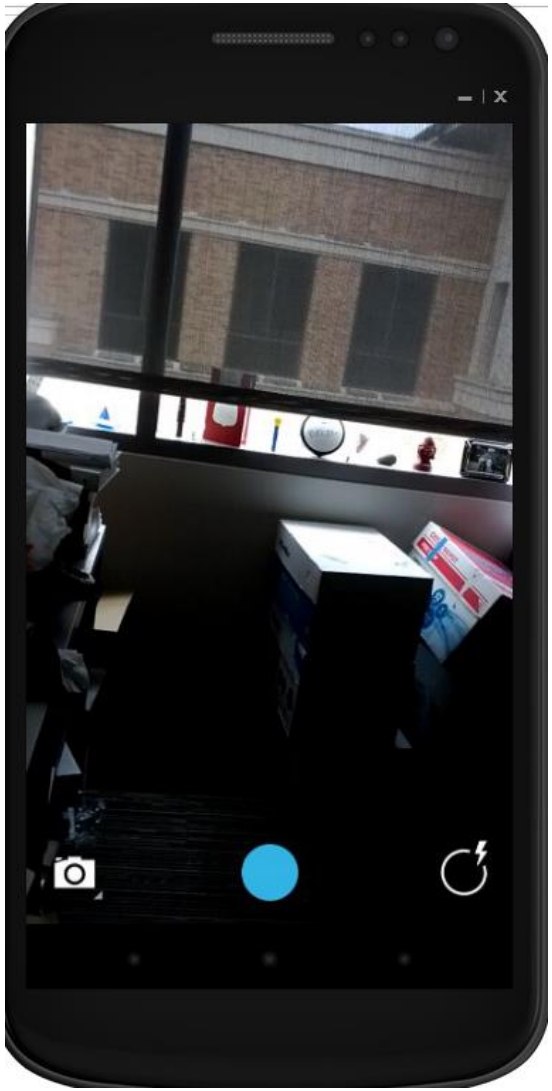
CS371m - Mobile Computing

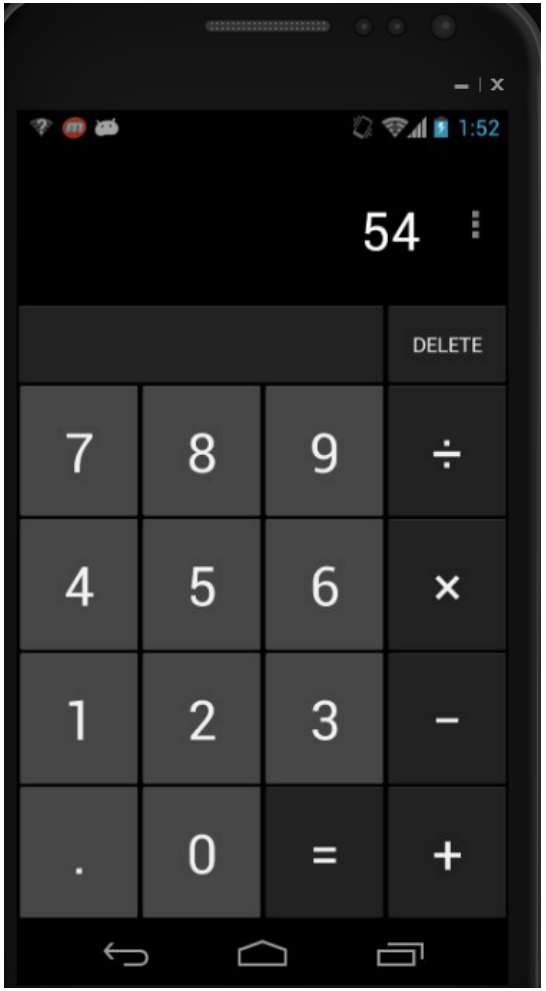
UI Redux, Navigation Patterns, Tabbed Views, Pagers, Drawers

USER INTERFACE NAVIGATION OPTIONS

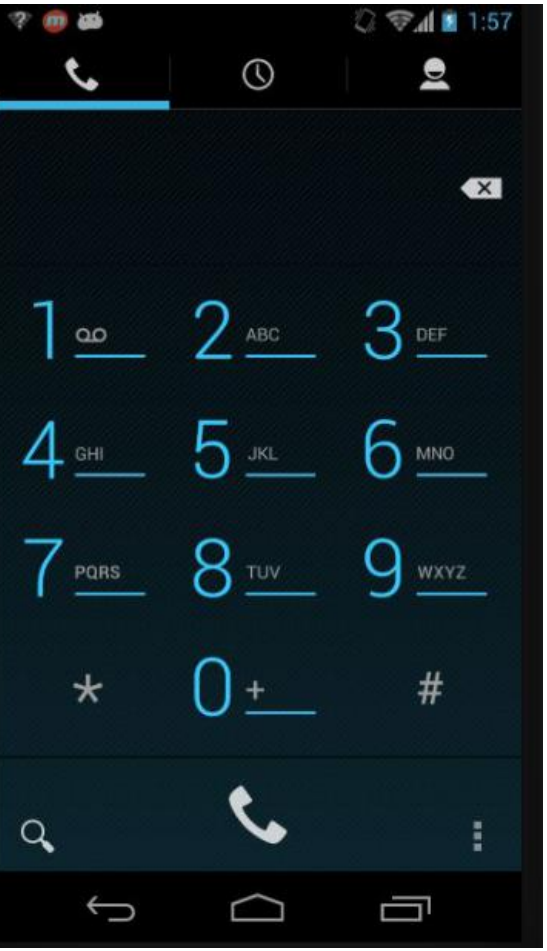
App Navigation Structures



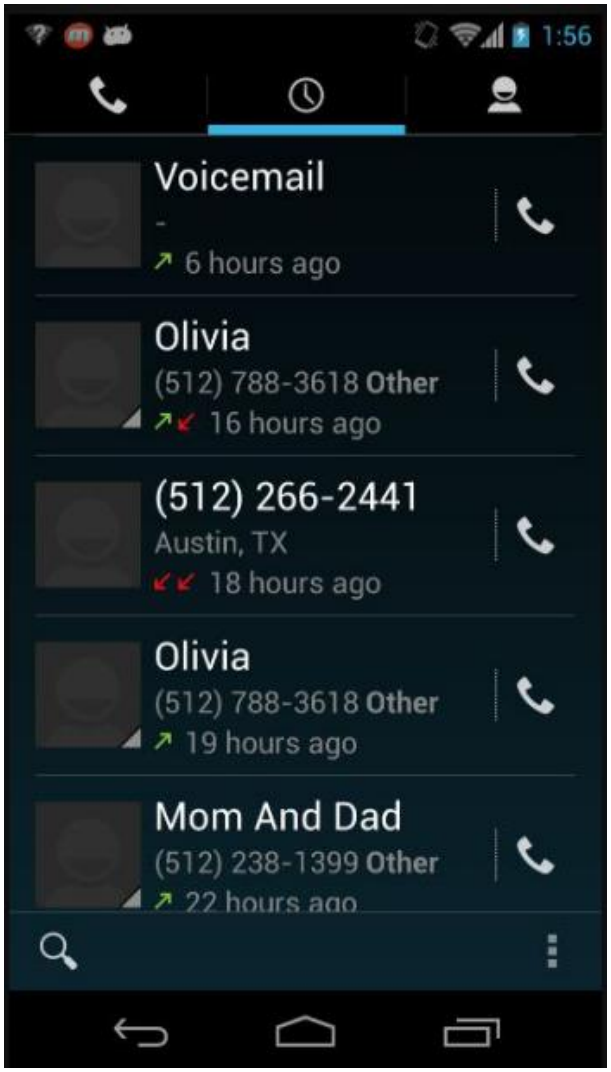
- the **Single Activity** app
  - focus on a Single Activity
  - calculator
  - camera



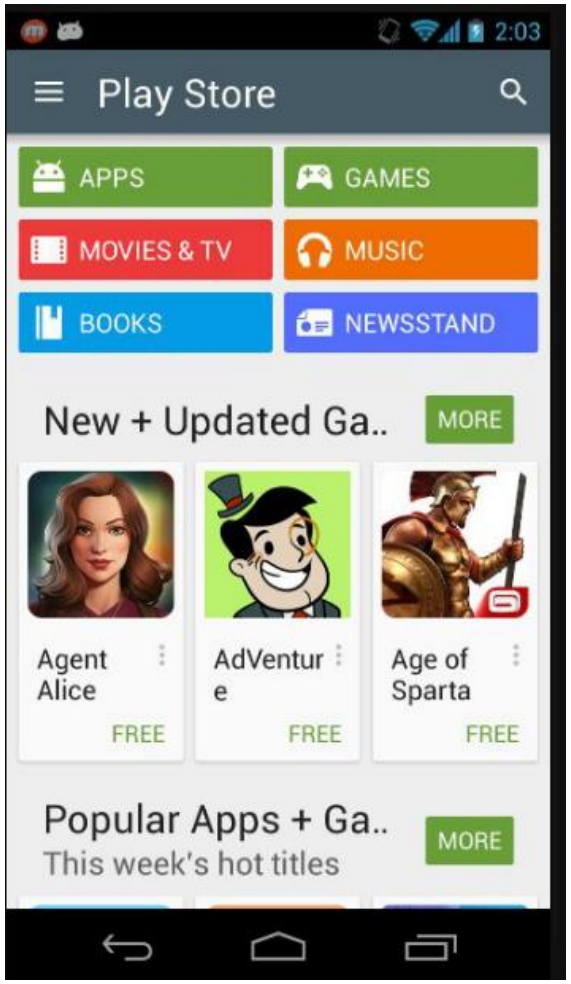
App Navigation Structures



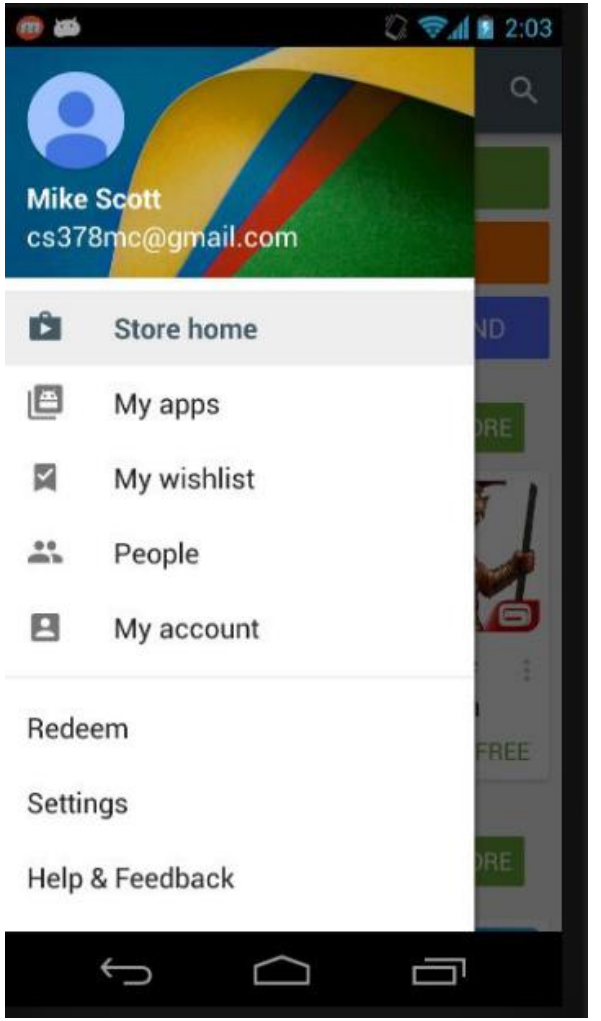
- the **Multiple Peer Activities** app
- multiple activities, but all on same level
- no deeper navigation
- phone app



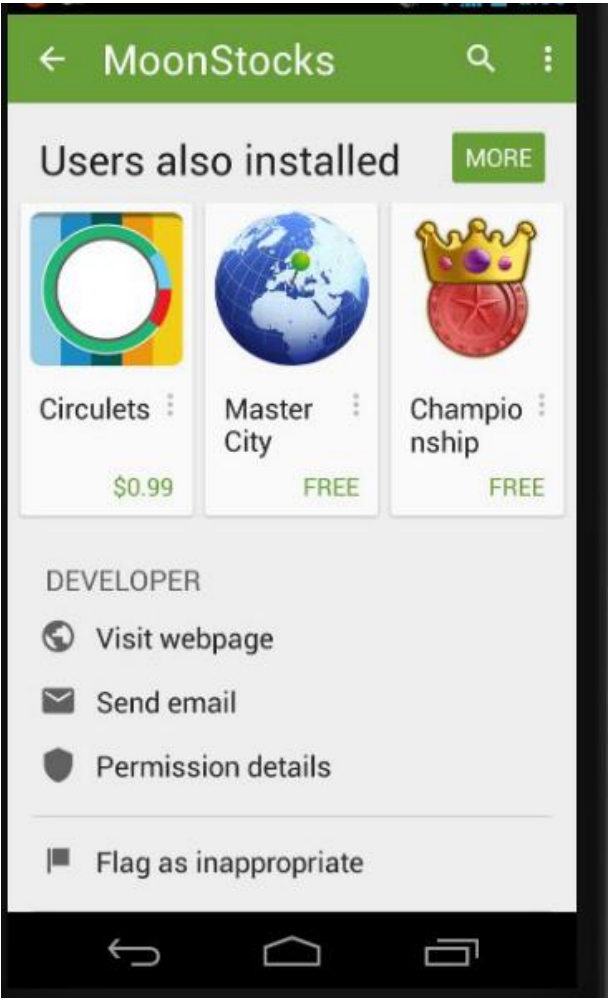
App Navigation Structures

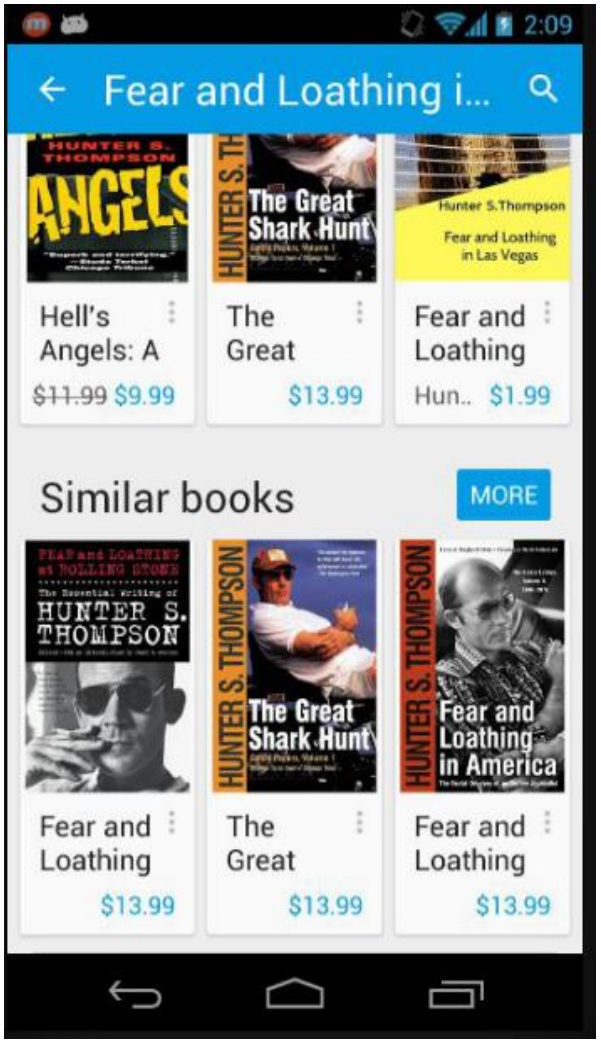


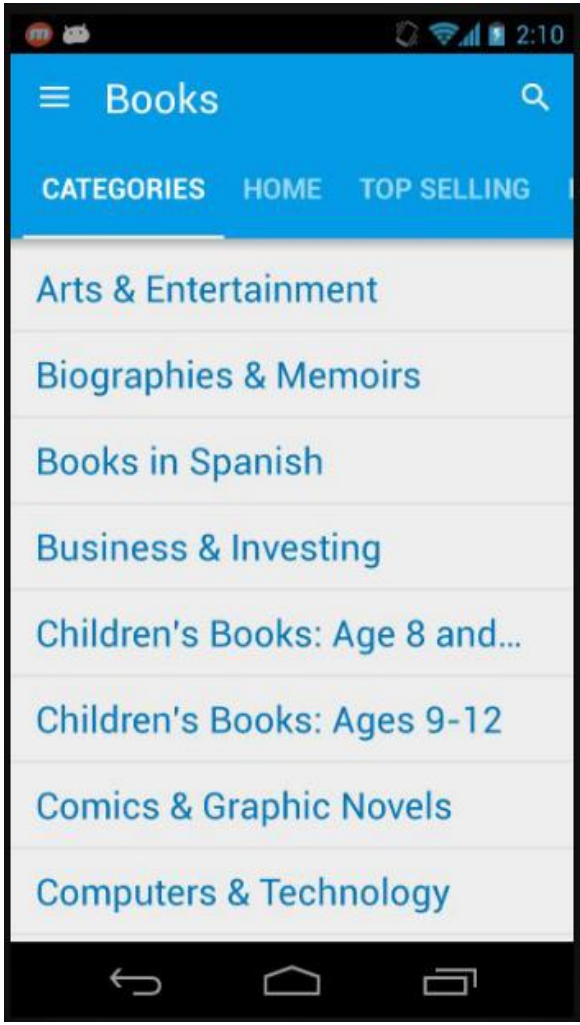
- the **Rabbit Hole** apps
- deep levels of navigation
- multiple data views
- Facebook, Play Store



Multiple Layers of Navigation







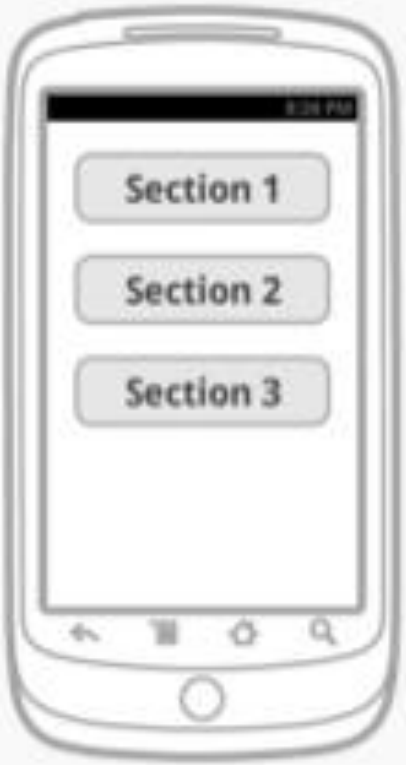
User Interface Patterns

- Just like software patterns, solutions to recurring UI design problems and situations
- Popular Android navigation patterns:
- Buttons and Simple Targets
- Lists and Grids
- Tabs
- Horizontal Paging
- The Navigation Drawer

NAVIGATION PATTERNS

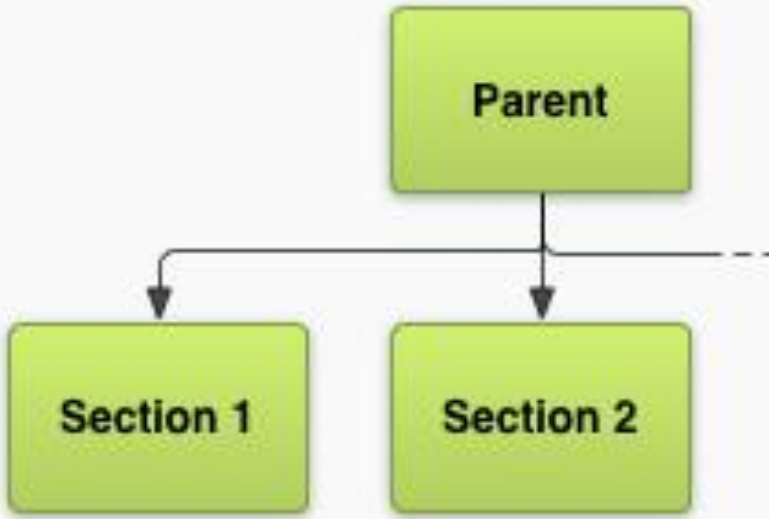
Buttons and Simple Targets

- Simple and familiar



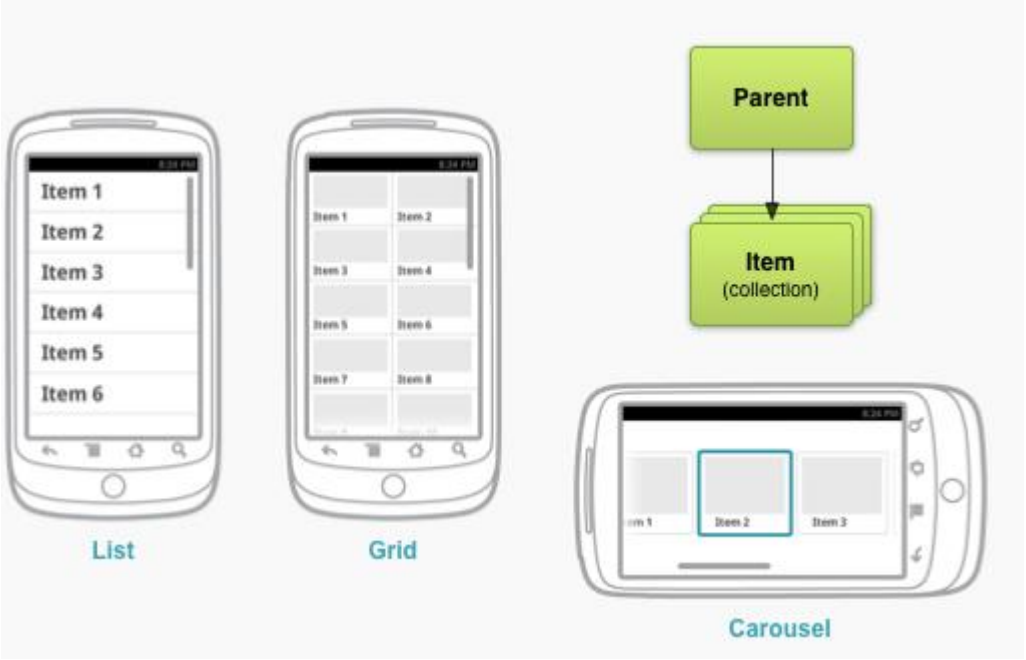
Simple Buttons

Dashboard



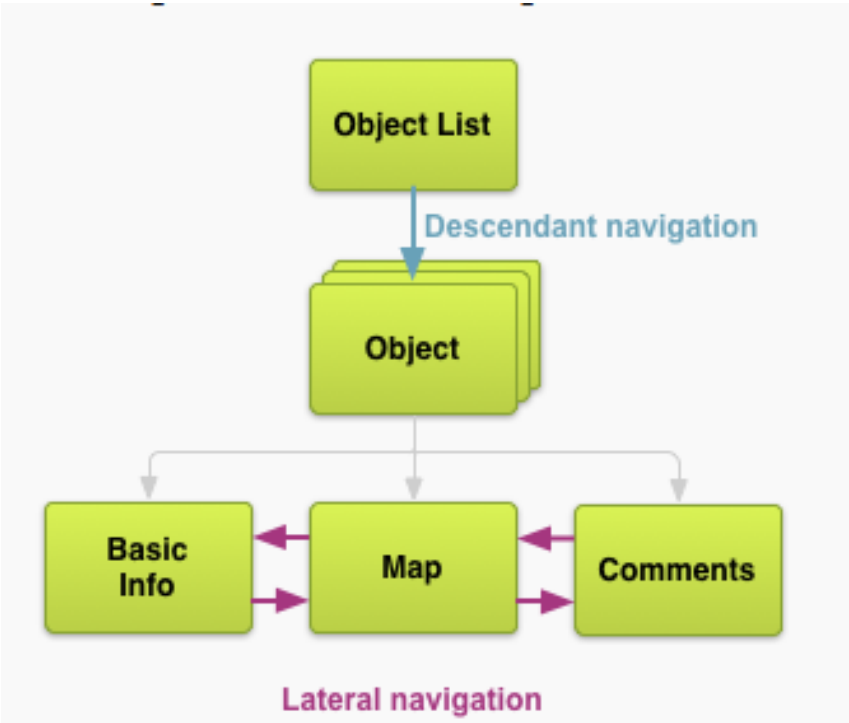
Lists and Grids

- For collection related screens, especially text based information
- ListView and GridView
- For photos and videos a scrolling list



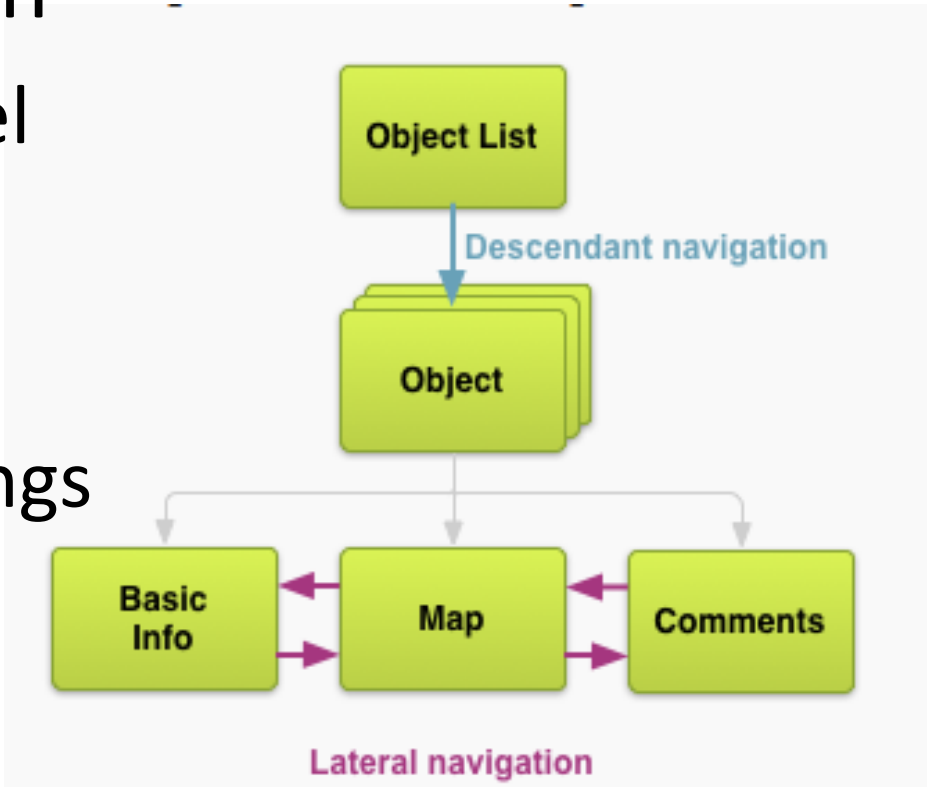
Tabbed Navigation

- Apps (should) have a navigation hierarchy
- Part of UI design is providing navigation between the different screens and activities
- developers need to think about the navigation so that users don't
- An alternative is Drawer Navigation



Navigation

- Descendant Navigation
  - moving from high level to low level
- Lateral navigation
  - moving between siblings
  - section siblings (in image)
  - content siblings
    - think image gallery



TABBED NAVIGATION

Tabs

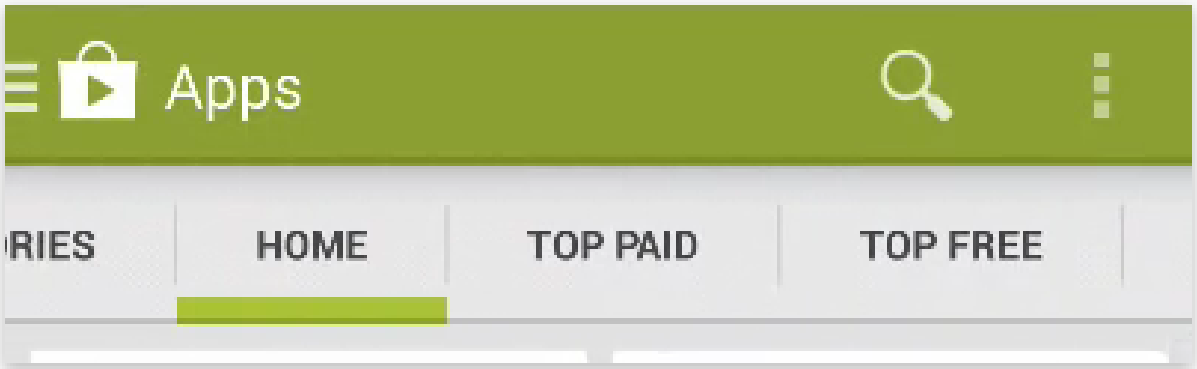
- Very popular
- used for sibling screens / activities
- Tabs should persist when changing screens
  - content changes to new screen, but tabs remain the same
- changing tabs should not create *history*
  - pressing back does should not cause a tab back
- tabs should always be at the top of the screen

Tabs vs. Buttons

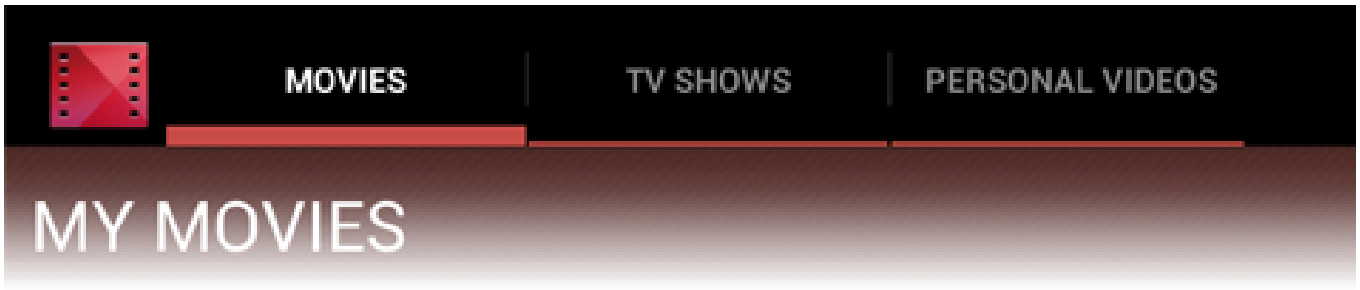
- initially selected tab in "parent" screen provides immediate access to content
- user navigation between screens without backtracking to parent
- … but, tabs take away space from the content screens

Tabs

- Tabs can be fixed or scrollable



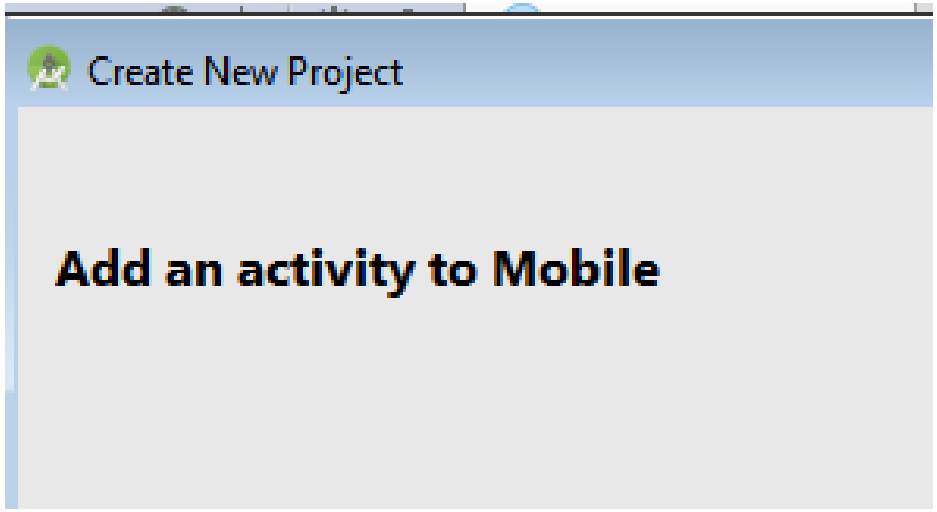
Scrolling tabs in the Play Store app.

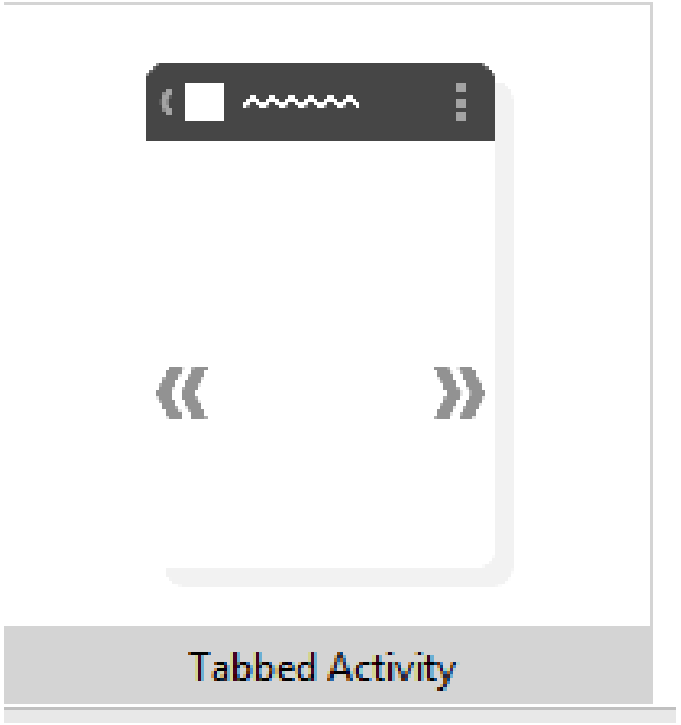


Tabs in the Google Play Movies app.

Implementing Tabs

- Android Studio project creation





Tabbed Activity

Creates a new blank activity, with an action bar and navigational elements such as tabs or horizontal swipe.

Implementing Tabs

- Swipe Views such as Tabs or Lateral Swipe Navigation use a ViewPager
- An descendant of ViewGroup
  - like LinearLayout, TableLayout, …
- Part of the *support library*
- A set of libraries to allow backward compatibility of apps
  - example, allow use of ActionBar on pre Android 3.0 devices

ViewPager in layout XML

```
<android.support.v4.view.ViewPager
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/pager"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

- add child views with a PageAdapter
  - recall the Adapter for the ListView
  - FragmentPagerAdapter for fixed # of siblings
  - FragmentStatePagerAdapter for a variable number of views, for example images

Rest of Layout File

- PagerTitleStrip Widget

```
<!--
This title strip will display the currently
visible page title, as well as the page
titles for adjacent pages.
-->
<android.support.v4.view.PagerTitleStrip
    android:id="@+id/pager_title_strip"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_gravity="top"
    android:background="#33b5e5"
    android:paddingBottom="4dp"
    android:paddingTop="4dp"
    android:textColor="#fff" />
```

Activity with Tabbed Navigation

```
public class MainActivity
        extends FragmentActivity
        implements ActionBar.TabListener {

    private AppSectionsPagerAdapter mAppSectionsPagerAdapter;

    private ViewPager mViewPager;

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
```

Setting Up The Navigation in onCreate()

```
// Specify that we will be displaying tabs in the action bar.
actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);

// Set up the ViewPager, attaching the adapter and setting up a listener for when th
// user swipes between sections.
mViewPager = (ViewPager) findViewById(R.id.pager);
mViewPager.setAdapter(mAppSectionsPagerAdapter);
mViewPager.setOnPageChangeListener(new ViewPager.SimpleOnPageChangeListener() {
    @Override
    public void onPageSelected(int position) {
        // When swiping between different app sections, select the corresponding tab
        // We can also use ActionBar.Tab#select() to do this if we have a reference
        // Tab.
        actionBar.setSelectedNavigationItem(position);
    }
});
```

Adding Tabs to ActionBar

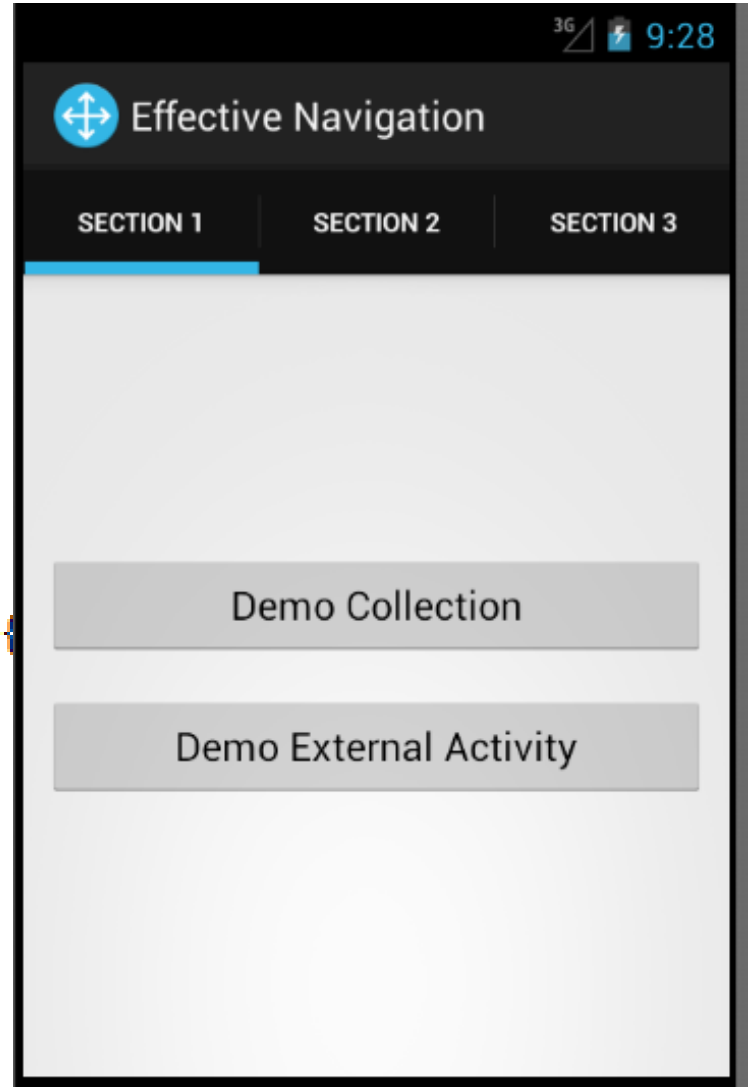
```
// For each of the sections in the app, add a tab to the action bar.
for (int i = 0; i < mAppSectionsPagerAdapter.getCount(); i++) {
    // Create a tab with text corresponding to the
    // page title defined by the adapter.
    // Also specify this Activity object, which
    // implements the TabListener interface, as the
    // listener for when this tab is selected.
    actionBar.addTab(
            actionBar.newTab()
                    .setText(mAppSectionsPagerAdapter.getPageTitle(i))
                    .setTabListener(this));
}
```

PagerAdapter

```
/**
 * A {@link FragmentPagerAdapter} that returns
 * a fragment corresponding to one of the primary
 * sections of the app.
 */
public static class AppSectionsPagerAdapter extends FragmentPagerAdapter {

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

    @Override
    public Fragment getItem(int i) {
        switch (i) {
            case 0:
                // The first section of the app is
                // the most interesting -- it offers
                // a launchpad into the other demonstrations
                // in this example application.
                return new LaunchpadSectionFragment();

            default:
                // The other sections of the app are dummy placeholders.
                Fragment fragment = new DummySectionFragment();
                Bundle args = new Bundle();
                args.putInt(DummySectionFragment.ARG_SECTION_NUMBER, i + 1);
                fragment.setArguments(args);
                return fragment;
        }
    }
```

PagerAdapter

```
@Override
public int getCount() {
    return 3;
}

@Override
public CharSequence getPageTitle(int position) {
    return "Section " + (position + 1);
}
```



Subviews are Fragments

```
/**
 * A fragment that launches other parts of the demo application.
 */
public static class LaunchpadSectionFragment extends Fragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        View rootView = inflater.inflate(R.layout.fragment_section_launchpad, container, false);

        // Demonstration of a collection-browsing activity.
        rootView.findViewById(R.id.demo_collection_button)
                .setOnClickListener(new View.OnClickListener() {
                    @Override
                    public void onClick(View view) {
                        Intent intent = new Intent(getActivity(), CollectionDemoActivity.class);
                        startActivity(intent);
                    }
                });
```

Clicker Question

- Have you used apps with a Navigation Drawer?

A. No

B. Yes

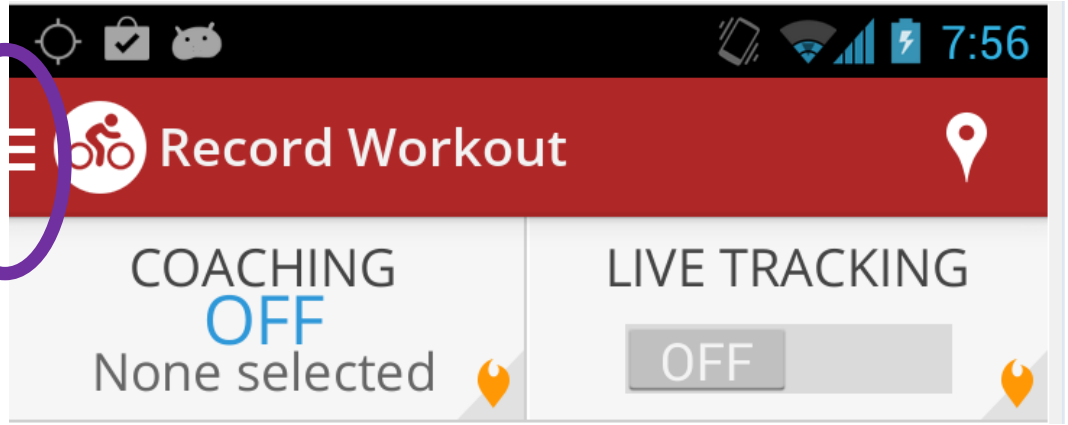
C. Maybe?

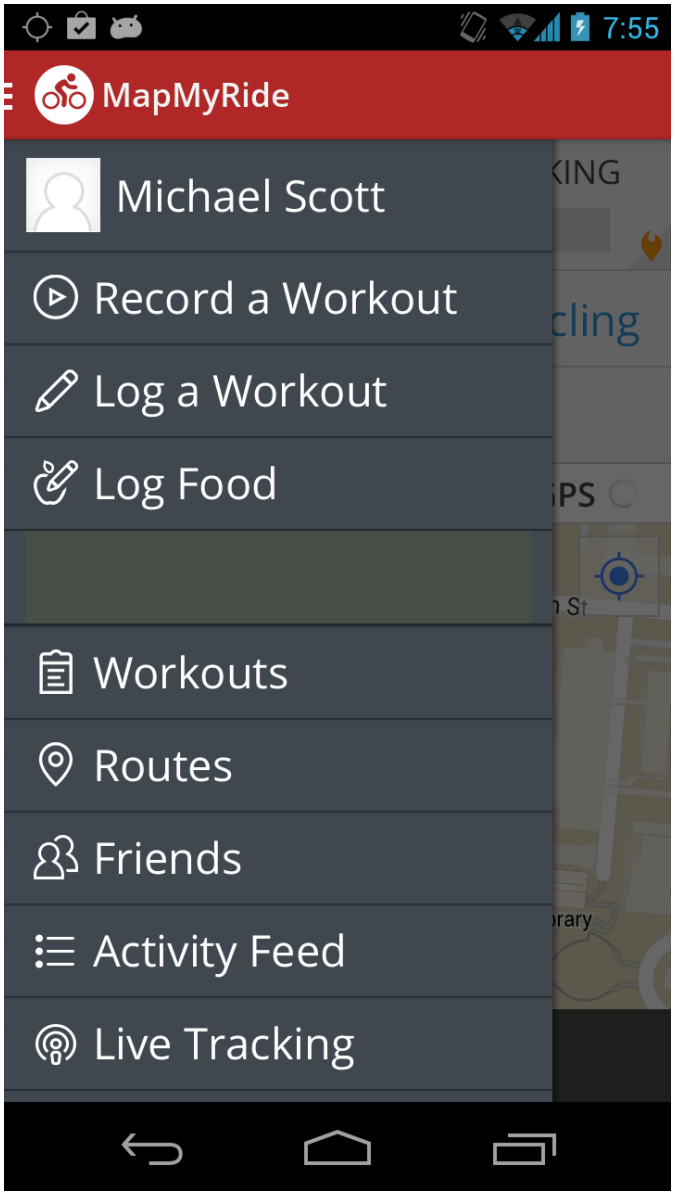
NAVIGATION DRAWER

Navigation Drawer

- A Drawer is an alternative for providing navigation through an app
  - especially between peer activities
- The drawer moves from the left edge of the screen when swiped in
  - or touch the app icon in the action bar
  - action bar altered when drawer displayed
- Drawer philosophy:
  - make the current view less cluttered
  - easier to move to important activities from anywhere within app

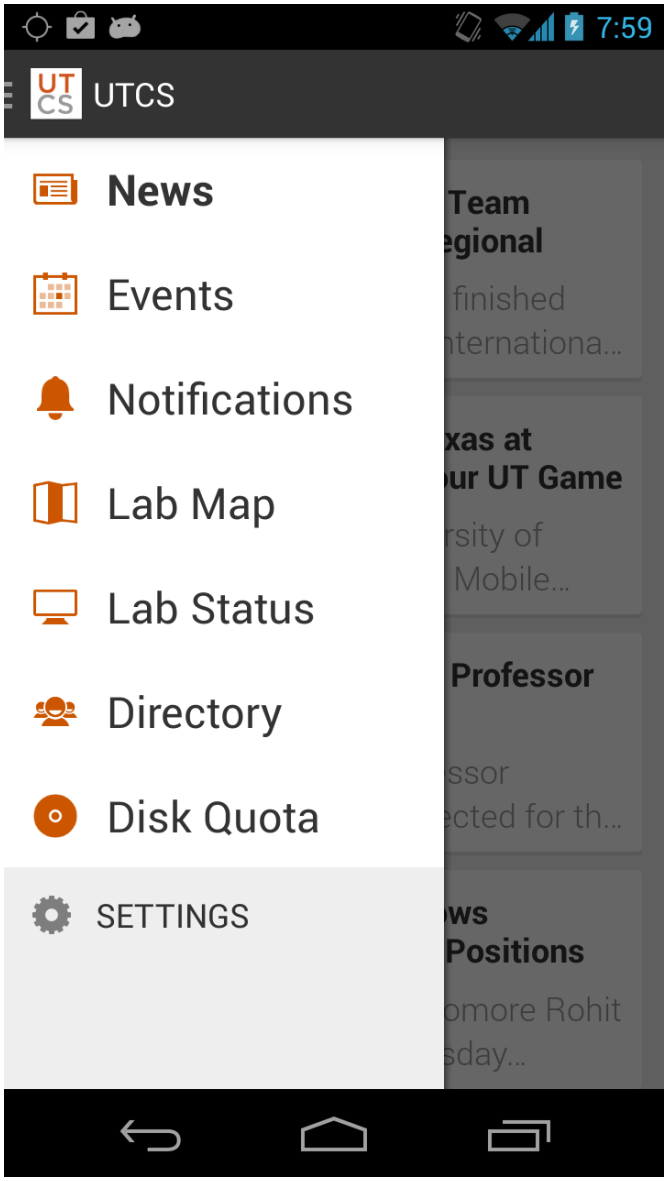
Example Navigation Drawers





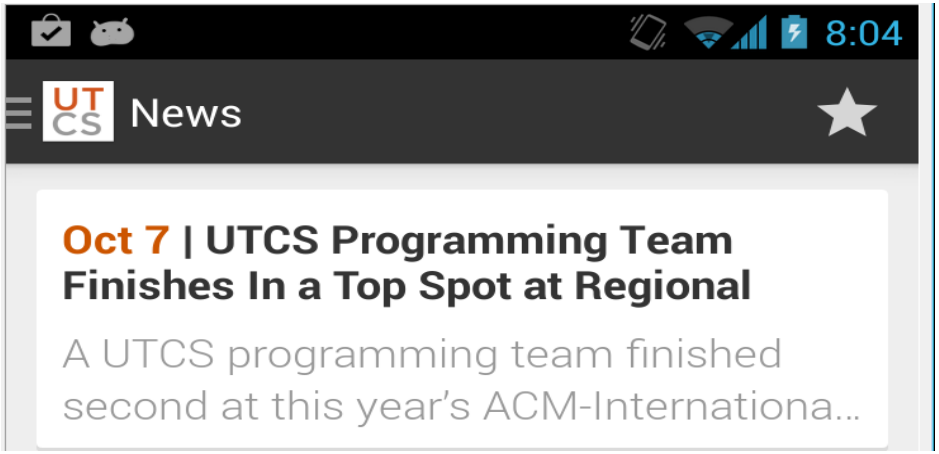
Example Navigation Drawers

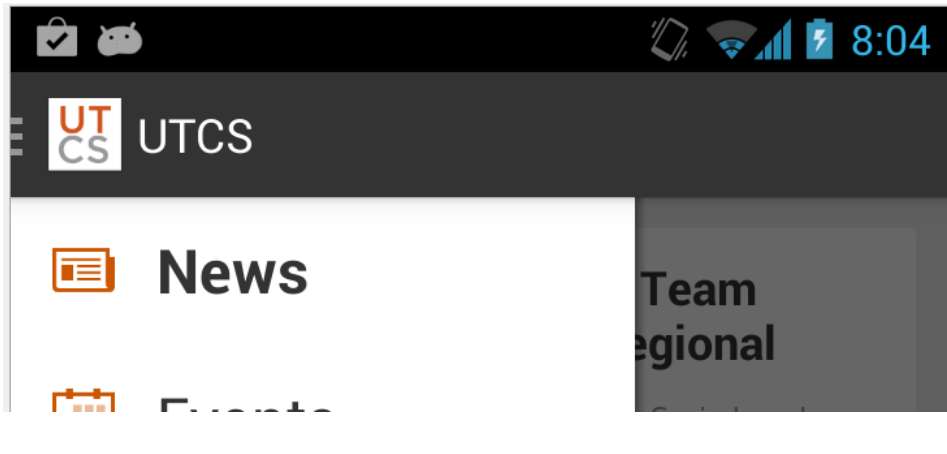
- The Drawer becomes the primary Navigation tool for the app
- Able to open from most Activities
- Different paradigm:
  - from a content view, back generally exits the app



Action Bar Changes

- Drawer overlays content, but not Action Bar
- Action Bar title should change from Activity Name to App name
- Hide any Action Bar items based on context of Activity



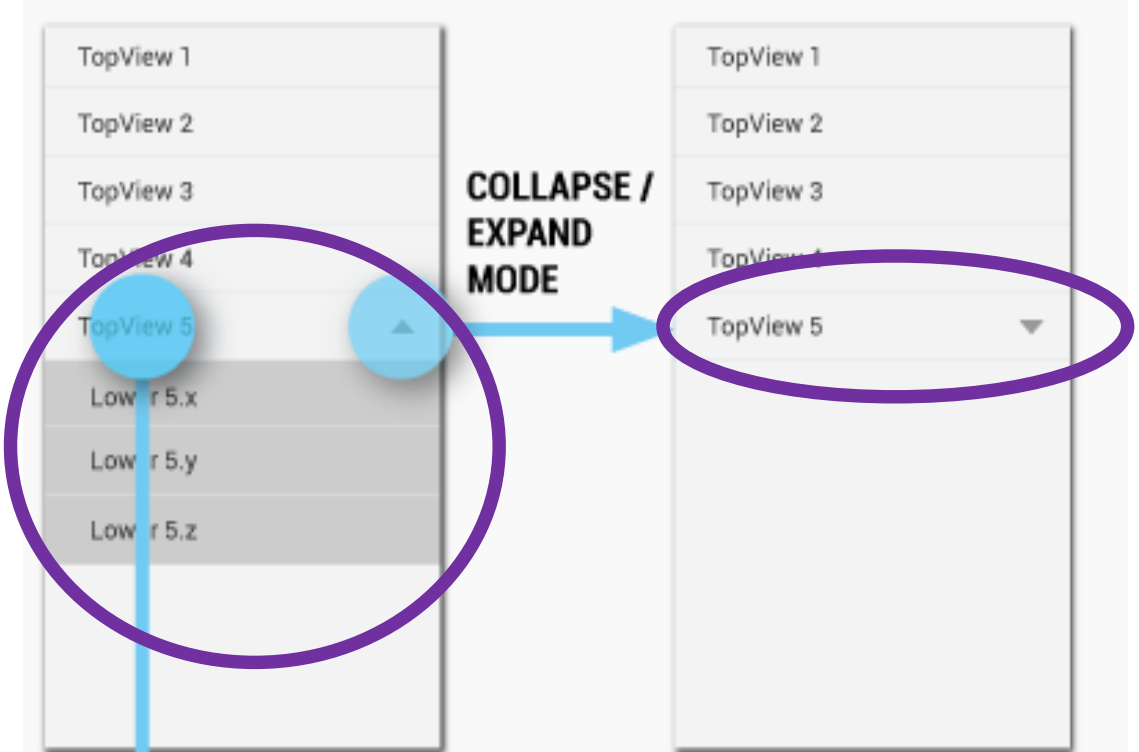


When to Use a Drawer

- Alternative top level navigation mechanism
  - not a replacement for tabs or spinners
- Navigation Drawers are a good option when:
  - many (>= 4) top level views
  - app requires lateral navigation between low level activities
  - deep navigation branches to ease pain of going back, back, back, back, …

Navigation Drawer Design

- Items in drawer broken up into rows
- Each row has a title and optional icon
- Possible to collapse multiple items into a single row



Navigation Bar Design Checklist

- The action bar remains in place and adjusts its content.
- Your navigation drawer overlays the content.
- Any view represented in the drawer has a navigation drawer indicator in its action bar that allows the drawer to be opened by touching the app icon.
- You take advantage of the new visual drawer transition.
- Any view not represented in the drawer maintains the traditional Up indicator in its action bar.
- You stay in sync with the general navigation patterns for Up and Back.

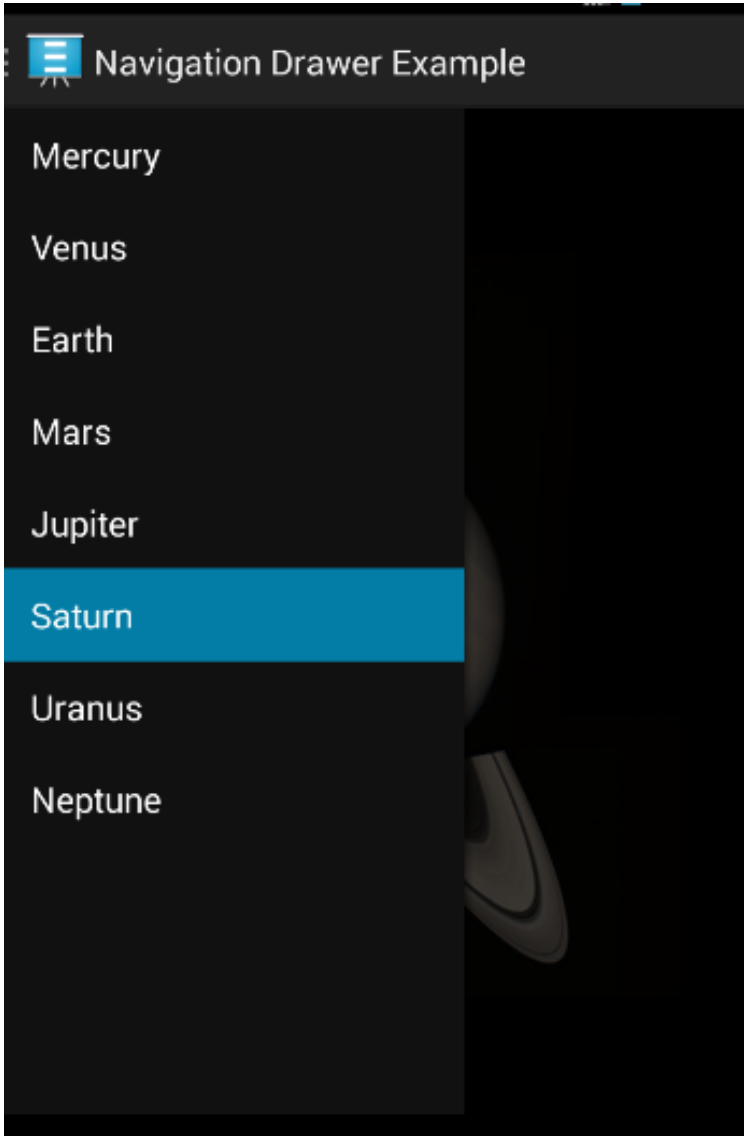
http://developer.android.com/design/patterns/navigation-drawer.html

Navigation Drawer Example

- Display Planets
- Image of planet from app
- ActionBar item to search web for planet
- Drawer to change planets

Drawer Open

- Note: Action Bar title change
- Note: removal of Action Item, search
- Note: drawer does not cover entire content view



Implementing a Navigation Drawer

- DrawerLayout APIs in the support library
- Create layout file with DrawerLayout as the root container
  - recall, LinearLayout, FrameLayout, …
- Inside Layout add two components
  - one for the regular content
  - and another for the Drawer content
  - likely a ListView, like the Countries app

DrawerLayout xml

```
<android.support.v4.widget.DrawerLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <!-- The main content view -->
    <FrameLayout
        android:id="@+id/content_frame"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
    <!-- The navigation drawer -->
    <ListView android:id="@+id/left_drawer"
        android:layout_width="240dp"
        android:layout_height="match_parent"
        android:layout_gravity="start"
        android:choiceMode="singleChoice"
        android:divider="@android:color/transparent"
        android:dividerHeight="0dp"
        android:background="#111"/>
</android.support.v4.widget.DrawerLayout>
```

DrawerLayout xml

- main content must be first
  - order in layout file sets z ordering, later items appear on top of earlier items
- main content matches parent width and height, entire UI when drawer hidden
- drawer view must specify layout gravity
  - "start", instead of "left" to support right to left languages
- height of drawer matches parent, width hard coded and should be no more than 320 dp so some portion of main content still visible

Populating Drawer

- Container for drawer is a ListView in example
  - typical, although other layouts allowed
- Recall, populate a ListView with an adapter
  - ArrayAdapter or SimpleCursorAdapter (for reading from a data base)
- Example with planets creates ArrayAdapter attached to String array from a resource file

String Array Resource File

```
<resources>
    <string name="app_name">Navigation Drawer Example</stri
    <string-array name="planets_array">
        <item>Mercury</item>
        <item>Venus</item>
        <item>Earth</item>
        <item>Mars</item>
        <item>Jupiter</item>
        <item>Saturn</item>
        <item>Uranus</item>
        <item>Neptune</item>
    </string-array>
    <string name="drawer_open">Open navigation drawer</stri
    <string name="drawer_close">Close navigation drawer</st
    <string name="action_websearch">Web search</string>
    <string name="app_not_available">Sorry, there\'s no web
</resources>
```

Populating Drawer in onCreate()

```
public class MainActivity extends Activity {
    private String[] mPlanetTitles;
    private DrawerLayout mDrawerLayout;
    private ListView mDrawerList;
    ...

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        mPlanetTitles = getResources().getStringArray(R.array.planets_array);
        mDrawerLayout = (DrawerLayout) findViewById(R.id.drawer_layout);
        mDrawerList = (ListView) findViewById(R.id.left_drawer);

        // Set the adapter for the list view
        mDrawerList.setAdapter(new ArrayAdapter<String>(this,
                R.layout.drawer_list_item, mPlanetTitles));
        // Set the list's click listener
        mDrawerList.setOnItemClickListener(new DrawerItemClickListener());
```

DrawerItemClickListener and selectItem()

```
/* The click listener for ListView in the navigation drawer */
private class DrawerItemClickListener implements ListView.OnItemClickListener {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long i
        selectItem(position);
    }
}
```

```
private void selectItem(int position) {
    // update the main content by replacing fragments
    Fragment fragment = new PlanetFragment();
    Bundle args = new Bundle();
    args.putInt(PlanetFragment.ARG_PLANET_NUMBER, position);
    fragment.setArguments(args);

    FragmentManager fragmentManager = getFragmentManager();
    fragmentManager.beginTransaction().replace(R.id.content_frame, fragment).commit();

    // update selected item and title, then close the drawer
    mDrawerList.setItemChecked(position, true);
    setTitle(mPlanetTitles[position]);
    mDrawerLayout.closeDrawer(mDrawerList);
}
```

drawer closing with animation

Responding to Click

- in example
  selecting a drawer
  item replaces the
  content in the DrawerLayout with a new *fragment*

```
@Override
public void setTitle(CharSequence title) {
    mTitle = title;
    getActionBar().setTitle(mTitle);
}
```

Opening and Closing

- YAL!, yet another listener
- call setDrawerListener() on DrawerLayout and pass an implementation of DrawerLayout.DrawerListener
- Methods such as
  - onDrawerOpened()
  - onDrawerClosed()

open / close Alternative

- If app has an ActionBar:
- extend ActionBarDrawerToggle class
- implements the DrawerListener class
- still have to override methods for drawerOpen and drawerClose
- … but, this class helps handle the interaction between drawer and action bar (title, action items)

More from onCreate()

```
// ActionBarDrawerToggle ties together the the proper interactions
// between the sliding drawer and the action bar app icon
mDrawerToggle = new ActionBarDrawerToggle(
        this,                  /* host Activity */
        mDrawerLayout,         /* DrawerLayout object */
        R.drawable.ic_drawer,  /* nav drawer image to replace 'Up' caret */
        R.string.drawer_open,  /* "open drawer" description for accessibility *
        R.string.drawer_close  /* "close drawer" description for accessibility
        ) {
    public void onDrawerClosed(View view) {
        getActionBar().setTitle(mTitle);
        invalidateOptionsMenu(); // creates call to onPrepareOptionsMenu()
    }

    public void onDrawerOpened(View drawerView) {
        getActionBar().setTitle(mDrawerTitle);
        invalidateOptionsMenu(); // creates call to onPrepareOptionsMenu()
    }
};
mDrawerLayout.setDrawerListener(mDrawerToggle);
```

Changing Action Bar Items

- In this instance only one action bar item, search web for planet name
- hide if drawer is open

```
/* Called whenever we call invalidateOptionsMenu() */
@Override
public boolean onPrepareOptionsMenu(Menu menu) {
    // If the nav drawer is open, hide action items related to the content view
    boolean drawerOpen = mDrawerLayout.isDrawerOpen(mDrawerList);
    menu.findItem(R.id.action_websearch).setVisible(!drawerOpen);
    return super.onPrepareOptionsMenu(menu);
}
```

Action Bar interaction

- If app has an Action Bar should:
  - allow user to open and close drawer by tapping the app icon
  - have an icon indicating the app has a drawer

```
// enable ActionBar app icon to behave as action to toggle nav drawer
getActionBar().setDisplayHomeAsUpEnabled(true);
getActionBar().setHomeButtonEnabled(true);

// ActionBarDrawerToggle ties together the the proper interactions
// between the sliding drawer and the action bar app icon
mDrawerToggle = new ActionBarDrawerToggle(
        this,                  /* host Activity */
        mDrawerLayout,         /* DrawerLayout object */
        R.drawable.ic_drawer,  /* nav drawer image to replace 'Up' ca
        R.string.drawer_open,  /* "open drawer" description for acces
        R.string.drawer_close  /* "close drawer" description for acce
```

ActionBarToggle and Lifecycle

```
@Override
protected void onPostCreate(Bundle savedInstanceState) {
    super.onPostCreate(savedInstanceState);
    // Sync the toggle state after onRestoreInstanceState has occurred.
    mDrawerToggle.syncState();
}

@Override
public void onConfigurationChanged(Configuration newConfig) {
    super.onConfigurationChanged(newConfig);
    mDrawerToggle.onConfigurationChanged(newConfig);
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Pass the event to ActionBarDrawerToggle, if it returns
    // true, then it has handled the app icon touch event
    if (mDrawerToggle.onOptionsItemSelected(item)) {
      return true;
    }
    // Handle your other action bar items...
```

Multiple Drawers

- Possible to have another drawer
- left / start drawer for app navigation
- right / end drawer for options with the current content view
- General Android design:
  Navigation on the LEFT
  Actions on the RIGHT
- http://tinyurl.com/lnb2jb3

DIALOGS

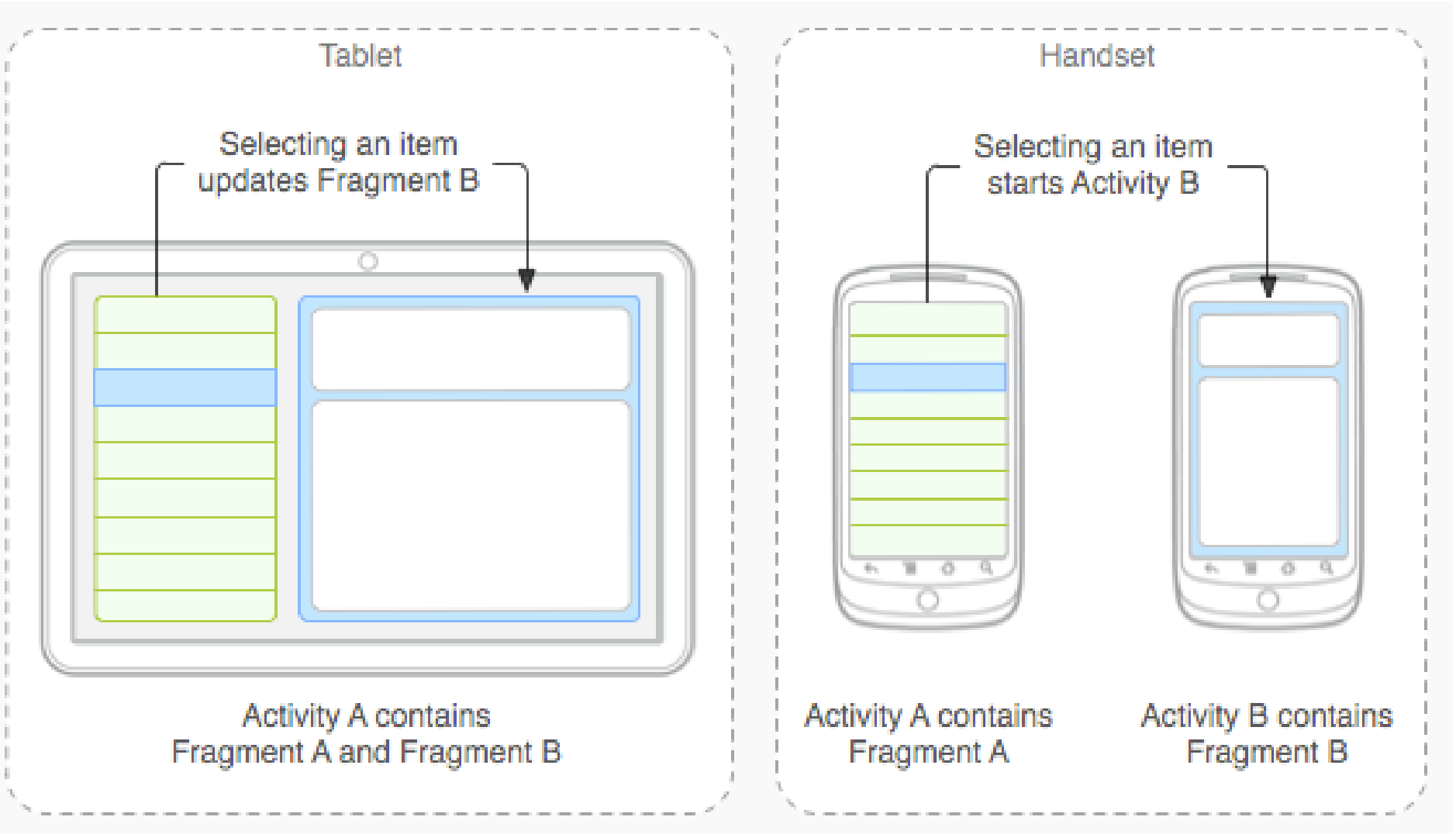
Dialogs - Old Way

- Dialogs from tutorials were cut and paste
- Implementing Dialogs demonstrates evolution of Android SDK
- legacy approach has Activity manage its own Dialogs
- created, initialized, updated, and destroyed using Activity class call back methods

Dialogs - New Way

- Android evolving from smartphone OS to smart device OS
- API level 11 (Android 3.0, the tablet release) introduced *Fragments*
- A fragment represents a behavior or a portion of a UI in an Activity
  - like a sub activity
- multiple fragments combined in multi-pane UI
- reuse fragments in multiple activities

Fragments


Tablet
Selecting an item updates Fragment B
Activity A contains Fragment A and Fragment B
Handset
Selecting an item starts Activity B
Activity A contains Fragment A
Activity B contains Fragment B

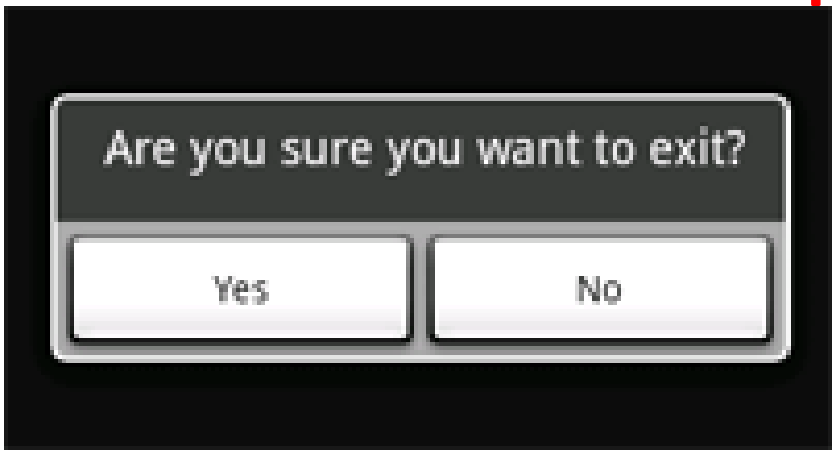
Dialogs as Fragments

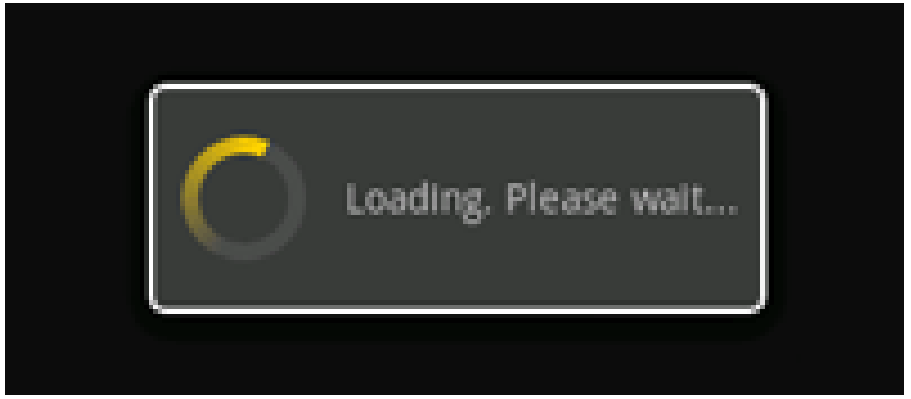
- Dialogs are special type of Fragment
- managed by the FragmentManager class
- still part of an activity, but lifecycle not managed by the Activity
  - life cycle issues of Dialogs as Fragments will be more difficult to deal with
  - must save state and restore instance

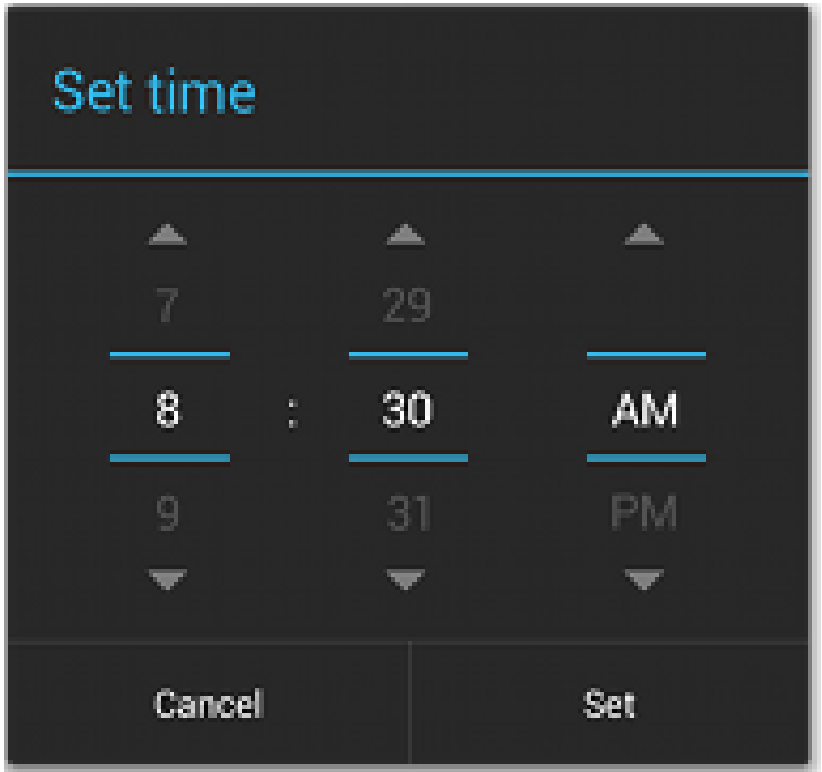
Types of Dialogs

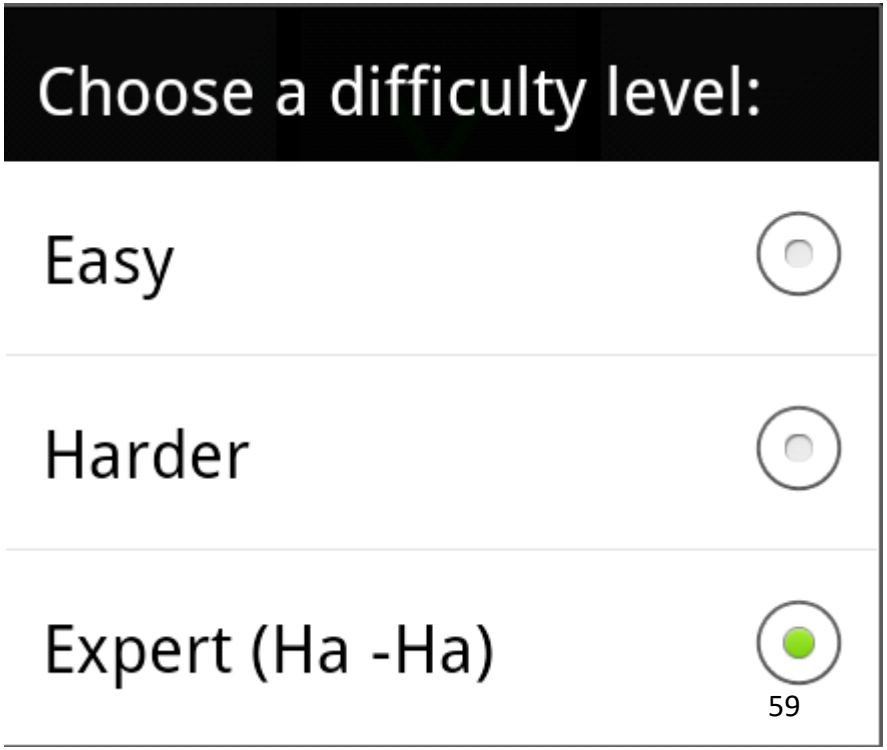
- Used to organize information and react to user events without creating a whole new activity
- Old Dialogs:
  - Dialog, AlertDialog, DatePickerDialog, TimePickerDialog, ProgressDialog
- New Dialogs:
  - DialogFragment

Sample Dialogs









Legacy Approach

- Dialog defined in Activity it is used
- Activity maintains a pool of Dialogs
- showDialog() method displays Dialog
- dismissDialog() method used to stop showing a Dialog
  - in tutorial, when we have difficulty
- removeDialog removes from pool

Legacy Approach - Steps

- Define unique indentifier for the Dialog in Activity (constants)

```
static final int DIALOG_DIFFICULTY_ID = 0;
static final int DIALOG_QUIT_ID = 1;
static final int DIALOG_ABOUT_ID = 2;
static final int DIALOG_CLEAR_SCORES = 3;
```

- implement onCreateDialog method, returns Dialog of appropriate type

onCreateDialog

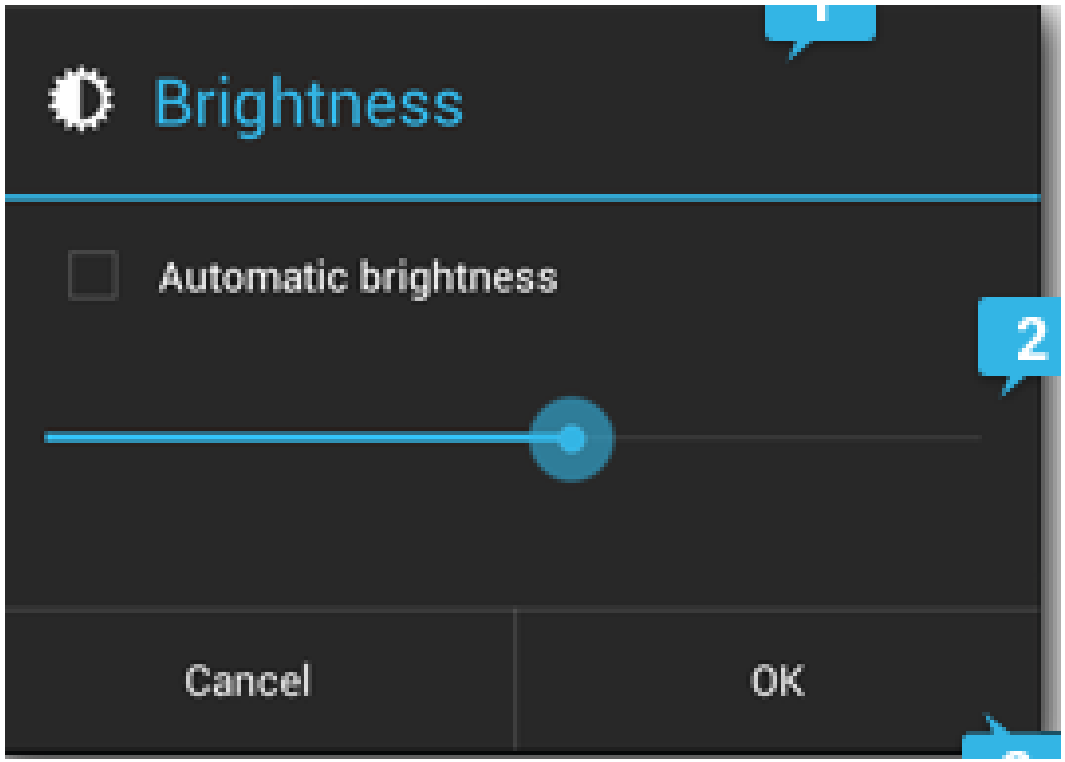
```
@Override
protected Dialog onCreateDialog(int id) {
    Dialog dialog = null;
    AlertDialog.Builder builder = new AlertDialog.Builder(this);

    switch(id) {
        case DIALOG_DIFFICULTY_ID:
            dialog = createDifficultyDialog(builder);
            break;    // this case
        case DIALOG_QUIT_ID:
            dialog = this.createQuitDialog(builder);
            break;
        case DIALOG_ABOUT_ID:
            dialog = createAboutDialog(builder);
            break;
        case DIALOG_CLEAR_SCORES:
            dialog = createClearScoresDialog(builder);
            break;
    }

    if(dialog == null)
        Log.d(TAG, "Uh oh! Dialog is null");
    else
        Log.d(TAG, "Dialog created: " + id + ", dialog: " + dialog);
    return dialog;
}
```

Dialog Steps - Legacy Approach

- implement onPrepareDialog() if necessary
  - if necessary to update dialog each time it is displayed
  - for example, a time picker, update with the current time
- launch dialog with showDialog()
  - in tutorials done when a menu or action bar menu item selected
  - could launch Dialogs for other reasons

Alert Dialogs

- Most common type
- Title, Content Area, Action buttons (up to 3)
- Content area could be message, list, seekbar, etc.
- set positive, set negative, set neutral



Custom Dialogs

- AlertDialog very flexible, but you can create CustomDialogs
- Create a layout

```
<?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/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:textColor="#FF00FF"
        android:textSize="30sp" />

</LinearLayout>
```

Custom Dialogs

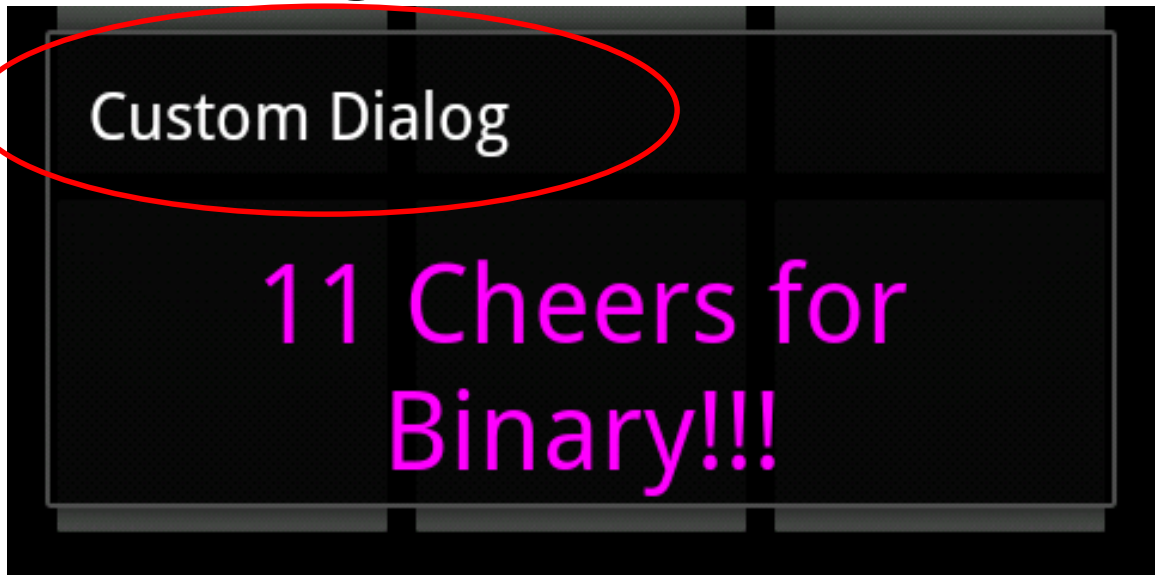
- from onCreateDialog

```
case DIALOG_CHEER_ID:
    Log.d(TAG, "CREATING CUSTOM DIALOG");
    dialog = new Dialog(this);

    dialog.setContentView(R.layout.cheer);
    dialog.setTitle("Custom Dialog");

    TextView text = (TextView) dialog.findViewById(R.id.text);
    text.setText("11 Cheers for Binary!!!");
```

Custom Dialog

- Simple dialogs are dismissed with the back button

dialog title



Dialogs - Fragment Method

- Decouple Dialogs from the Activity
  - good SE approach?
  - TicTacToe UI is almost 500 lines long!
- Implement a class that is a subclass of DialogFragment
  - DifficultyFragment
  - Send info to newInstance method (current difficulty, listener for updates)
  - onCreateDialog now in DifficultyFragment

DifficultyFragment

```
public class DifficultyFragment extends DialogFragment {


    public interface DifficultyListener {
        public void difficlutySelected(int diff, String name);
    }

    private DifficultyListener mListener;

    public static DifficultyFragment newInstance(int currentDiffulty,
                DifficultyListener mListener) {

        DifficultyFragment newInstance = new DifficultyFragment();
        Bundle args = new Bundle();
        args.putInt("diff", currentDiffulty);
        newInstance.setArguments(args);
        newInstance.mListener = mListener;
        return newInstance;
    }
```

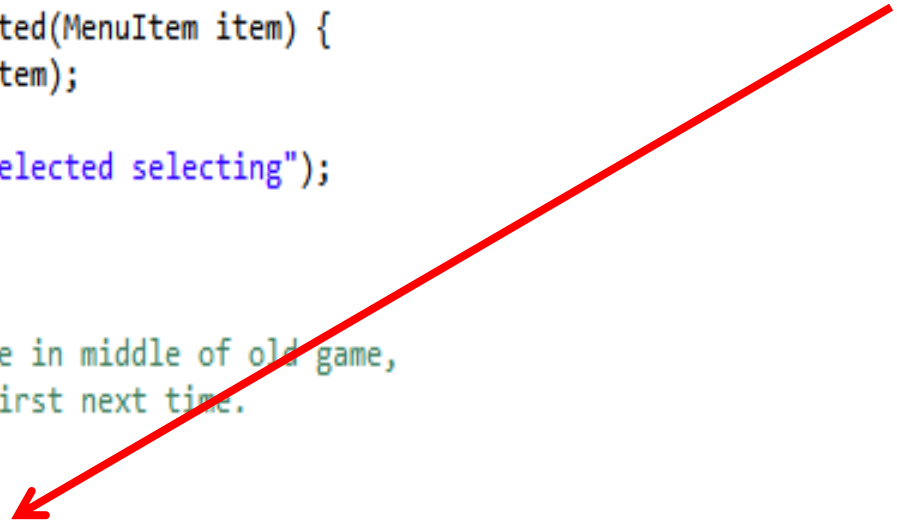
DifficultyFragment - onCreateDialog

```
@Override
public Dialog onCreateDialog(Bundle saveInstanceState) {
    int currentDifficulty = getArguments().getInt("diff");

    AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());

    final CharSequence[] levels = {
            getResources().getString(R.string.difficulty_easy),
            getResources().getString(R.string.difficulty_harder),
            getResources().getString(R.string.difficulty_expert)};

    builder.setTitle(R.string.difficulty_choose);
    builder.setIcon(R.drawable.difficulty_level);
    builder.setSingleChoiceItems(levels, currentDifficulty,
            new DialogInterface.OnClickListener() {

        public void onClick(DialogInterface dialog, int item) {
            dialog.dismiss();   // Close dialog

            mListener.difficlutySelected(item, levels[item].toString());
        }
    });
    return builder.create();
}
```

Using DifficultyFragment

- In AndroidTicTacToe create a listener to pass to the newInstance method
- create and show Dialog as part of onOptionsItemSelected()

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    super.onOptionsItemSelected(item);

    Log.d(TAG, "in onOptionsItemSelected selecting");
    switch (item.getItemId()) {
    case R.id.new_game:
        stopComputerDelay();
        // if user starts new game in middle of old game,
        // they don't get to go first next time.
        startNewGame(false);
        return true;
    case R.id.ai_difficulty:
        DifficultyFragment df = DifficultyFragment.newInstance(mGame.getDifficultyLevel().ordinal(), diffListener
        df.show(getFragmentManager(), "difficultyFragment");
        return true;
```

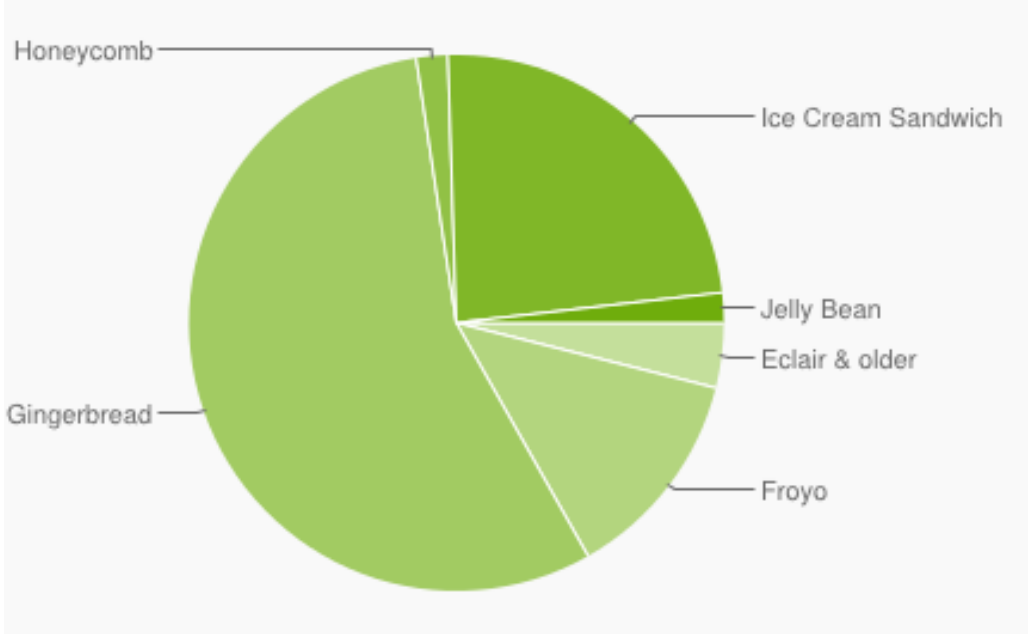


DifficultyListener

```
private DifficultyFragment.DifficultyListener diffListener
        = new DifficultyFragment.DifficultyListener() {

    @Override
    public void difficlutySelected(int diffLevel, String diff) {
        mGame.setDifficultyLevel(TicTacToeGame.DifficultyLevel.values()[diffLevel]);
        Log.d(TAG, "Difficulty level: " + mGame.getDifficultyLevel());

        // Display the selected difficulty level
        Toast.makeText(getApplicationContext(), diff,
                Toast.LENGTH_LONG).show();
    }

};
```

Using Fragments

- Fragments added in API level 11, Android 3.0, the tablet release
- Developers behind Android think fragments are so important that can be used in pre API 11 builds using the *Android Support Library*



Froyo and Gingerbread pre API 11

Android Support Library (ASL)

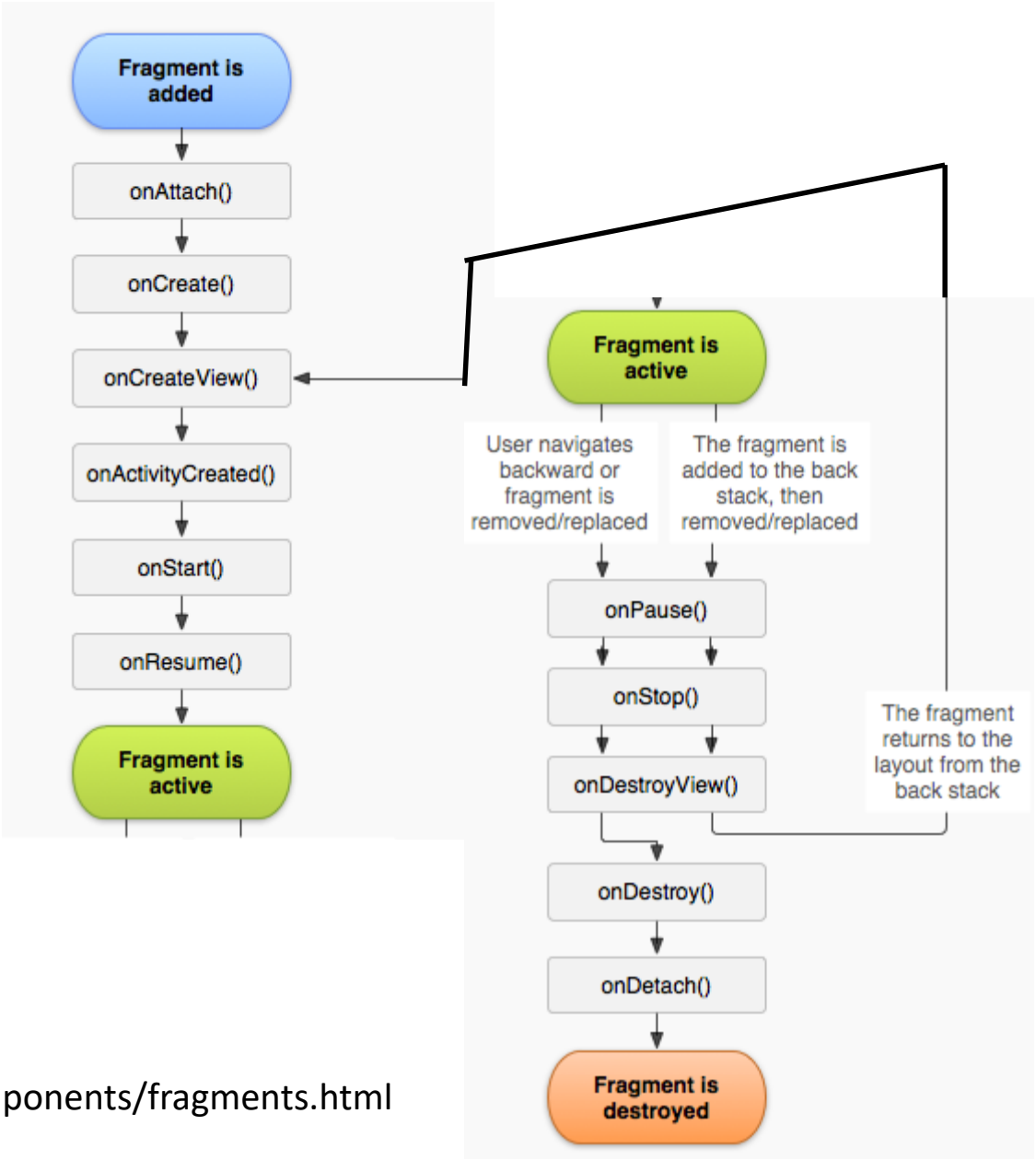
- add library to project and application
- android.support.v4.app.DialogFragment
  - for example
  - instead of android.app.DialogFragment
- ASL does not support action bar in earlier versions of API
  - discover ActionBarSherlock

- Fragment
- FragmentManager
- FragmentTransaction
- ListFragment
- DialogFragment
- LoaderManager
- Loader
- AsyncTaskLoader

Fragment Lifecycle

- Common error:

not dealing with orientation change when Dialog is open



http://developer.android.com/guide/components/fragments.html

THEMES

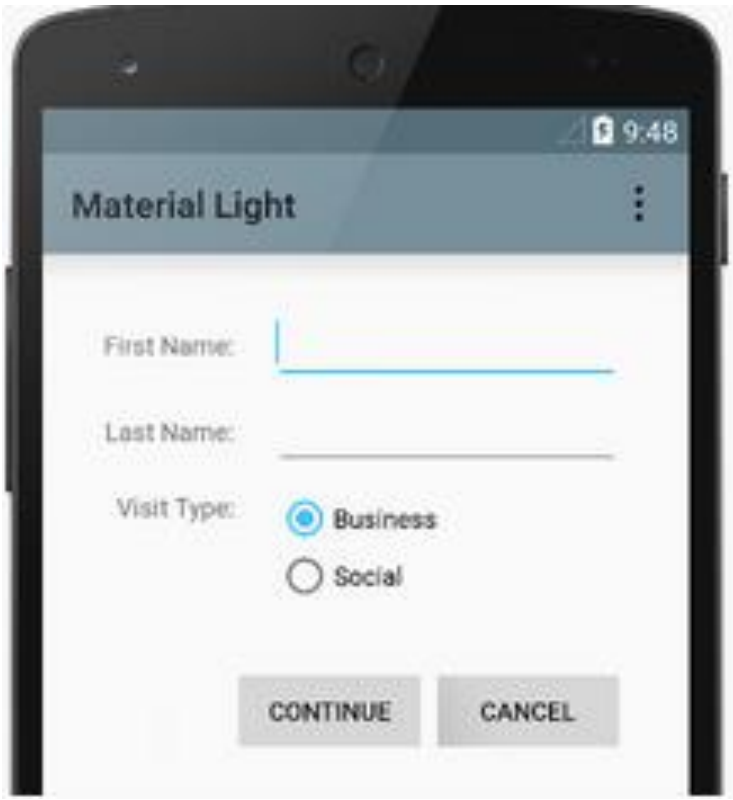
Consistency

- Themes are Android's mechanism for a consistent *style* in an app or activity
- Theme is a predefined style
- sets properties of layouts and widgets such as
  - color
  - height
  - padding
  - font size.

HOLO DARK THEME

New Themes

- Holo light and dark were the Honeycomb (3.0) themes
- Lollipop (5.0) added the *Material Design* theme
- System Widgets that allow you to pick color palette (customize)
- Touch feedback animations for system Widgets
- Activity transition animations



LIGHT MATERIAL THEME

Setting a Theme

- App theme set in the Manifest

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.android.navigationdrawerexample"
    android:versionCode="1"
    android:versionName="1.0">

    <uses-sdk
        android:minSdkVersion="14"
        android:targetSdkVersion="17" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@android:style/Theme.Holo.Light.DarkActionBar"
```

Using Built in Styles

- R.style class
  - not to be confused with our R class

public static final class

R.style

extends Object

java.lang.Object
↳android.R.style

Summary

Constants	
int	Animation
int	Animation_Activity
int	Animation_Dialog

Constants		
int	Animation	Base style for animations.
int	Animation_Activity	Standard animations for a full-screen window or activity.
int	Animation_Dialog	Standard animations for a non-full-screen window or activity.
int	Animation_InputMethod	Window animations that are applied to input method overlay windows.
int	Animation_Toast	
int	Animation_Translucent	Standard animations for a translucent window or activity.
int	DeviceDefault_ButtonBar	Other Styles
int	DeviceDefault_ButtonBar_AlertDialog	
int	DeviceDefault_Light_ButtonBar	
int	DeviceDefault_Light_ButtonBar_AlertDialog	
int	DeviceDefault_Light_SegmentedButton	
int	DeviceDefault_SegmentedButton	
int	Holo_ButtonBar	
int	Holo_ButtonBar_AlertDialog	
int	Holo_Light_ButtonBar	
int	Holo_Light_ButtonBar_AlertDialog	
int	Holo_Light_SegmentedButton	
int	Holo_SegmentedButton	
int	MediaButton	

R.style

- Our widgets (buttons, seek bars, edit texts, etc.) are using the android R.style
- We are overriding some attributes
- Also for Views:
  - "@android:style/Theme.NoTitleBar"

R.style

- Not well documented
- Suggestion is too look at the actual xml
- Styles at http://tinyurl.com/nz3j3ak
- Themes at http://tinyurl.com/ls9cxbf

Example Android Style

```
<style name="Widget.SeekBar">
    <item name="android:indeterminateOnly">false</item>
    <item name="android:progressDrawable">@android:drawable/progres
    <item name="android:indeterminateDrawable">@android:drawable/pr
    <item name="android:minHeight">20dip</item>
    <item name="android:maxHeight">20dip</item>
    <item name="android:thumb">@android:drawable/seek_thumb</item>
    <item name="android:thumbOffset">8dip</item>
    <item name="android:focusable">true</item>
    <item name="android:mirrorForRtl">true</item>
</style>
```

Example Android Theme

```
<resources>
    <!-- The default theme for apps on API level 10 and lower. This is the
         activities that have not explicitly set their own theme.
         <p>You can count on this being a dark
         background with light text on top, but should try to make no
         other assumptions about its appearance. In particular, the text
         inside of widgets using this theme may be completely different,
         with the widget container being a light color and the text on top
         of it a dark color.
         <p>If you're developing for API level 11 and higher, you should in
         #Theme_Holo} or {@link #Theme_DeviceDefault}.</p>
    -->
    <style name="Theme">
```

More of the Theme

```
<!-- Text styles -->
<item name="textAppearance">@android:style/TextAppearance</item>
<item name="textAppearanceInverse">@android:style/TextAppearance.Inverse</ite

<item name="textColorPrimary">@android:color/primary_text_dark</item>
<item name="textColorSecondary">@android:color/secondary_text_dark</item>
<item name="textColorTertiary">@android:color/tertiary_text_dark</item>
<item name="textColorPrimaryInverse">@android:color/primary_text_light</item>
<item name="textColorSecondaryInverse">@android:color/secondary_text_light</i
<item name="textColorTertiaryInverse">@android:color/tertiary_text_light</ite
<item name="textColorPrimaryDisableOnly">@android:color/primary_text_dark_dis
```

STYLES

Styles

- Defined in XML file
- res/values/style
- similar to a cascading style sheet as used in html
- group layout attributes in a style and apply to various View objects (TextView, EditText, Button)

Sample Styles, in styles.xml

```
<style name="sample1">
    <item name="android:textSize">20pt</item>
    <item name="android:textColor">@color/Orange</item>
    <item name="android:textStyle">bold</item>
    <item name="android:gravity">center</item>
    <item name="android:padding">10dp</item>
</style>

<style name="sample2">
    <item name="android:textSize">8pt</item>
    <item name="android:textColor">@color/AliceBlue</item>
    <item name="android:textStyle">italic</item>
    <item name="android:gravity">right</item>
    <item name="android:padding">2dp</item>
</style>
```

Apply Style - in main xml

```
<TextView
    android:id="@+id/textView1"
    style="@style/sample2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="field number 1" />

<EditText
    android:id="@+id/editText1"
    style="@style/sample1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:inputType="textCapWords"
    android:text="First Edit Text" />

<TextView
    android:id="@+id/textView2"
    style="@style/sample2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="field number 2" />
```

Result of Styles

- can override elements of style
  - bottom edit text overrides color
- one style can inherit from another
- use UI editor to create view and then extract to style