelayout and it can be displayed on all platforms of Android. Now we will see the simple example of CardView implementation.

Cardview

Attribute	Description
app:cardCornerRadius	Sets a corner radius for the card
app:cardBackgroundColor	Sets a background color for the card
app:cardElevation	Sets an elevation to the card. This is used to add a drop shadow to the card
app:cardMaxElevation	Sets a max elevation for the card
app:UseCompatPadding	Used to define a space around the view which prevents the view's elevation to get clipped by the parent view

Fragment AND ITS LIFECYCLE

- A Fragment represents a reusable portion of your app's UI.
- A fragment defines and manages its own layout, has its own lifecycle, and can handle its own input events. Fragments can't live on their own.
- They must be hosted by an activity or another fragment

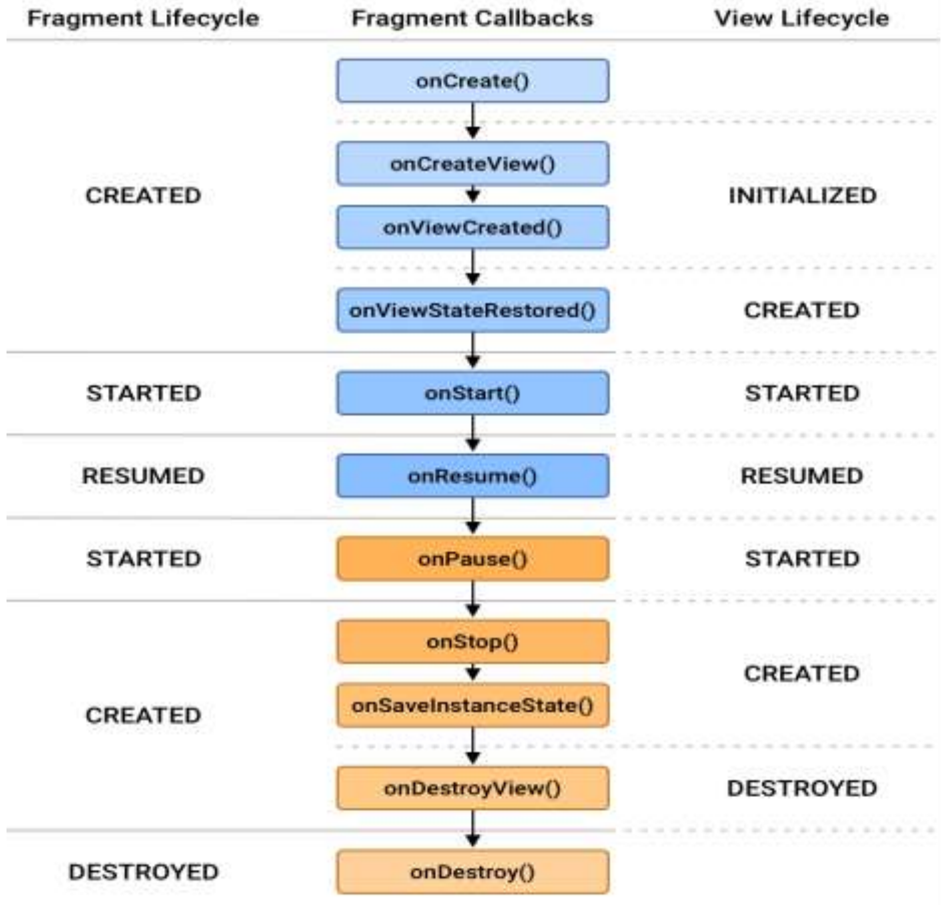
Fragment AND ITS LIFECYCLE

- Each Fragment instance has its own lifecycle. When a user navigates and interacts with your app, your fragments transition through various states in their lifecycle as they are added, removed, and enter or exit the screen.
- Each possible Lifecycle state is represented in the Lifecycle.State enum.

- INITIALIZED
- CREATED
- STARTED
- RESUMED
- DESTROYED
- These include onCreate(), onStart(), onResume(), onPause(), onStop(), and onDestroy()..

Fragment AND ITS LIFECYCLE

Figure 1 shows each of the fragment's Lifecycle states and how they relate to both the fragment's lifecycle callbacks and the fragment's view Lifecycle.



As a fragment progresses through its lifecycle, it moves upward and downward through its states. For example, a fragment that is added to the top of the back stack moves upward from CREATED to STARTED to RESUMED. Conversely, when a fragment is popped off of the back stack, it moves downward through those states, going from RESUMED to STARTED to CREATED and finally DESTROYED.