IT332: Mobile Application Development

Lecture # 06: Notifications & Fragments

Muhammad Imran

Outline

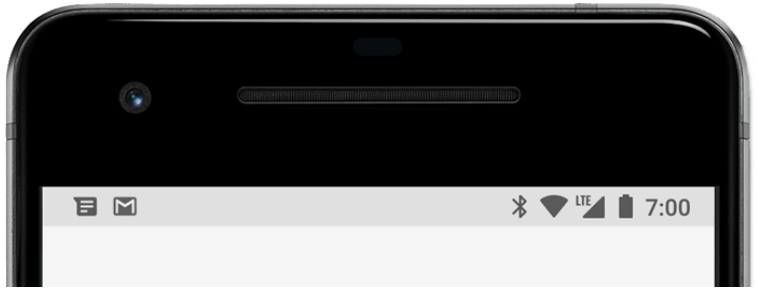
- Notifications
- Notification Anatomy
- Creating Basic Notifications
- Fragments
- Adding Fragments Dynamically
- Life Cycle of a Fragment
- Interactions between Fragments

Notifications Overview

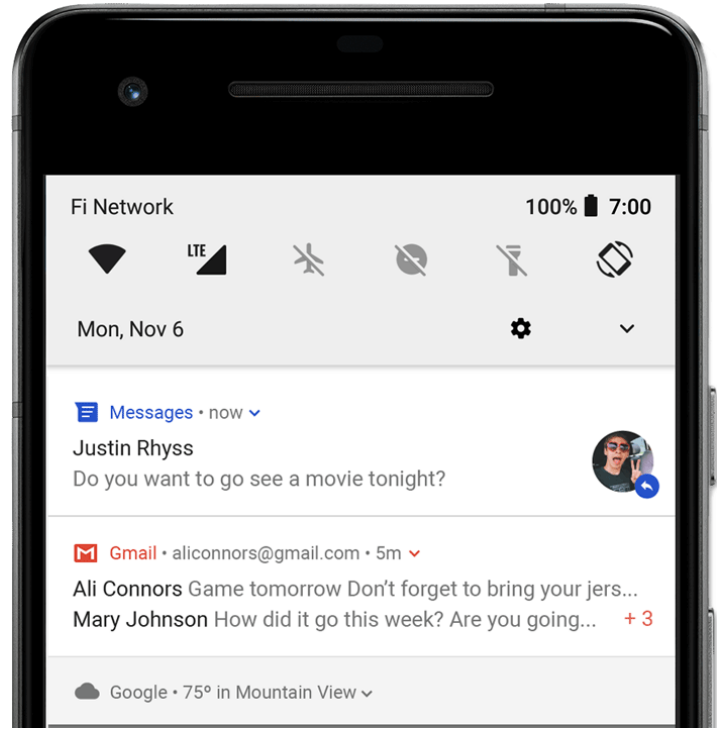
- A notification is a message that Android displays outside your app's UI to provide the user with reminders, communication from other people, or other timely information from your app.
- Users can tap the notification to open your app or take an action directly from the notification.
- Notifications appear to users in different locations and formats, such as an icon in the status bar, a more detailed entry in the notification drawer, as a badge on the app's icon, and on paired wearables automatically.

Notifications Overview

- When you issue a notification, it first appears as an icon in the status bar.



- Users can swipe down on the status bar to open the notification drawer, where they can view more details and take actions with the notification.
- A notification remains visible in the notification drawer until dismissed by the app or the user.

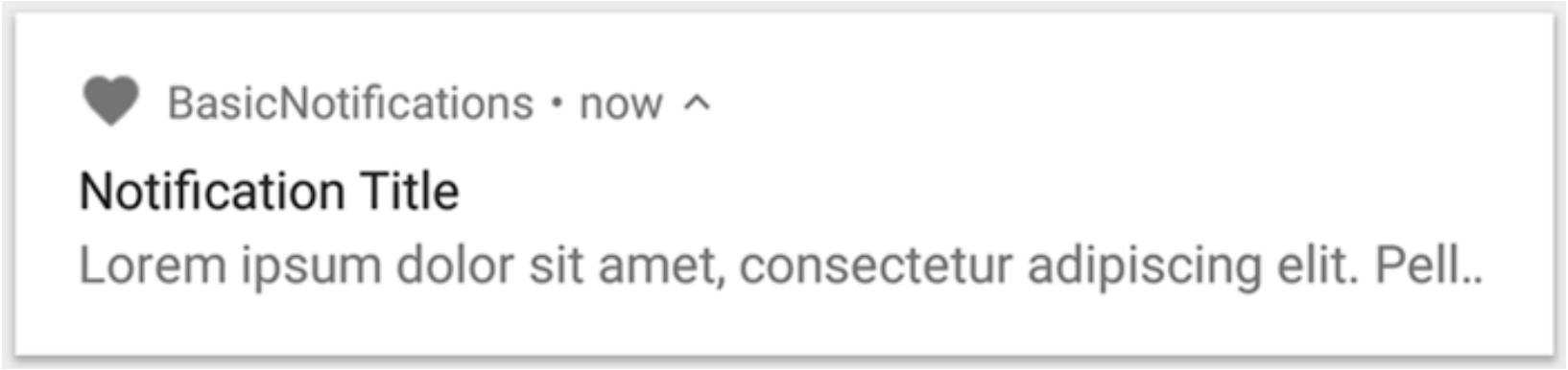


Displaying Notifications

- We have used the Toast class to display messages to the user.
- While the Toast class is a handy way to show users alerts, it is not persistent. It flashes on the screen for a few seconds and then disappears.
- If it contains important information, users may easily miss it if they are not looking at the screen.
- For messages that are important, we can use a more persistent method.
- NotificationManager is used to display a persistent message at the top of the device, commonly known as the status bar (sometimes also referred to as the notification bar).

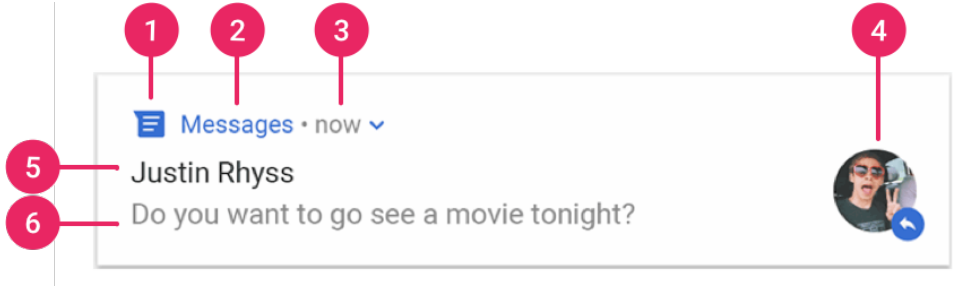
Create a Basic Notification

- A notification in its most basic and compact form (also known as collapsed form) displays an icon, a title, and a small amount of content text.



Notification Anatomy

- The design of a notification is determined by system templates—your app simply defines the contents for each portion of the template.
- Some details of the notification appear only in the expanded view.



1. Small icon: This is required and set with `setSmallIcon()`.
2. App name: This is provided by the system.
3. Time stamp: This is provided by the system but you can override with `setWhen()` or hide it with `setShowWhen(false)`.
4. Large icon: This is optional (usually used only for contact photos; do not use it for your app icon) and set with `setLargeIcon()`.
5. Title: This is optional and set with `setContentTitle()`.
6. Text: This is optional and set with `setContentText()`.

Create a Basic Notification

- To get started, you need to set the notification's content and channel using a NotificationCompat.Builder object.
  - A small icon, set by setSmallIcon(). This is the only user-visible content that's required.
  - A title, set by setContentTitle().
  - The body text, set by setContentText().
  - The notification priority, set by setPriority()

```
NotificationCompat.Builder notificationBuilder = new NotificationCompat.Builder( context: this,ANDROID_CHANNEL_ID)
        .setSmallIcon(R.mipmap.ic_launcher)
        .setContentTitle("Test Notify")
        .setContentText("This is a sample test notification")
        .setPriority(NotificationCompat.PRIORITY_DEFAULT);
```

Create a Basic Notification

- By default, the notification's text content is truncated to fit one line.
- If you want your notification to be longer, you can enable an expandable notification by adding a style template with setStyle()

```
NotificationCompat.Builder notificationBuilder = new NotificationCompat.Builder( context: this,ANDROID_CHANNEL_ID)
        .setSmallIcon(R.mipmap.ic_launcher)
        .setContentTitle("Test Notify")
        .setContentText("This is a sample test notification")
        .setPriority(NotificationCompat.PRIORITY_DEFAULT)
        .setStyle(new NotificationCompat.BigTextStyle()
                .bigText("This is a sample test notification on more than a line means on two lines"));
```

Create a Channel and Set the Importance

- Starting in Android 8.0 (API level 26), all notifications must be assigned to a channel.
- Because you must create the notification channel before posting any notifications on Android 8.0 and higher, you should create a channel as soon as your app starts.
- It's safe to call channel creation repeatedly because creating an existing notification channel performs no operation.
- This channel importance determines how to interrupt the user for any notification that belongs to this channel

Create a Channel and Set the Importance

```
private void createNotificationChannel() {
    // Create the NotificationChannel, but only on API 26+ because
    // the NotificationChannel class is new and not in the support library
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        NotificationChannel channel =
                new NotificationChannel(ANDROID_CHANNEL_ID, ANDROID_CHANNEL_NAME,
                        NotificationManager.IMPORTANCE_DEFAULT);
        // Register the channel with the system; you can't change the importance
        // or other notification behaviors after this
        NotificationManager notificationManager = getSystemService(NotificationManager.class);
        notificationManager.createNotificationChannel(channel);
    }
}
```

Set the Notification's Tap Action

- Every notification should respond to a tap, usually to open an activity in your app that corresponds to the notification.
- So, a PendingIntent object is created with an intent and passed to setContentIntent()

```
Intent intent = new Intent( packageContext: this, NotificationView.class);
intent.putExtra( name: "notificationID",notificationID);
intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TASK);
PendingIntent pi = PendingIntent.getActivity( context: this, requestCode: 0,intent, flags: 0);
```

```
NotificationCompat.Builder notificationBuilder = new NotificationCompat.Builder( context: this,ANDROID_CHANNEL_ID)
        .setSmallIcon(R.mipmap.ic_launcher)
        .setContentTitle("Test Notify")
        .setContentText("This is a sample test notification")
        .setPriority(NotificationCompat.PRIORITY_DEFAULT)
        .setContentIntent(pi)
        .setAutoCancel(true);
```

- Notice this code calls setAutoCancel(), which automatically removes the notification when the user taps it.

Set the Notification's Tap Action

- The setFlags() method helps preserve the user's expected navigation experience after they open the app via notification.
- The use of setFlags() depends on what type of activity you're starting, which may be one of the following:
  - An activity that exists exclusively for responses to the notification. There's no reason the user would navigate to this activity during normal app use, so the activity starts a new task instead of being added to your app's existing task and back stack.

```
intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TASK);
```

  - An activity that exists in your app's regular app flow. In this case, starting the activity should create a back stack so that the user's expectations for the Back and Up buttons is preserved.

Show the Notification

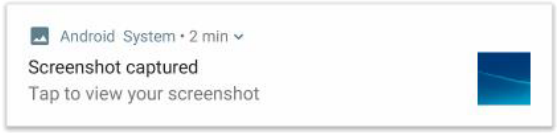
- To make the notification appear, call NotificationManagerCompat.notify(), passing it a unique ID for the notification and the notification object.

- The notification ID can be used to update or remove the notification later.

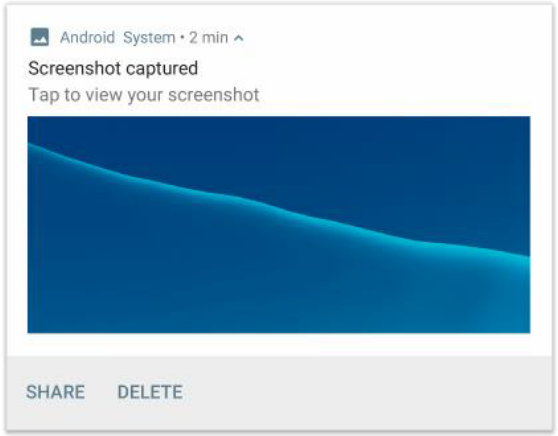
```
NotificationManager nm = (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
nm.notify(notificationID,notificationBuilder.build());
```

Create an Expandable Notification

Collapsed

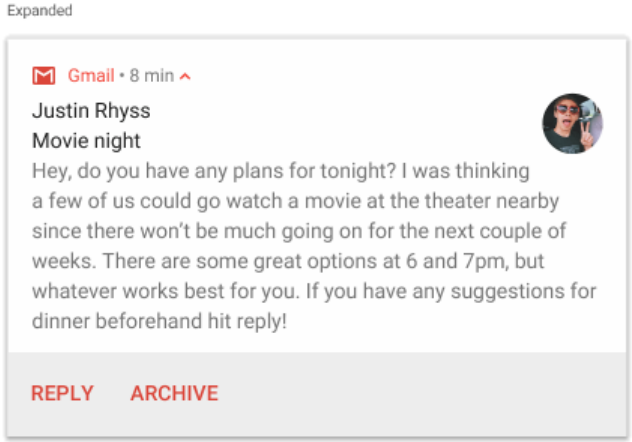


Expanded

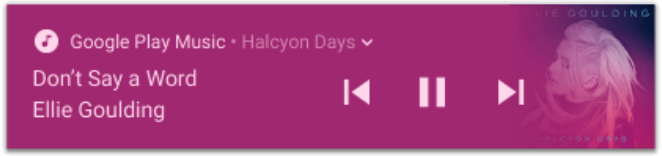


Collapsed

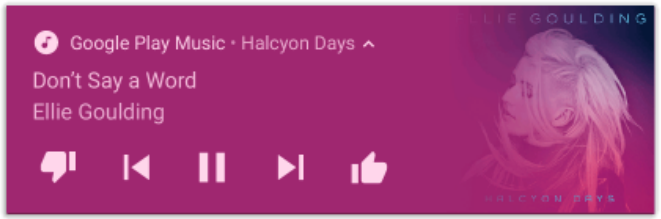
Expanded

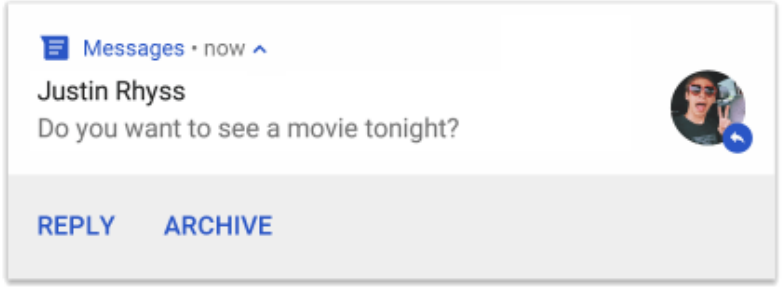


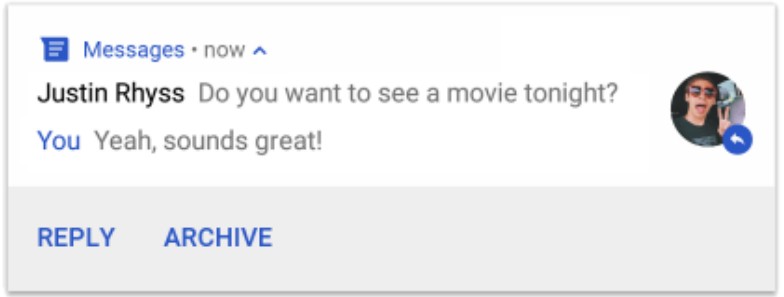
Collapsed



Expanded







Create an Expandable Notification

- A basic notification usually includes a title, a line of text, and one or more actions the user can perform in response.

- To provide even more information, you can also create large, expandable notifications by applying one of several notification templates like:
  - Add a large image
  - Add a large block of text
  - Create an inbox styled notification
  - Show a conversation in notification
  - Create a notification with media controls

Add a Large Image

- To add an image in your notification, pass an instance of NotificationCompat.BigPictureStyle to setStyle().

```
Notification notification = new NotificationCompat.Builder(context, CHANNEL_ID)
        .setSmallIcon(R.drawable.new_post)
        .setContentTitle(imageTitle)
        .setContentText(imageDescription)
        .setStyle(new NotificationCompat.BigPictureStyle()
              .bigPicture(myBitmap))
        .build();
```

Add a Large Image

- To make the image appear as a thumbnail only while the notification is collapsed, call setLargeIcon() and pass it the image, but also call BigPictureStyle.bigLargeIcon() and pass it null so the large icon goes away when the notification is expanded:

```
Notification notification = new NotificationCompat.Builder(context, CHANNEL_ID)
        .setSmallIcon(R.drawable.new_post)
        .setContentTitle(imageTitle)
        .setContentText(imageDescription)
        .setLargeIcon(myBitmap)
        .setStyle(new NotificationCompat.BigPictureStyle()
                .bigPicture(myBitmap)
                .bigLargeIcon(null))
        .build();
```

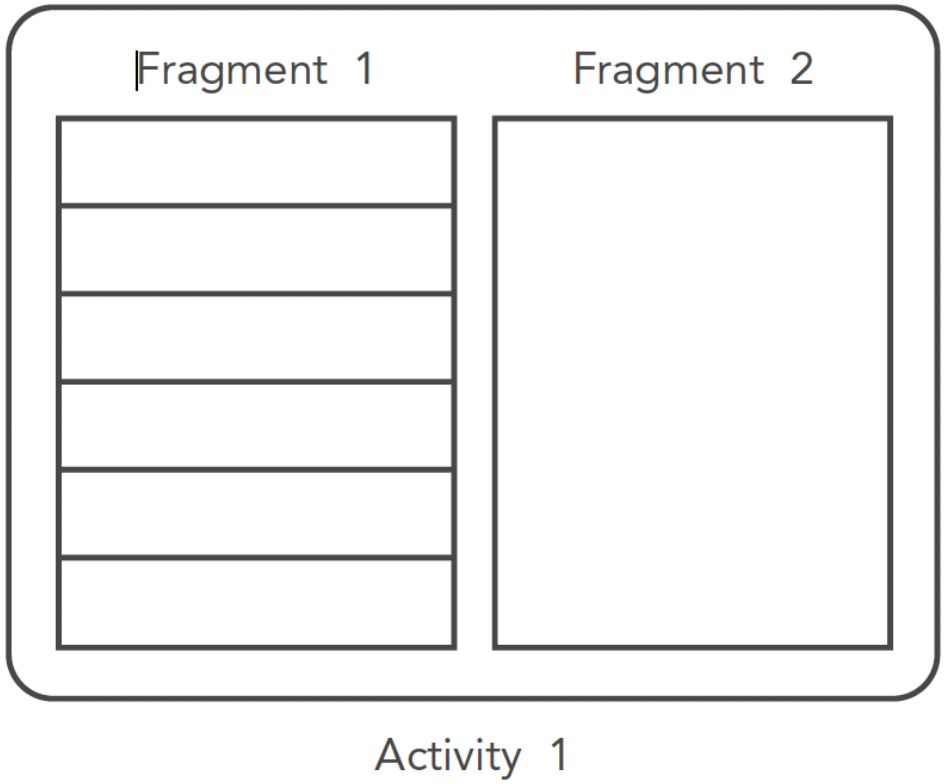
Add a Large Block of Text

- Apply NotificationCompat.BigTextStyle to display text in the expanded content area of the notification:

```
Notification notification = new NotificationCompat.Builder(context, CHANNEL_ID)
        .setSmallIcon(R.drawable.new_mail)
        .setContentTitle(emailObject.getSenderName())
        .setContentText(emailObject.getSubject())
        .setLargeIcon(emailObject.getSenderAvatar())
        .setStyle(new NotificationCompat.BigTextStyle()
                .bigText(emailObject.getSubjectAndSnippet()))
        .build();
```

Fragments

- Activity is a container for views => typically fills the entire screen
- Fragments are introduced for large screen devices
  - One activity contains several mini-activities (fragments)



Fragments

- You can think of a fragment as a modular section of an activity, which has its own lifecycle, receives its own input events, and which you can add or remove while the activity is running (sort of like a "sub activity" that you can reuse in different activities).
- You can combine multiple fragments in a single activity to build a multi-pane UI and reuse a fragment in multiple activities.
- A fragment must always be hosted in an activity and the fragment's lifecycle is directly affected by the host activity's lifecycle.
  - For example, when the activity is paused, so are all fragments in it, and when the activity is destroyed, so are all fragments

Fragments

- When you add a fragment as a part of your activity layout, it lives in a ViewGroup inside the activity's view hierarchy and the fragment defines its own view layout.
- You can insert a fragment into your activity layout by declaring the fragment in the activity's layout file, as a <fragment> element, or from your application code by adding it to an existing ViewGroup

Importance of Fragments

- There are many use cases for fragments but the most common use cases include:
- Reusing View and Logic Components - Fragments enable re-use of parts of your screen including views and event logic over and over in different ways across many distinct activities. For example, using the same list across different data sources within an app.
- Tablet Support - Often within apps, the tablet version of an activity has a substantially different layout from the phone version which is different from the TV version. Fragments enable device-specific activities to reuse shared elements while also having differences.
- Screen Orientation - Often within apps, the portrait version of an activity has a substantially different layout from the landscape version. Fragments enable both orientations to reuse shared elements while also having differences.

- To reiterate, in a fragment-based architecture, the activities are for navigation and the fragments are for views and logic.

Creating a Fragment

- A fragment, like an activity, has an XML layout file and a Java class.

Defining Fragment Layout Resource

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

    <TextView
        android:id="@+id/tv1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="TextView" />

    <Button
        android:id="@+id/btn1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button" />

</LinearLayout>
```

Creating a Fragment

- To create a fragment, you must create a subclass of Fragment
- It contains callback methods like an activity, such as onCreate(), onStart(), onPause(), and onStop()
- A fragment is used as part of an activity's user interface and contributes its own layout to the activity.
- To provide a layout for a fragment, you must implement the onCreateView() callback method, which the Android system calls when it's time for the fragment to draw its layout.
- Your implementation of this method must return a View that is the root of your fragment's layout.

Creating a Fragment

- To return a layout from onCreateView(), you can inflate it from a layout resource defined in XML using the provided LayoutInflater object.

```
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);
    }
}
```

- The container parameter passed to onCreateView() is the parent ViewGroup (from the activity's layout) in which your fragment layout is inserted.
- The savedInstanceState parameter is a Bundle that provides data about the previous instance of the fragment, if the fragment is being resumed.

Creating a Fragment

- 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 during inflation. (In this case, this is false because the system is already inserting the inflated layout into the container—passing true would create a redundant view group in the final layout.)

Adding a Fragment to an Activity

- A fragment contributes a portion of UI to the host activity

- There are two ways you can add a fragment to the activity layout:
  - Declare the fragment inside the activity's layout file (statically using XML)
  - Or, programmatically add the fragment to an existing ViewGroup (dynamically using Java)

Add Fragment Statically

- To add the fragment statically, simply embed the fragment in the activity's xml layout file:

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

    <fragment
        android:name="com.nomadlearner.fragments.TestFragment"
        android:id="@+id/testFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</LinearLayout>
```

Add Fragment Dynamically

- The second way is by adding the fragment dynamically in Java using the FragmentManager
- The FragmentManager class and the FragmentTransaction class allow you to add, remove and replace fragments in the layout of your activity at runtime.
- In this case, you want to add a "placeholder" container (usually a FrameLayout) to your activity where the fragment is inserted at runtime:

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

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


</LinearLayout>
```

Add Fragment Dynamically

- Then FragmentManager can be used to create a FragmentTransaction which allows us to add fragments to the FrameLayout at runtime:

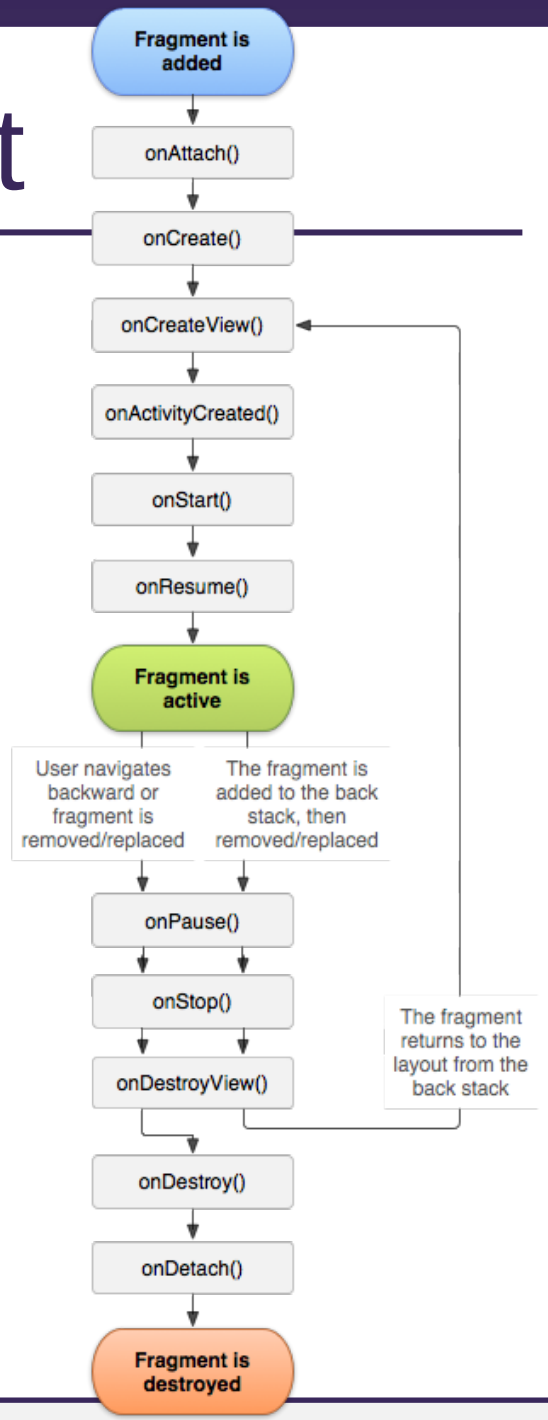
```
// Begin the transaction
FragmentTransaction ft = getSupportFragmentManager().beginTransaction();

// Replace the contents of the container with the new fragment
ft.replace(R.id.your_placeholder, new TestFragment());

// or ft.add(R.id.your_placeholder, new TestFragment());
// Complete the changes added above
ft.commit();
```

The Lifecycle of a Fragment

- The most common ones to override are
  - onCreateView which is in almost every fragment to setup the inflated view,
  - onCreate for any data initialization and
  - onActivityCreated used for setting up things that can only take place once the Activity has been fully created.



The Lifecycle of a Fragment

- onAttach() is called when a fragment is connected to an activity.
- onCreate() is called to do initial creation of the fragment.
- onCreateView() is called by Android once the Fragment should inflate a view.
- onViewCreated() is called after onCreateView() and ensures that the fragment's root view is non-null. Any view setup should happen here. E.g., view lookups, attaching listeners.
- onActivityCreated() is called when host activity has completed its onCreate() method.
- onStart() is called once the fragment is ready to be displayed on screen.
- onResume() - Allocate “expensive” resources such as registering for location, sensor updates, etc.
- onPause() - Release “expensive” resources. Commit any changes.

The Lifecycle of a Fragment

- onDestroyView() is called when fragment's view is being destroyed, but the fragment is still kept around.
- onDestroy() is called when fragment is no longer in use.
- onDetach() is called when fragment is no longer connected to the activity.

Looking Up a Fragment Instance

- Often, we need to lookup or find a fragment instance within an activity layout file.
- Two methods for looking up an existing fragment instance:
  1. ID - Lookup a fragment by calling findFragmentById on the FragmentManager

```
TestFragment fragmentDemo = (TestFragment)
        getSupportFragmentManager().findFragmentById(R.id.testFragment);
```

  2. Tag - Lookup a fragment by calling findFragmentByTag on the FragmentManager
     - If the fragment was dynamically added at runtime within an activity, then we can lookup this fragment by tag by calling `findFragmentByTag` on the `FragmentManager`

```
// Let's first dynamically add a fragment into a frame container
getSupportFragmentManager().beginTransaction().
        replace(R.id.your_placeholder, new TestFragment(), "SOMETAG").
        commit();
// Now later we can lookup the fragment by tag
TestFragment fragmentTest = (TestFragment)
        getSupportFragmentManager().findFragmentByTag("SOMETAG");
```

Using Fragments

- Create a new Android project and name it Fragments
- In the res/layout folder, add a new layout resource file and name it fragment1.xml.
- Populate it with the following code

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:background="#00FF00">
<TextView
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="This is fragment #1"
      android:textColor="#000000"
      android:textSize="25sp" />
</LinearLayout>
```

Using Fragments

- Also in the res/layout folder, add another new layout resource file and name it fragment2.xml
- Populate it as follows

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:background="#FFFE00">
<TextView
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="This is fragment #2"
      android:textColor="#000000"
      android:textSize="25sp" />
</LinearLayout>
```

Using Fragments

- In activity_main.xml, replace all with in the following code:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout android:orientation="vertical"
    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.wenbing.fragments.MainActivity">
     <fragment
             android:name="com.username.fragments.Fragment1"
             android:id="@+id/fragment1"
             android:layout_weight="1"
             android:layout_width="fill_parent"
             android:layout_height="match_parent" />
     <fragment
             android:name="com. username.fragments.Fragment2"
             android:id="@+id/fragment2"
             android:layout_weight="1"
             android:layout_width="fill_parent"
             android:layout_height="match_parent" />
</LinearLayout>
```

Using Fragments

- Add two Java class files and name them Fragment1.java and Fragment2.java

Fagments1.java

```
package ....;
import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
public class Fragment1 extends Fragment {
       @Override
       public View onCreateView(LayoutInflater inflater,
                 ViewGroup container, Bundle savedInstanceState) {
              //---Inflate the layout for this fragment---
              return inflater.inflate(R.layout.fragment1, container,
false);
       }
}
```

Using Fragments

Fagments2.java

```
package ....;
import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
public class Fragment2 extends Fragment {
       @Override
       public View onCreateView(LayoutInflater inflater,
                ViewGroup container, Bundle savedInstanceState) {
              //---Inflate the layout for this fragment---
              return inflater.inflate(R.layout.fragment2, container, false);
       }
}
```

To draw the UI for a fragment, override the onCreateView() method. This method returns a View object

use a LayoutInflater object to inflate the UI from the specified XML file

The container argument refers to the parent ViewGroup, which is the activity in which you are trying to embed the fragment

Using Fragments

Adding Fragments Dynamically

- In the same project, modify the activity_main.xml file by commenting out the two <fragment> elements

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout android:orientation="vertical"
     ......
     tools:context="com. username.fragments.MainActivity">
<!--
     <fragment
            ......
     <fragment
            ......
-->
</LinearLayout>
```

Adding Fragments Dynamically

- Add the bolded lines in the following code to the MainActivity.java file

```
import android.app.Activity;
import android.app.FragmentManager;
import android.app.FragmentTransaction;
import android.os.Bundle;
import android.util.DisplayMetrics;
public class MainActivity extends Activity {
     @Override
     public void onCreate(Bundle savedInstanceState) {
          super.onCreate(savedInstanceState);
          FragmentManager fragmentManager = getFragmentManager();
          FragmentTransaction fragmentTransaction =
        fragmentManager.beginTransaction();
          //---get the current display info---
          DisplayMetrics display = this.getResources().getDisplayMetrics();
           int width = display.widthPixels;  int height = display.heightPixels;
.....
```

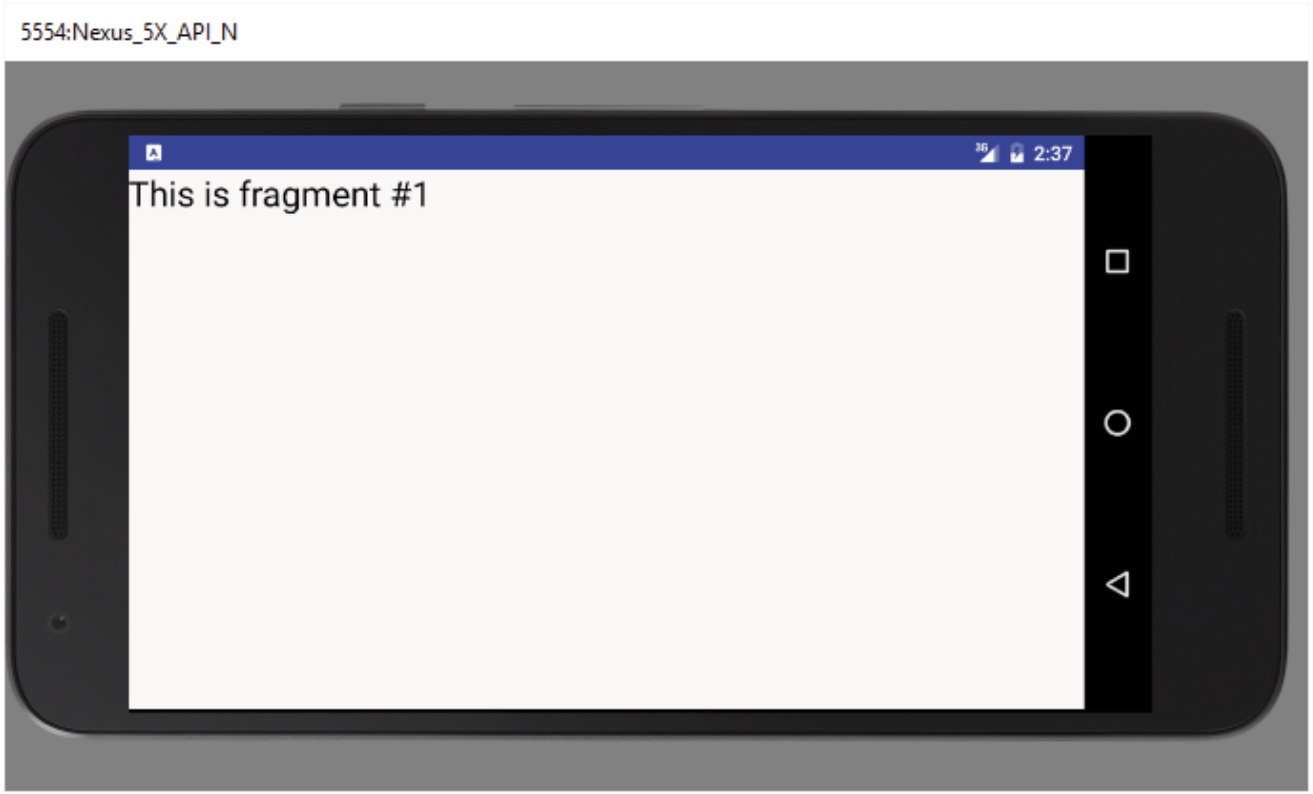
Remove:
setContentView(R.layout.*activity_main*);

Adding Fragments Dynamically

```
        if (width> height)
        {
              //---landscape mode---
              Fragment1 fragment1 = new Fragment1();
              // android.R.id.content refers to the content view of the activity
              fragmentTransaction.replace(android.R.id.content, fragment1);
        }
        else
        {
              //---portrait mode---
              Fragment2 fragment2 = new Fragment2();
              fragmentTransaction.replace(android.R.id.content, fragment2);
        }
        fragmentTransaction.commit();
    }
}
```

FragmentTransactionclass to perform fragment transactions (such as add, remove, or replace) in your activity

Adding Fragments Dynamically



Interactions between Fragments

- An activity might contain two or more fragments working together to present a coherent UI to the user
  - E.g.: the user taps on an item in that fragment, details about the selected item might be displayed in another fragment
- Continue in the same project, add bolded statements to Fragment1.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:background="#00FF00">
<TextView
      android:id="@+id/lblFragment1"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="This is fragment #1"
      android:textColor="#000000"
      android:textSize="25sp" />
```

Interactions between Fragments

- Add the following bolded lines to fragment2.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      ......
<TextView
      ..... />
<Button
     android:id="@+id/btnGetText"
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:text="Get text in Fragment #1"
     android:textColor="#000000"
     android:onClick="onClick" />
</LinearLayout>
```

Interactions between Fragments

- In activity_main.xml, uncomment the two fragments:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout android:orientation="vertical"
      ......
      tools:context="com. username.fragments.MainActivity">
      <fragment
              android:name="com.username.fragments.Fragment1"
              android:id="@+id/fragment1"
              android:layout_weight="1"
              android:layout_width="fill_parent"
              android:layout_height="match_parent" />
       <fragment
              android:name="com. username.fragments.Fragment2"
              android:id="@+id/fragment2"
              android:layout_weight="1"
              android:layout_width="fill_parent"
              android:layout_height="match_parent" />
</LinearLayout>
```

Interactions between Fragments

- Modify the MainActivity.java file by commenting out the code added in the previous step, and add setContentView() back

```
public class MainActivity extends Activity {
     @Override
     public void onCreate(Bundle savedInstanceState) {
          super.onCreate(savedInstanceState);
         setContentView(R.layout.activity_main);
          /*
          FragmentManager fragmentManager = getFragmentManager();
          FragmentTransaction fragmentTransaction =
       fragmentManager.beginTransaction();
          //---get the current display info---
          DisplayMetrics display = this.getResources().getDisplayMetrics();
           int width = display.widthPixels;  int height = display.heightPixels;
.....
```

Interactions between Fragments

```
            if (width> height)
            {
                  //---landscape mode---
                  Fragment1 fragment1 = new Fragment1();
                  // android.R.id.content refers to the content view of the activity
                  fragmentTransaction.replace(android.R.id.content, fragment1);
            }
            else
            {
                  //---portrait mode---
                  Fragment2 fragment2 = new Fragment2();
                  fragmentTransaction.replace(android.R.id.content, fragment2);
            }
            fragmentTransaction.commit();
            */
      }
}
```

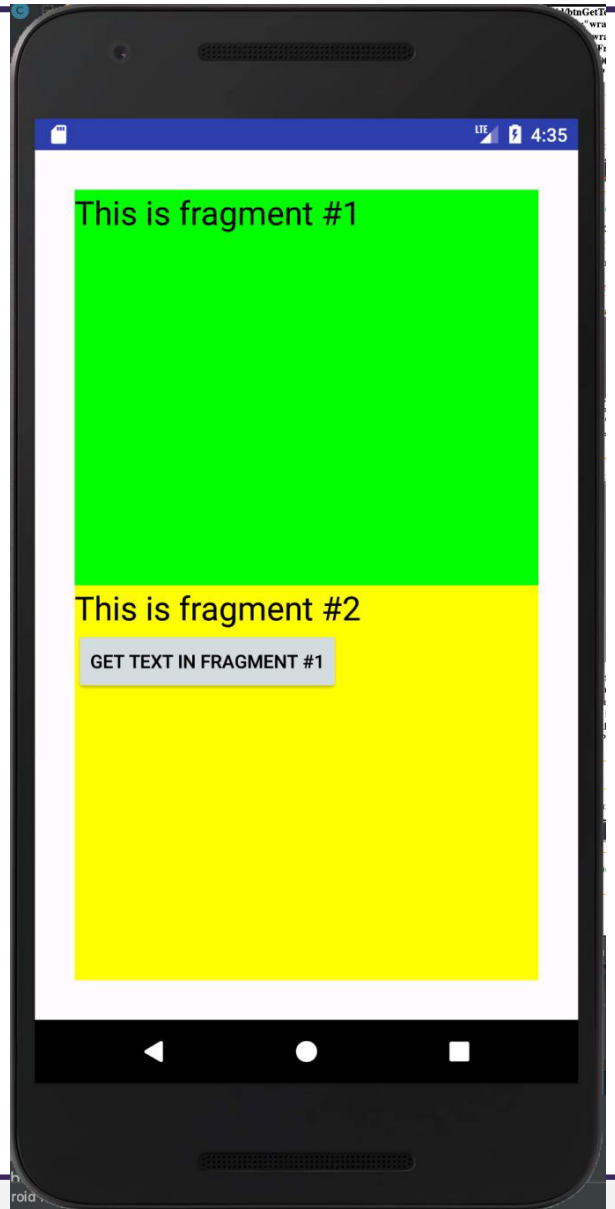
Interactions between Fragments

- Add the bolded statements to the Fragment2.java

Fagments2.java

```
public class Fragment2 extends Fragment {
     @Override
     public View onCreateView(LayoutInflater inflater,
        ViewGroup container, Bundle savedInstanceState) {......
     }
    @Override
    public void onStart() {
         super.onStart();
         //---Button view---
         Button btnGetText = (Button)getActivity().findViewById(R.id.btnGetText);
         btnGetText.setOnClickListener(new View.OnClickListener() {
               public void onClick(View v) {
                    TextView lbl = (TextView)getActivity().findViewById(R.id.lblFragment1);
                    Toast.makeText(getActivity(), lbl.getText(),Toast.LENGTH_SHORT).show();
               }
         });
      }
}
```

Interactions between Fragments



In Class Task # 04

- Using the android studio or any other IDE, perform the following tasks/experiment to reinforce what we learned in class regarding Fragments

- Crate a simple Notepad app with custom keypad in which
  - Top fragment: display the note entered
  - Bottom fragment: display several rows of letters, numbers, and symbols (each as a button) for text input; when a user push a button, the corresponding input is appended in the top fragment

Recommended Readings

- Page # 75 to 92, Chapter # 03: Activities, Fragments, and Intents from Beginning Android Programming with Android Studio, 4th Edition by J. F. DiMarzio, Wrox, 2017
- Page # 94 to 98, Chapter # 03: Activities, Fragments, and Intents from Beginning Android Programming with Android Studio, 4th Edition by J. F. DiMarzio, Wrox, 2017
- User Guide: https://developer.android.com/training/notify-user/build-notification
- Creating and Using Fragments at https://guides.codepath.com/android/creating-and-using-fragments