WebView
Fragments
Navigation Drawer

http://www.android.com/
Web Apps

• You can make your web content available to users in two ways
  – In a traditional web browser
  – In an Android application, by including a WebView in the layout

• From Android 4.4 a new WebView component based on the Chromium open source project
  – https://developer.chrome.com/multidevice/webview/overview
Android WebView is a system component powered by Chrome that allows Android apps to display web content. This component is pre-installed on your device and should be kept up to date to ensure you have the latest security updates and other bug fixes.

WebView 1 (old way)

// There are two ways to obtain a WebView object.
// It can be instantiated from the constructor:
WebView webView = new WebView(this);
// Alternatively a WebView can be used in a layout and declared in the activity
WebView webView = (WebView) findViewById(R.id.tutView);
// After the object is retrieved, a web page can be displayed
// using the loadURL() method
webView.loadUrl("http://www.google.com/");
// The WebSettings class can be used to define the features of the browser
WebSettings webSettings = webView.getSettings();
webSettings.setXYZ(); // or get a feature - webSettings.getXYZ();

<?xml version="1.0" encoding="utf-8"?>
<WebView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/tutView">
</WebView>

@override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_tut_view);
    // Returns the intent that started this activity
    Intent launchingIntent = getIntent();
    String content = launchingIntent.getData().toString();
    WebView viewer = (WebView) findViewById(R.id.tutView);
    WebSettings webSettings = viewer.getSettings();
    webSettings.setBuiltInZoomControls(true);
    viewer.loadUrl(content);
WebView 2 (current way)

- Overriding the url and read internally from assets folder

```java
private static final String assetUri = "file:///android_asset/TIJ/";

private void loadCustomSite() {
    // A very good WebView tutorial: https://developer.chrome.com/multidevice/webview/gettingstarted
    WebSettings ws = webview.getSettings();
    ws.setJavaScriptEnabled(true);
    final Activity activity = this;
    webview.setWebChromeClient(new WebChromeClient() {
        @Override
        public void onProgressChanged(WebView view, int progress) {
            // Activities and WebViews measure progress with different scales.
            // The progress meter will automatically disappear when we reach 100%
            activity.setProgress(progress * 1000);
        }
    });
    webview.setWebViewClient(new WebViewClient() {
        @Override
        public void onReceivedError(WebView view, int errorCode, String description, String failingUrl) {
            Toast.makeText(activity, "Oh no! " + description, Toast.LENGTH_SHORT).show();
        }
        @Override
        public boolean shouldOverrideUrlLoading(WebView view, String url) {
            if (url.startsWith(changeUri)) {
                Log.d(TAG, "url " + url);
                String newUrl = assetUri + url.replace(changeUri, ":" + newUrl);
                Log.d(TAG, "newUrl " + newUrl);
                webview.loadUrl(newUrl);
                Toast.makeText(activity, newUrl, Toast.LENGTH_LONG).show();
                return true;
            }
            return false;
        }
    });
    //webview.loadUrl(assetUri + "index.htm");
    webview.loadUrl(assetUri + "TIJ3.htm");
}
```
A fragment is an independent component which can be used (embedded) by an activity.

A fragment encapsulate functionality so that it is easier to reuse within activities and layouts.

A fragment runs in the context of an activity (it is however directly affected by the host activity's lifecycle), but it has its own lifecycle and typically its own user interface.

A fragment can be added dynamically (code) or statically (xml) to an activity.

Android introduced fragments in Android 3.0 (API level 11), primarily to support more dynamic and flexible UI designs on large screens.

By dividing the layout of an activity into fragments, you become able to modify the activity's appearance at runtime and preserve those changes in a back stack that's managed by the activity.

Fragments 2

- For example - in a news application - the application can embed two fragments in Activity A, when running on a tablet-sized device.
- However, on a handset-sized screen, there's not enough room for both fragments, so Activity A includes only the fragment for the list of articles, and when the user selects an article, it starts Activity B, which includes the second fragment to read the article.
- Thus, the application supports both tablets and handsets by reusing fragments in different combinations.
- All Fragment-to-Fragment communication is done through the associated Activity. Two Fragments should never communicate directly!
Fragment lifecycle

- To create a fragment, you must create a subclass of Fragment - or an existing subclass of it as: DialogFragment, ListFragment or PreferenceFragment etc.
- The Fragment class has code that looks a lot like an Activity. It contains callback methods similar to an activity, such as `onCreate()`, `onStart()`, `onPause()` and `onStop()`
  - When the activity get these lifecycle calls every fragment in the activity get them as well
- In fact, if you're converting an existing Android application to use fragments, you might simply move code from your activity's callback methods into the respective callback methods of your fragment
- The back stack handling is the major difference!

http://developer.android.com/guide/components/fragments.html#Lifecycle
Adding a user interface

- 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 `onCreateView()` method must return a View that is the root of your fragment's layout.
  - Note: If your fragment is a subclass of `ListFragment`, the default implementation returns a `ListView` from `onCreateView()`, so you don't need to implement it (just as with `ListFragmentActivity`).
- The `container` parameter passed to `onCreateView()` is the parent ViewGroup (from the activity's layout) in which your fragment layout will be inserted.
- The `savedInstanceState` parameter is a Bundle that provides data about the previous instance of the fragment, if the fragment is being resumed - just as with Activities.

```java
public static class ExampleFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        // Inflate the layout for this fragment, false == means root of the inflated file
        return inflater.inflate(R.layout.example_fragment, container, false);
    }
}
```
Add fragment via XML to an Activity

• Usually a fragment contributes a portion of UI to the host activity, which is embedded as a part of the activity's overall view hierarchy
• As usual with Views you can add the fragment in two ways to an activity
  – Via XML in the activity's layout file - explicit as below OR implicit via a FrameLayout - ie. programmatically add the fragment to an existing ViewGroup
• The android:name attribute in the <fragment> specifies the Fragment class to instantiate in the layout
• When the system creates this activity layout, it instantiates each fragment specified in the layout and calls the oncreateView() method for each one, to retrieve each fragment's layout. The system inserts the View returned by the fragment directly in place of the <fragment> element.
• Each fragment requires an unique identifier that the system can use to restore the fragment if the activity is restarted

```xml
<?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>
```
Add and manage the fragment programmatically

- 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 as `R.id.fragment_container` which usually is a **FrameLayout** (an area to normally display a single item).
- To make fragment transactions in your activity (such as add, remove or replace a fragment) you must use APIs from FragmentTransaction

```xml
<?xml version="1.0" encoding="utf-8"?>
<!- put this in your layout where you want to show your fragment -->
<FrameLayout android:id="@+id/fragment_container"
             android:layout_width="match_parent" android:layout_height="match_parent"/>
...
```

```java
// get an instance of FragmentTransaction from the Activity
FragmentManager fragmentManager = getFragmentManager();
FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();

ExampleFragment fragment = new ExampleFragment(), newFragment = new ExampleFragment();
// specify the fragment to add and the view in which to insert it
fragmentTransaction.add(R.id.fragment_container, fragment);
// or replace a fragment in the fragment container with a new fragment
fragmentTransaction.replace(R.id.fragment_container, newFragment, optionalTagName);
// in this case add the transaction to the back stack so the user can press back button and return to it
fragmentTransaction.addToBackStack(optionalTagName);  // the parameter is an optional name/tag
// make changes take effect
fragmentTransaction.commit();
```
Add and manage the fragment programmatically 2

• With the FragmentManager you can also manage the fragments, for example get fragments that exist in the activity with `findFragmentById()` or `findFragmentByTag()`, register listener interface and pop fragments from the back stack.

```java
// Create a new fragment in a generic way
String tag = FragmentOne.class.getSimpleName();
Class fragmentClass = FragmentOne.class;
Fragment fragment = (Fragment) fragmentClass.newInstance();

// Insert the fragment by replacing any existing fragment
FragmentManager fragmentManager = getFragmentManager();
fragmentManager.beginTransaction().replace(R.id.container, fragment, tag).commit();

// Later on from Activity - get the fragment by tag and call a method in the fragment if visible
Fragment curFragment =
    (Fragment) fragmentManager.findFragmentByTag(FragmentOne.class.getSimpleName());

if (curFragment != null && curFragment.isVisible()) {
    ((FragmentOne) curFragment).setLocationAndDateTime(mCurrentLocation);
}
```
Fragments with arguments

• In certain cases, your fragment may want to accept arguments

```java
public class DemoFragment extends Fragment {
    // Fragment must have only a constructor with no arguments
    public DemoFragment() {} // Creates a new fragment given an int and title

    public static DemoFragment newInstance(int someInt, String someTitle) {
        DemoFragment fragmentDemo = new DemoFragment();
        Bundle args = new Bundle();
        args.putInt("someInt", someInt);
        args.putString("someTitle", someTitle);
        fragmentDemo.setArguments(args);
        return fragmentDemo;
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // Get back arguments
        int SomeInt = getArguments().getInt("someInt", 0);
        String someTitle = getArguments().getString("someTitle", "");
    }
}
```

// Within the activity send the parameters
FragmentTransaction ft = getFragmentManager().beginTransaction();
DemoFragment fragmentDemo = DemoFragment.newInstance(5, "my title");
ft.replace(R.id.your_placeholder, fragmentDemo);
ft.commit();
```
Communication and callbacks 1

• Communication between Activity ↔ Fragment

// Fragments can access the Activity instance with getActivity() and easily perform
// tasks such as find a view in the activity layout or call public methods etc.
final Activity activity = getActivity();
View listView = activity.findViewById(R.id.list);
((YourActivityClassName) activity).yourPublicMethod();

// Likewise, your Activity can call methods etc. in the fragment by acquiring a reference to the
// Fragment from FragmentManager, using findFragmentById() or findFragmentByIdTag() if no id is available
ExampleFragment fragment = (ExampleFragment) getFragmentManager().findFragmentById(R.id.example_fragment);
ExampleFragment fragment = (ExampleFragment) getFragmentManager().findFragmentByTag("fragmentname");

• Creating event callbacks to the Activity
  – In some cases, you might need a fragment to share events with
    the Activity. A good way to do that is to define a callback
    interface inside the Fragment and require that the host activity
    implements it

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

public class MainActivity extends Activity
implements FragmentA.OnArticleSelectedListener {
  ...
public static class FragmentA extends ListFragment {
    OnArticleSelectedListener mListener;
    // Container Activity must implement this interface
    public interface OnArticleSelectedListener {
        public void onArticleSelected(Uri articleUri);
    }
    // ensure that the host activity implements the interface, if not an exception is thrown
    // On success, the mListener member holds a reference to activity's implementation of
    // OnArticleSelectedListener, so that fragment A can share events with the activity by
    // calling methods defined by the OnArticleSelectedListener interface.
    @Override
    public void onAttach(Activity activity) {
        super.onAttach(activity);
        super.onAttach(context);
        try {
            mListener = (OnArticleSelectedListener) activity;
        } catch (ClassCastException e) {
            throw new ClassCastException(activity.toString() +
            " must implement OnArticleSelectedListener");
        }
    }
    // if fragment A is an extension of ListFragment, each time the user clicks a list item,
    // the system calls onListItemClick() in the fragment, which then calls onArticleSelected()
    // to share the event with the host activity
    @Override
    public void onListItemClick(ListView l, View v,
    int position, long id) {
        // Append the clicked item's row ID with the content provider Uri
        Uri noteUri = ContentUris.withAppendedId(ArticleColumns.CONTENT_URI, id);
        // Send the event and Uri to the host Activitys overridden onArticleSelected() method
        mListener.onArticleSelected(noteUri);
    }
    ...
}
Adding items to the Fragment/App Bar

• Fragments can contribute menu items to the activity's Options Menu (and, consequently, the App Bar) by implementing `onCreateOptionsMenu()`
  – To receive calls, however, you must call `setHasOptionsMenu()` during `onCreate()`
  – The fragment also receives callbacks to `onOptionsItemSelected()` when a menu item is selected

• You can also register a view in your fragment layout to provide a context menu by calling `registerForContextMenu()`
  – When the user opens the context menu, the fragment receives a call to `onCreateContextMenu()`
  – When the user selects an item, the fragment receives a call to `onContextItemSelected()`

• **Note:** Although your fragment receives an on-item-selected callback for each menu item it adds, the activity is first to receive the respective callback when the user selects a menu item. If the activity's implementation of the on-item-selected callback does not handle the selected item, then the event is passed to the fragment's callback
Supporting Different Screen Sizes

- Methods beyond flexible/relative layouts and \texttt{wrap\_content/match\_parent}
- **Size qualifiers** as: \texttt{res/layout-large/main.xml}, two-pane layout
  - Layout will be selected on devices with screens classified as large, for example 7" tablets and above (works on pre-3.2)
- **Smallest-width Qualifier** allows you to target screens that have a certain minimum width given in dp (does not work on pre-3.2)
  - Using a layout as: \texttt{res/layout-sw600dp/main.xml}, means that devices whose smallest width is greater than or equal to 600dp will use a two-pane layout (define the fragments \texttt{layout\_width} or \texttt{layout\_weight})
- **Layout Aliases**
  - Let your layout file point out some other layout file to avoid duplication
- **Orientation Qualifiers** ++
  - Identify device screen properties and apply suitable layouts
- **Nine-patch Bitmaps**
  - Stretched image resources
- **Sample app:** NewsReader
Supporting Multiple Screen Sizes 1
http://developer.android.com/guide/practices/screens_support.html

- Density-independent pixel (dp) - A virtual pixel unit that you should use when defining UI layout, to express layout dimensions or position in a density-independent way (dpi/ppi = dots/pixels per inch)
  - For example, on a 240 dpi screen, 1 dp equals 1.5 physical pixels → pixels per inch = dp * (dpi / 160). For a xxhdpi (3.0x) screen it is 3 pixels

- Layout-small, layout-large or layout-xlarge is deprecated

- Use smallestWidth sw<N>dp after Android 3.2 – note dp
  - For example: res/layout-sw600dp/ - defines the smallest available width required by your layout resources

Baseline = mdpi = 160 dpi

Figure 1. Illustration of how Android roughly maps actual sizes and densities to generalized sizes and densities (figures are not exact).
• We also have w<N>dp and h<N>dp which is minimum available width and height in dp units
• Example 800x1280px 7” tablet → 215,6 ppi falls into hdpi (1,5x)
  – We should use 800/1,5 = sw533dp, but if we set layout-sw480dp we handle 720x1280 pixel tablets as well
• 1080x1920px 5,2” phone → 423,6 ppi falls into xxhdpi (3.0x)
  – 1080/3 = 360 so if we would like another layout on this phone we must set layout-sw360dp
• We could also use the w<N>dp (1920/3 = 640), setting layout-sw600dp gives any screen with 600dp available width or more the desired layout whether the device is in landscape or portrait orientation

Calculation of screen PPI (Pixels Per Inch)

1. Calculate diagonal resolution in pixels using the Pythagorean theorem:
   \[ d_p = \sqrt{w_p^2 + h_p^2} \]
2. Calculate PPI:
   \[ PPI = \frac{d_p}{d_i} \]

where
- \( d_p \) is diagonal resolution in pixels
- \( w_p \) is width resolution in pixels
- \( h_p \) is height resolution in pixels
- \( d_i \) is diagonal size in inches (this is the number advertised as the size of the display).
Supporting Multiple Screen Sizes

Wrap-up

• Get and set the density
  – From a command shell get the density with: adb shell wm density
  – To set the density just put a number after as: adb shell wm density 320
  – With Android Nougat users can control the DPI under Settings > Display > Display size

• Calculate the density
  – The physical density of a 5.2 inch screen with 1920 x 1080 is 480
    • This translates into: 480/160 = xxhdpi 3
  – The formula is actual-dpi / 160. (Everything is scaled to 160 dpi)
  – To get the physical density = sqrt((wp * wp) + (hp * hp)) / di
  – Where:
    • wp is width resolution in pixels, hp is height resolution in pixels, and di is diagonal size in inches. Example a 8.3 inch tablet with 1920 x 1200 pixels
      • (1200^2 + 1920^2)^0.5 = 2265 / 8.3 = 273 dpi or ppi
      • 273/160 = 1.7. Will fall into xhdpi 2 or hdpi 1.5 (1200/2 = sw-600dp)
Fragment example 1

Using the FragmentBasics sample (slightly modified – no support lib)
http://developer.android.com/training/basics/fragments/creating.html

<!-- Adds Fragments to an Activity, res/layout-sw300dp/news_articles.xml -->
<LinearLayout 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"
android:orientation="horizontal"
android:baselineAligned="false">
  <fragment
    android:id="@+id/headlines_fragment"
    android:name="com.example.android.fragments.HeadlinesFragment"
    android:layout_width="0dp"
    android:layout_height="match_parent"
    android:layout_weight="1"
    tools:layout="@android:layout/simple_list_item_1"/>
  <fragment
    android:id="@+id/article_fragment"
    android:name="com.example.android.fragments.ArticleFragment"
    android:layout_width="0dp"
    android:layout_height="match_parent"
    android:layout_weight="2"
    tools:layout="@layout/article_view"/>
</LinearLayout>

<!-- Adds a View to a Fragment, res/layout/article_view.xml -->
<ScrollView xmlns:android=
=http://schemas.android.com/apk/res/android"
android:layout_width="fill_parent"
android:layout_height="fill_parent">
  <TextView
    android:id="@+id/article"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:padding="16dp"
    android:textSize="18sp"/>
</ScrollView>

<!-- frame layouts are primarily used to organize individual or overlapping view controls on the screen in a container
Adds a Layout to an Activity, res/layout/news_articles.xml -->
<FrameLayout xmlns:android=
=http://schemas.android.com/apk/res/android"
android:id="@+id/fragment_container"
android:layout_width="match_parent"
android:layout_height="match_parent"/>

headline layout alias

article layout alias – use same article layout regardless of screen size

two-pane layout
	one-pane layout
public class MainActivity extends Activity implements HeadlinesFragment.OnHeadlineSelectedListener {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.news_articles);
        // Check whether the activity is using the layout version with the fragment_container FrameLayout.
        // If so we run on a smaller screen, we must add the first fragment by code, else we do nothing
        // since the two-pane layout (res/layout-swXXXdp) is defined and inflated in this case
        if (findViewById(R.id.fragment_container) != null) {
            // Create an instance of a HeadlineFragment which reads the headlines from an ArrayAdapter and layout
            // of type android.R.layout.simple_list_item_XXX for the one-pane layout
            HeadlinesFragment firstFragment = new HeadlinesFragment();
            // Add the created fragment to the 'fragment_container' FrameLayout at runtime
            getFragmentManager().beginTransaction().add(R.id.fragment_container, firstFragment).commit();
        }
    }
}

public class HeadlinesFragment extends ListFragment {
    OnHeadlineSelectedListener mCallback;

    public interface OnHeadlineSelectedListener {
        // Method in MainActivity is called by HeadlinesFragment when a list item is selected
        public void onArticleSelected(int position);
    }
    @Override
    public void onListItemClick(ListView l, View v, int position, long id) {
        // Notify the parent activity of selected item
        mCallback.onArticleSelected(position);
        // Set the item as checked to be highlighted when in two-pane layout
        getListView().setItemChecked(position, true);
    }
}
Fragment example 3

```java
public class MainActivity extends Activity implements HeadlinesFragment.OnHeadlineSelectedListener {
    // called from HeadlinesFragment.onListItemClick()
    @Override
    public void onArticleSelected(int position) {
        // The user selected the headline of an article from the HeadlinesFragment

        // Capture the article fragment from the activity layout
        ArticleFragment articleFrag = (ArticleFragment)
            getSupportFragmentManager().findFragmentById(R.id.article_fragment);

        if (articleFrag != null) {
            articleFrag.updateArticleView(position);
        } else {
            // If the frag is not available, we're in the one-pane layout and must swap frags...
        }
    }
}
```

```java
public class ArticleFragment extends Fragment {
    final static String ARG_POSITION = "position";
    int mCurrentPosition = -1;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        // If activity recreated (such as from screen rotate), restore the previous article selection
        // set by onSaveInstanceState(). This is primarily necessary when in the two-pane layout.
        if (savedInstanceState != null) {
            mCurrentPosition = savedInstanceState.getInt(ARG_POSITION);
        }

        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.article_view, container, false);
    }
}
```
} else {
    // If the frag is not available, we're in the one-pane layout and must swap frags...

    // Create fragment and give it an argument for the selected article
    ArticleFragment newFragment = new ArticleFragment();
    Bundle args = new Bundle();
    args.putInt(ArticleFragment.ARG_POSITION, position);

    /*
     * Supply the construction arguments for this fragment. This can only be called
     * before the fragment has been attached to its activity; that is, you should
     * call it immediately after constructing the fragment. The arguments supplied
     * here will be retained across fragment destroy and creation.
     * The arguments passed in are best to handle in the Fragments onStart() method
     */
    newFragment.setArguments(args);
    FragmentTransaction transaction = getFragmentManager().beginTransaction();

    // Replace whatever is in the fragment_container view with this fragment,
    // and add the transaction to the back stack so the user can navigate back
    transaction.replace(R.id.fragment_container, newFragment);
    transaction.addToBackStack(null);

    // Commit the transaction
    transaction.commit();
}
public class ArticleFragment extends Fragment {
    final static String ARG_POSITION = "position";
    int mCurrentPosition = -1;
    ...
    public void updateArticleView(int position) {
        TextView article = (TextView) getActivity().findViewById(R.id.article);
        article.setText(Ipsum.Articles[position]);
        mCurrentPosition = position;
    }
}
Using Preference Fragments

• Of course after API 11 settings are fragment based

// Fragments provide a more flexible architecture for your application, compared
// to using activities alone, no matter what kind of activity you're building
public static class SettingsFragment extends PreferenceFragment {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // Load the preferences from an XML resource
        addPreferencesFromResource(R.xml.preferences);
    }
    ...
}

// you can then add this fragment to an Activity just
// as you would for any other Fragment
public class SettingsActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // Display the fragment as the main content
        // Return the FragmentManager for interacting with
        // fragments associated with this activity
        getFragmentManager().beginTransaction()
            .replace(android.R.id.content, new SettingsFragment()).commit();
    }
}
Use the AS Wizard - SettingsActivity

- May make things a bit advanced/complicated?
- Wizard adds fragment (two pane) adaptable code (prev. slide)
  - Java
    - SettingsActivity which extends from AppCompatPreferenceActivity
  - XML
    - pref_headers.xml which contains tablet setting headers and the corresponding underlying
      - pref_data_sync.xml,
      - pref_general.xml and
      - pref_notification.xml

// If you choose to use this settings framework you must be able to extend/edit the XML and code a little bit. Specifically note the methods:
public Preference findPreference (CharSequence key) // Finds a Preference based on its key
private static void bindPreferenceSummaryToValue(Preference preference) // Binds a preference's summary to its value

// Bind the summaries of EditText/List/Dialog/Ringtone preferences // to their values. When their values change, their summaries are // updated to reflect the new value, per the Android Design guidelines.
bindPreferenceSummaryToValue(findPreference("example_text"));
bindPreferenceSummaryToValue(findPreference("example_list"));

Full example in MapsProject
public static class MyDialogFragment extends DialogFragment{
    /*
     The Android system requires a Fragment class to have a constructor with no parameters in order to
     instantiate the fragment when it needs to. In a Fragment you have a reference to the activity where
     this fragment is used with the method getActivity()(use it instead of mContext).
     Also the fragment must be declared in its own java file or as an inner static class in another class.
     Otherwise Android will not be able to find your fragment to instantiate it.
     */

    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState)
    {
        AlertDialog.Builder alertDialogBuilder = new AlertDialog.Builder(getActivity());
        alertDialogBuilder.setTitle("Really?");
        alertDialogBuilder.setMessage("Are you sure?");
        // null should be your OnClickListener listener as with cancel
        alertDialogBuilder.setPositiveButton("OK", null);
        alertDialogBuilder.setNegativeButton("Cancel", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                dialog.dismiss();
            }
        });
        return alertDialogBuilder.create();
    }

    // Display the dialog, adding the fragment using an existing transaction and then committing the transaction.
    new MyDialogFragment().show(getFragmentManager(), "MyDialog");
    // And to dismiss the fragment dialog from somewhere
    ((MyDialogFragment)getFragmentManager().findFragmentByTag("MyDialog")).getDialog().dismiss();
One of the most flexible navigational structures available within Android apps is the Navigation Drawer inside a DrawerLayout

- NavigationView was introduced 2015 to ease the development

**Navigation Drawer**
- `<android.support.design.widget.NavigationView`
- Basically a ListView (used before NavigationView)
- `/layout/nav_header.xml`
- `/menu/drawer_view.xml`

**DrawerLayout puts everything together**
- `/layout/activity_main.xml`
- `<android.support.v4.widget.DrawerLayout`
- App bar (Toolbar)
- FrameLayout for fragments
- `<android.support.design.widget.NavigationView`

**Tutorials**
activity_main.xml

<android.support.v4.widget.DrawerLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="true">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">
        <!-- The ActionBar -->
        <include
            layout="@layout/toolbar"
            android:layout_width="match_parent"
            android:layout_height="wrap_content" />
        <!-- The main content view -->
        <FrameLayout
            android:id="@+id/container"
            android:layout_width="match_parent"
            android:layout_height="match_parent" />
    </LinearLayout>
    <!-- The navigation drawer -->
    <android.support.design.widget.NavigationView
        android:id="@+id/nvView"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:layout_gravity="start"
        android:background="@android:color/white"
        app:menu="@menu/drawer_view"
        app:headerLayout="@layout/nav_header" />
</android.support.v4.widget.DrawerLayout>

nav_header.xml

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:padding="@dimen/activity_margin"
    android:background="?attr/colorPrimary"
    android:theme="@style/ThemeOverlay.AppCompat.Dark"
    android:orientation="vertical"
    android:gravity="bottom">
    <!-- The main content view -->
    <!--android:background="?attr/colorPrimary"
     or
    android:background="@drawable/header"
    -->
    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/header"
        android:layout_gravity="top" />
    <TextView
        android:id="@+id/nav_text2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:src="@drawable/header"
        android:layout_gravity="top"/>
</LinearLayout>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="@dimen/navigation_drawer_height"
    android:padding="@dimen/activity_margin"
    android:background="?attr/colorPrimary"
    android:theme="@style/ThemeOverlay.AppCompat.Dark"
    android:orientation="vertical"
    android:gravity="bottom">
    <!-- The main content view -->
    <!--android:background="?attr/colorPrimary"
     or
    android:background="@drawable/header"
    -->
    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/header"
        android:layout_gravity="top"/>
<!--
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical"
        android:gravity="bottom">
        <TextView
            android:id="@+id/nav_text2"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="@string/app_name"
            android:background="@android:color/white"
            android:layout_gravity="top"/>
        </LinearLayout>
-->
Fragment Navigation Drawer 3

Heart of the Fragment Logic
Put a tag on the fragments so you are able to get/find them at a later stage!

```java
public void selectDrawerItem(MenuItem menuItem) {
    // Create a new fragment and specify the fragment to show based on
    // nav item clicked
    Fragment fragment = null;
    Class fragmentClass;
    switch(menuItem.getItemId()) {
        case R.id.nav_first_fragment:
            fragmentClass = FirstFragment.class;
            break;
        case R.id.nav_second_fragment:
            fragmentClass = SecondFragment.class;
            break;
        case R.id.nav_third_fragment:
            fragmentClass = ThirdFragment.class;
            break;
        default:
            fragmentClass = FirstFragment.class;
    }
    try {
        fragment = (Fragment) fragmentClass.newInstance();
    } catch (Exception e) {
        e.printStackTrace();
    }
    // Insert the fragment by replacing any existing fragment
    FragmentManager fragmentManager = getSupportFragmentManager();
    fragmentManager.beginTransaction().replace(R.id.flContent, fragment).commit();
    // Highlight the selected item has been done by NavigationView
    menuItem.setChecked(true);
    // Set action bar title
    setTitle(menuItem.getTitle());
    // Close the navigation drawer
    mDrawer.closeDrawers();
}
```

drawer_view.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <group android:checkableBehavior="single">
        <item
            android:id="@+id/nav_first_fragment"
            android:icon="@drawable/ic_south_park"
            android:title="South Park"/>
        <item
            android:id="@+id/nav_second_fragment"
            android:icon="@drawable/ic_family_guy"
            android:title="Family Guy"/>
        <item
            android:id="@+id/nav_third_fragment"
            android:icon="@drawable/ic_simpsons"
            android:title="Simpsons"/>
        <item
            android:id="@+id/nav_fourth_fragment"
            android:icon="@drawable/ic_futurama"
            android:title="Futurama"/>
        <item android:title="Sub items">
            <menu>
                <group android:checkableBehavior="single">
                    <item
                        android:id="@+id/nav_home"
                        android:icon="@drawable/ic_one"
                        android:title="Sub item 1"/>
                    <item
                        android:id="@+id/nav_about"
                        android:icon="@drawable/ic_two"
                        android:title="Sub item 2"/>
                </group>
            </menu>
        </item>
    </group>
</menu>
```