Android Development

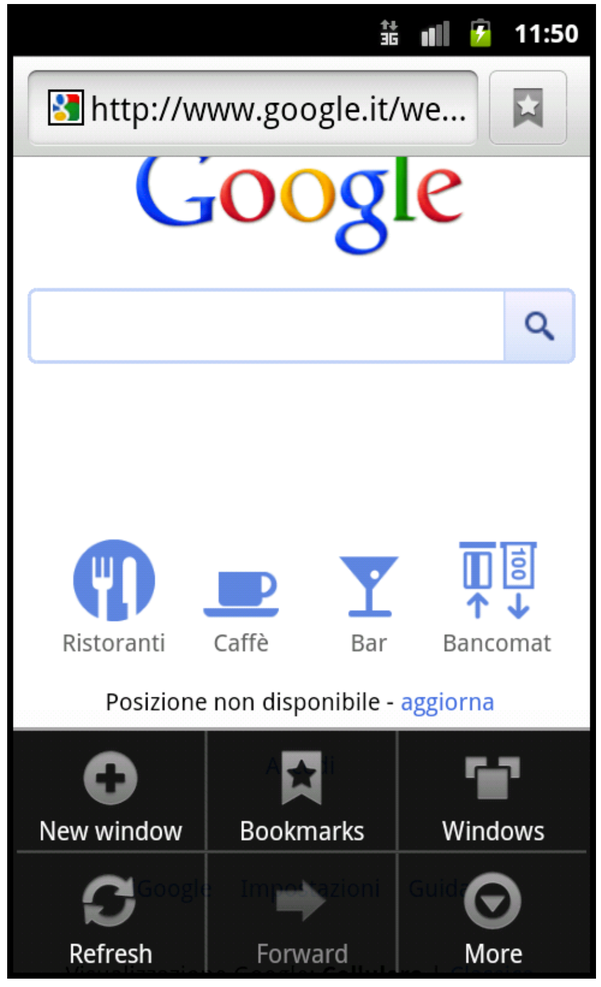
Lecture 4
Android Graphical User Interface 2

Lecture Summary

- Application Menu
- The ActionBar
- Handling Actions with the ActionBar
- Navigation with the ActionBar
- ActionBar & Tabs
- View Pager
- Action Bar & View Pager

Application Menu

- Menus are a common user interface component in many types of applications. For all menu types, Android provides a standard XML format to define menu items. Instead of building a menu in your activity's code, you should define a menu and all its items in an XML menu resource. You can then inflate the menu resource (load it as a Menu object) in your activity or fragment.



- Using a menu resource is a good practice for a few reasons:
  - It's easier to visualize the menu structure in XML.
  - It separates the content for the menu from your application's behavioral code.
  - It allows you to create alternative menu configurations for different platform versions, screen sizes, and other configurations by leveraging the app resources framework.

Application Menu

- To define the menu, create an XML file inside your project's res/menu/ directory and build the menu with the following elements:
- **<menu>**
  - Defines a Menu, which is a container for menu items. A <menu> element must be the root node for the file and can hold one or more <item> and <group> elements.
- **<item>**
  - Creates a MenuItem, which represents a single item in a menu. This element may contain a nested <menu> element in order to create a submenu.
- **<group>**
  - An optional, invisible container for <item> elements. It allows you to categorize menu items so they share properties such as active state and visibility.

Application Menu

```xml
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/newBookmark"
          android:icon="@drawable/ic_menu_add"
          android:title="@string/menu_add"/>
    <item android:id="@+id/appInfo"
          android:icon="@drawable/ic_menu_info"
          android:title="@string/menu_info" />
</menu>
```

- To specify the options menu for an Activity, override onCreateOptionsMenu() (fragments provide their own onCreateOptionsMenu() callback). In this method, you can inflate your menu resource (defined in XML) into the Menu provided in the callback.

```java
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.game_menu, menu);
    return true;
}
```

Application Menu

- When the user selects an item from the options menu (including action items in the action bar), the system calls your activity's onOptionsItemSelected() method.
- This method passes the MenuItem selected. You can identify the item by calling getItemId(), which returns the unique ID for the menu item (defined by the android:id attribute in the menu resource or with an integer given to the add() method). You can match this ID against known menu items to perform the appropriate action.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle item selection
    switch (item.getItemId()) {
        case R.id.appInfo:
         openAppInfoActivity();
            return true;
        case R.id.newBookmark:
         openAddBookmarkActivity();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Application Menu / Action Bar

- Beginning with Android 3.0 (API level 11), Android-powered devices are no longer required to provide a dedicated Menu button. With this change, Android apps should migrate away from a dependence on the traditional menu panel to the action bar.

- The action bar as a combination of on-screen action items and overflow options. On Android 3.0 devices, the Menu button is deprecated (some devices don't have one), so you should migrate toward using the action bar to provide access to actions and other options.

http://developer.android.com/guide/topics/ui/menus.html

http://developer.android.com/guide/topics/ui/actionbar.html

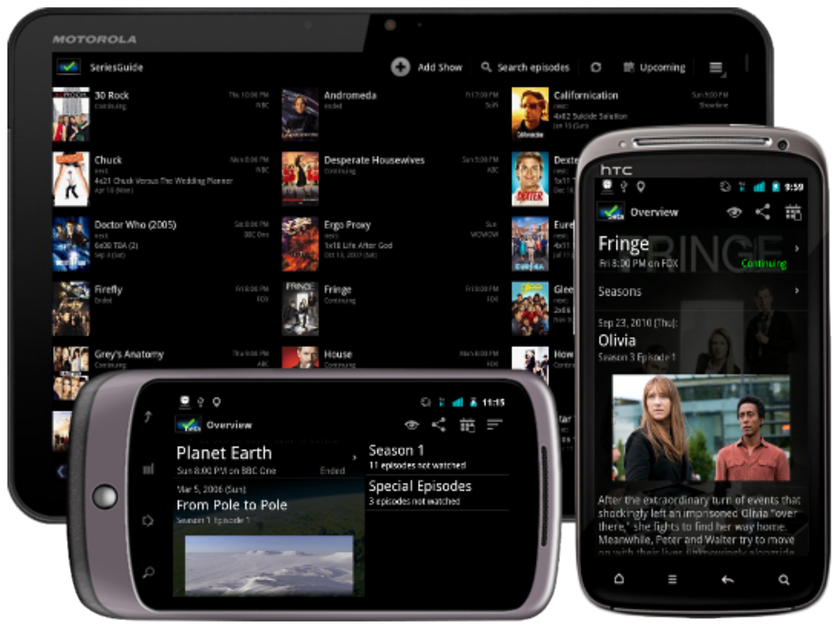
http://android-developers.blogspot.it/2012/01/say-goodbye-to-menu-button.html

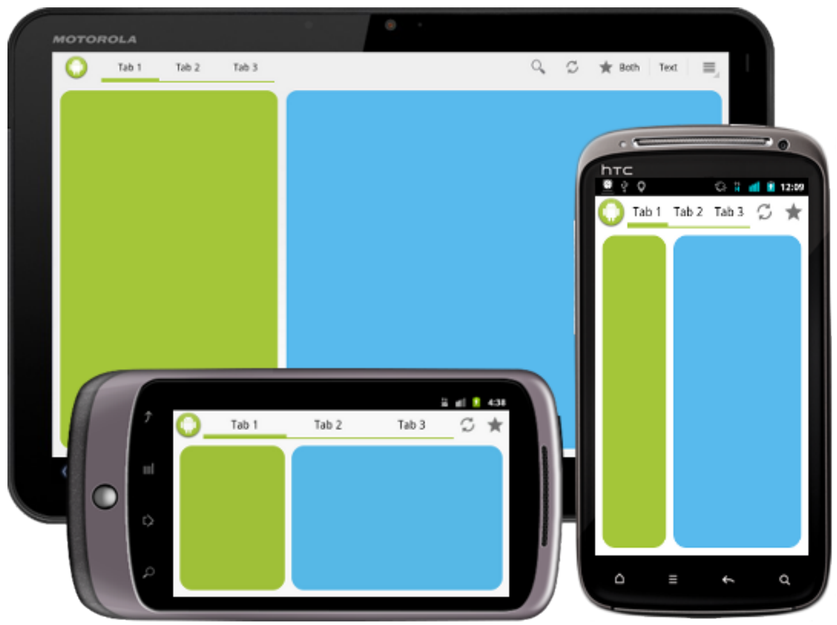
http://android-developers.blogspot.it/2011/09/preparing-for-handsets.html?utm_source=feedburner&utm_medium=feed&utm_campaign=Feed:+blogspot/hsDu+(Android+Developers+Blog)

http://www.vogella.com/articles/AndroidActionBar/article.html

ActionBar Sherlock
http://actionbarsherlock.com/

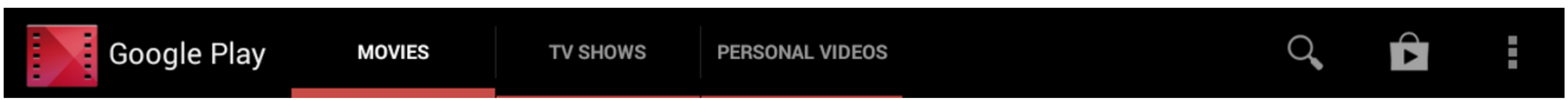
- *"ActionBarSherlock is an extension of the support library designed to facilitate the use of the action bar design pattern across all versions of Android with a single API."*
- *"The library will automatically use the native action bar when appropriate or will automatically wrap a custom implementation around your layouts. This allows you to easily develop an application with an action bar for every version of Android from 2.x and up."*





ActionBar

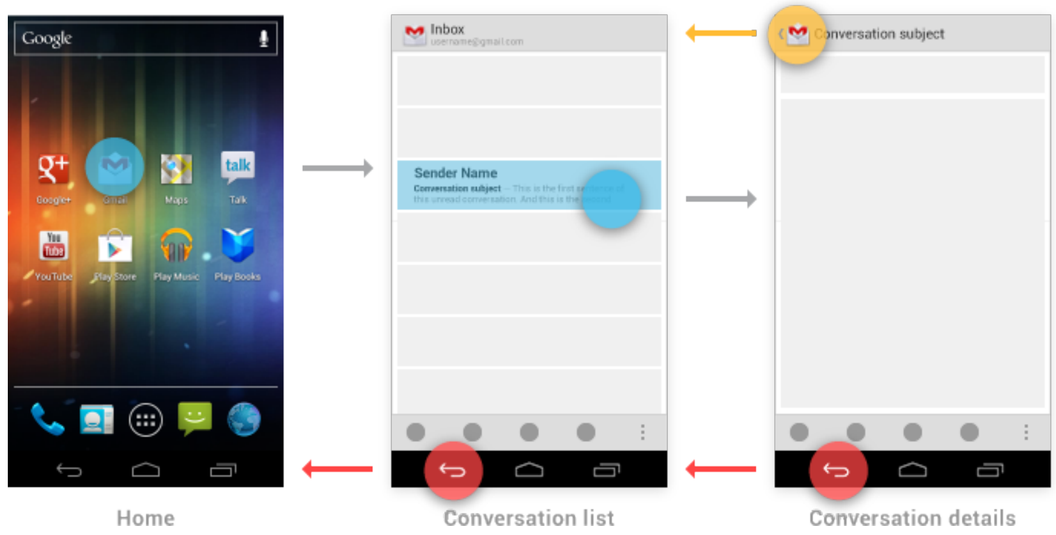
- The action bar is a window feature that allows to
  - identify the user location (current Activity/View)
  - provide user actions and navigation modes
  - offer a familiar interface across applications that the system properly adapts for different screen configurations.
  - support consistent navigation and view switching within apps (with tabs or drop-down lists).



UI Design Guide: http://developer.android.com/design/patterns/actionbar.html

ActionBar & Navigation

- *“Consistent navigation is an essential component of the overall user experience.”*
- With Android 2.3 and earlier OS versions the navigation was based on system Back button.
- With the introduction of action bars in Android 3.0, a second navigation mechanism appeared using the **Up button** (the app icon and a left-point caret).



ActionBar & Navigation Approach: http://developer.android.com/design/patterns/navigation.html

ActionBar & Compatibility

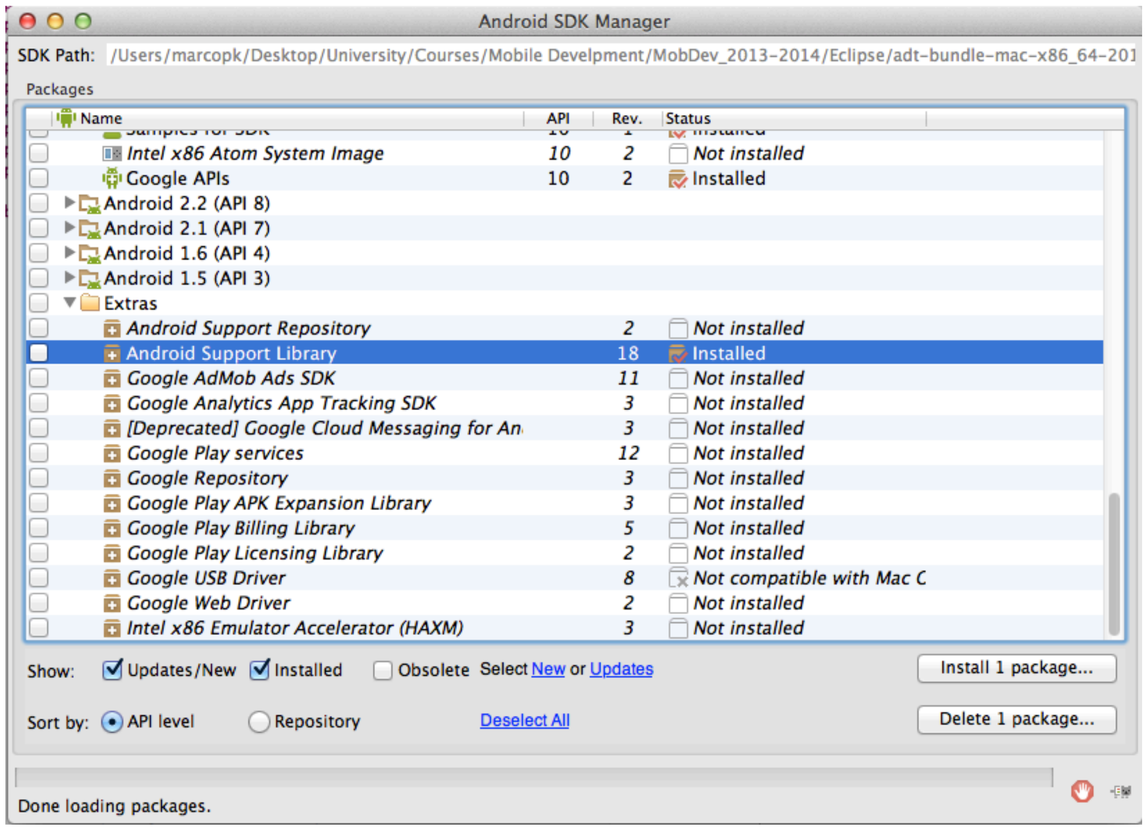
- The ActionBar APIs were first added in Android 3.0 (API level 11)
- They are also available in the **Support Library** for compatibility with Android 2.1 (API level 7) and above.
- In order to support previous Android versions it is also possible to use external Libraries such as the Sherlock Action Bar. (Provides the same methods of the official API and could be easily used also in existing projects)
- Using the official Android API you should remember that:
  - To support API levels lower than 11: import android.support.v7.app.ActionBar
  - To support only API level 11 and higher: import android.app.ActionBar
- During the course we adopt the official API

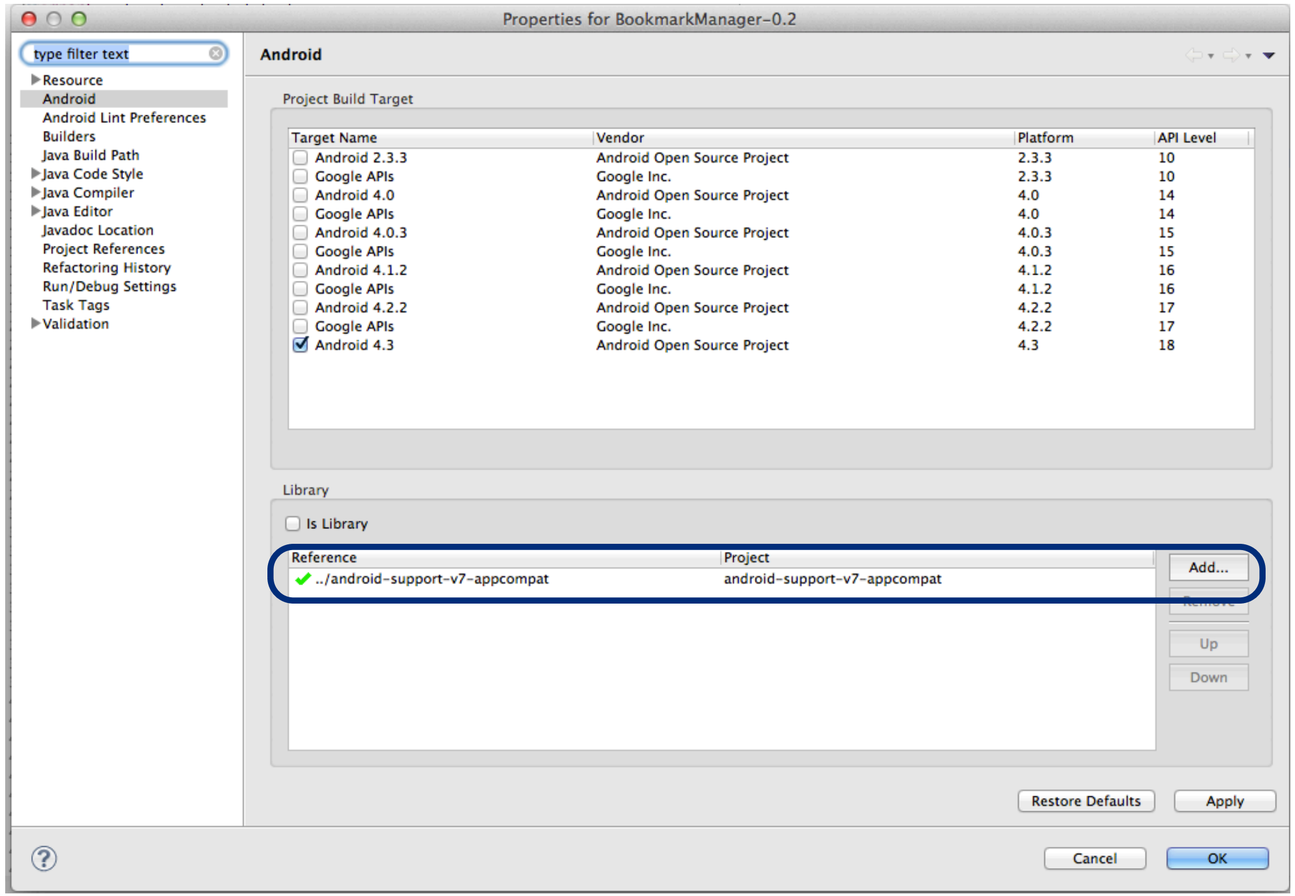
ActionBar & Support Library

(1) - Android SDK Manager

(2) Import into your workspace the project in <your_sdk_path>/extras/android/support/v7/appcompat

(3) - Update your Project Properties adding the imported project as library for your project





Adding the ActionBar

- In order to add and use the ActionBar to your project you must:
  - set up your project with the appcompat v7 support library.
  - Create your activity by extending ActionBarActivity.
    - Use (or extend) one of the Theme.AppCompat themes for your activity. For example:

```
<activity android:theme="@style/Theme.AppCompat.Light" ... >
```

- The right theme allow the OS to correctly load and create an ActionBar for your Activity. Some themes do not provides the support for the ActionBar and your will receive a null object in your Activity.

Action Bar Code

```
<application
       android:allowBackup="true"
       android:icon="@drawable/ic_launcher"
       android:label="@string/app_name"
       android:theme="@style/Theme.AppCompat">

   <activity
           android:name=".MainActivity"
           android:label="@string/app_name">
           <intent-filter>
               <action android:name="android.intent.action.MAIN" />

               <category android:name="android.intent.category.LAUNCHER" />
           </intent-filter>
       </activity>

...

 </application>
```

```
public class MainActivity extends ActionBarActivity{

...

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

        // Set up the action bar.
        ActionBar actionBar = getSupportActionBar();

        // Specify that the Home/Up button should not be enabled,
        //since there is no hierarchical parent.
        actionBar.setHomeButtonEnabled(false);

        //UnComment the following lines to hide the top part of the action bar
        //actionBar.setDisplayShowTitleEnabled(false);
        //actionBar.setDisplayHomeAsUpEnabled(false);
        //actionBar.setDisplayShowHomeEnabled(false);
    }

}
```

getSupportActionBart() if we have the support library v7 otherwise just getActionBar().

Methods to display or not some part of the ActionBar

ActionBar Actions

- The action bar provides users access to the most important action items relating to the app's current context.
- Those that appear directly in the action bar with an icon and/or text are known as action buttons. Actions that can't fit in the action bar or aren't important enough are hidden in the action overflow.
- The user can reveal a list of the other actions by pressing the overflow button on the right side (or the device Menu button, if available).
- When your activity starts, the system populates the action items by calling your activity's onCreateOptionsMenu() method. Use this method to inflate a menu resource that defines all the action items. For example, here's a menu resource defining a couple of menu items:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <item android:id="@+id/action_search"
          android:icon="@drawable/ic_action_search"
          android:title="@string/action_search"/>
    <item android:id="@+id/action_compose"
          android:icon="@drawable/ic_action_compose"
          android:title="@string/action_compose" />
</menu>
```

res/menu/main_activity_actions.xml

ActionBar Actions

```xml
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <item android:id="@+id/action_search"
          android:icon="@drawable/ic_action_search"
          android:title="@string/action_search"/>
    <item android:id="@+id/action_compose"
          android:icon="@drawable/ic_action_compose"
          android:title="@string/action_compose" />
</menu>
```

res/menu/main_activity_actions.xml

XML inflated in the method onCreateOptionsMenu(...)

```java
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu items for use in the action bar
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.main_activity_actions, menu);
    return super.onCreateOptionsMenu(menu);
}
```

ActionBar Actions

- To request that an item appear directly in the action bar as an action button, include showAsAction="ifRoom"in the <item> tag.
- If there's not enough room for the item in the action bar, it will appear in the action overflow.

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:yourapp="http://schemas.android.com/apk/res-auto" >
    <item android:id="@+id/action_search"
          android:icon="@drawable/ic_action_search"
          android:title="@string/action_search"
          yourapp:showAsAction="ifRoom"  />
    ...
</menu>
```

If your menu item use a title and an icon—with the title and icon attributes—then the action item shows only the icon by default. If you want to display the text title, add "withText" to the showAsAction attribute.

```
<item yourapp:showAsAction="ifRoom|withText" ... />
```

Handling Actions

- When the user presses an action, the system calls your activity's onOptionsItemSelected() method.
- Using the MenuItem passed to this method, you can identify the action by calling getItemId().
- This returns the unique ID provided by the <item> tag's id attribute so you can perform the appropriate action.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle presses on the action bar items
    switch (item.getItemId()) {
        case R.id.action_search:
            openSearch();
            return true;
        case R.id.action_compose:
            composeMessage();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

ActionBar & Navigation

- Enabling the app icon as an Up button allows the user to navigate your app based on the hierarchical relationships between screens. For instance, if screen A displays a list of items, and selecting an item leads to screen B, then screen B should include the Up button, which returns to screen A.

- To enable the app icon as an Up button, call setDisplayHomeAsUpEnabled().

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_details);

    ActionBar actionBar = getSupportActionBar();
    actionBar.setDisplayHomeAsUpEnabled(true);
    ...
}
```

Now the icon in the action bar appears with the *Up* caret (as shown in figure 4). However, it won't do anything by default. To specify the activity to open when the user presses *Up* button, you have two options:

ActionBar & Navigation - XML

- This is the best option when the parent activity is always the same. By declaring in the manifest which activity is the parent, the action bar automatically performs the correct action when the user presses the Upbutton.
- Beginning in Android 4.1 (API level 16), you can declare the parent with the parentActivityName attribute in the <activity> element.
- To support older devices with the support library, also include a <meta-data> element that specifies the parent activity as the value for android.support.PARENT_ACTIVITY.
- Once the parent activity is specified in the manifest like this and you enable the Up button with setDisplayHomeAsUpEnabled(), your work is done and the action bar properly navigates up.

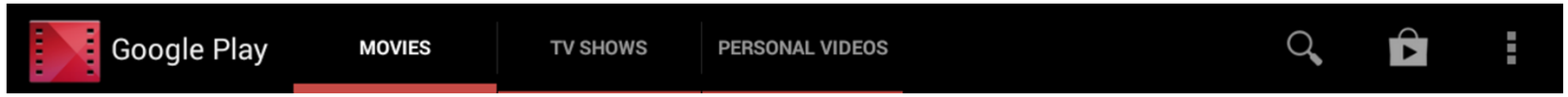
```
<application ... >
    ...
    <!-- The main/home activity (has no parent activity) -->
    <activity
        android:name="com.example.myfirstapp.MainActivity" ...>
        ...
    </activity>
    <!-- A child of the main activity -->
    <activity
        android:name="com.example.myfirstapp.DisplayMessageActivity"
        android:label="@string/title_activity_display_message"
        android:parentActivityName="com.example.myfirstapp.MainActivity" >
        <!-- Parent activity meta-data to support API level 7+ -->
        <meta-data
            android:name="android.support.PARENT_ACTIVITY"
            android:value="com.example.myfirstapp.MainActivity" />
    </activity>
</application>
```

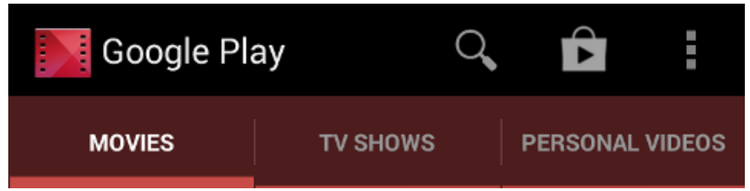
ActionBar & Navigation - Code

- Define the navigation through code is appropriate when the parent activity may be different depending on how the user arrived at the current screen.
- The system calls getSupportParentActivityIntent() when the user presses the Up button while navigating your app.
- If the activity that should open upon up navigation differs depending on how the user arrived at the current location, then you should override this method to return the Intent that starts the appropriate parent activity.
- The system calls onCreateSupportNavigateUpTaskStack() for your activity when the user presses the Up button while your activity is running in a task that does not belong to your app. Thus, you must use theTaskStackBuilder passed to this method to construct the appropriate back stack that should be synthesized when the user navigates up.
- Even if you override getSupportParentActivityIntent() to specify up navigation as the user navigates your app, you can avoid the need to implement onCreateSupportNavigateUpTaskStack() by declaring "default" parent activities in the manifest file as shown above. Then the default implementation of onCreateSupportNavigateUpTaskStack() will synthesize a back stack based on the parent activities declared in the manifest.

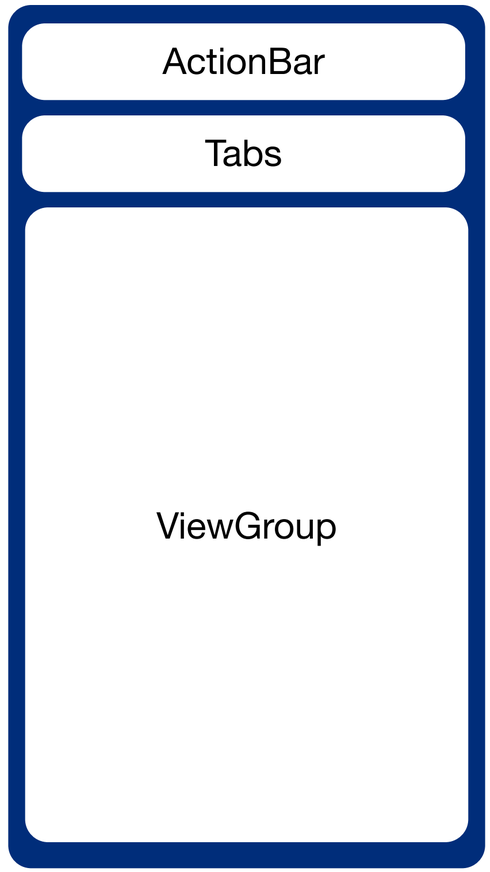
Navigating Up with the App Icon
http://developer.android.com/guide/topics/ui/actionbar.html

ActionBar & Navigation Tabs





- Tabs in the action bar make it easy for users to explore and switch between different views in your app.
- The tabs provided by the ActionBar are able adapt to different screen sizes.
- Your layout must include a ViewGroup in which you place each Fragment associated with a tab.
- Verify that the ViewGroup has a resource ID so you can reference it from your code and swap the tabs within it.
- If the tab content will fill all the activity layout, then your activity does not need a layout at all (you do not even need to call setContentView()). Instead, you can place each fragment in the default root view, which you can refer to with the android.R.id.content ID.



ActionBar & Navigation Tabs

- Once you determine where the fragments appear in the layout, the basic procedure to add tabs is:
  - Implement the ActionBar.TabListener interface.
  - TabListener provides callbacks for tab events, such as when the user presses one so you can swap the tabs.
  - For each tab you want to add
    - instantiate an ActionBar.Tab
    - set the ActionBar.TabListener by calling setTabListener().
    - Set the tab's title and with setText() (and optionally, an icon with setIcon()).
  - Add each tab to the action bar by calling addTab().

ActionBar & Navigation Tabs

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

   //Retrieve the ActionBar
   final ActionBar actionBar = getActionBar();

   //Set the Navigation mode to support tabs with the ActionBar
   actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);

   // for each of the sections in the app, add a tab to the action bar.
   actionBar.addTab(actionBar.newTab().setText("Section 1").setTabListener(this));
   actionBar.addTab(actionBar.newTab().setText("Section 2").setTabListener(this));
   actionBar.addTab(actionBar.newTab().setText("Section 3").setTabListener(this));
}
```

Set the layout with the ViewGroup that will contain the Fragment associated to the selected Tab.

Set the navigation mode in the ActionBar in order to support multiple tabs and its management in the Activity.

ActionBar & Navigation Tabs

ViewGroup Layout used to contain the Fragment associated to each Tab in the ActionBar.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/container"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
</LinearLayout>
```

Id that will be used in the ActionBar.TabListener methods to push the right Fragment according to the Tab.

ActionBar & Navigation Tabs

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

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        ...
    }

    @Override
    public void onTabReselected(Tab tab, FragmentTransaction fragmentTransaction) {
    }

    @Override
    public void onTabSelected(Tab tab, FragmentTransaction fragmentTransaction) {
    }

    @Override
    public void onTabUnselected(Tab tab, FragmentTransaction fragmentTransaction) {
    }
}
```

Implement ActionBar.TabListener to define the behavior of the tabs.

Methods provided by the listener to receive the callbacks when the user interacts with a specific tab.

ActionBar & Navigation Tabs

```
@Override
public void onTabSelected(Tab tab, FragmentTransaction fragmentTransaction) {
    // When the given tab is selected, show the tab contents in the container view.
    Fragment fragment = new DemoSectionFragment();
    Bundle args = new Bundle();
    args.putInt(DemoSectionFragment.ARG_SECTION_NUMBER,tab.getPosition() + 1);
    fragment.setArguments(args);

    getFragmentManager().beginTransaction().replace(R.id.container, fragment).commit();
}
```

Add the Fragment to the ViewGroup container identified by the resource R.id.container

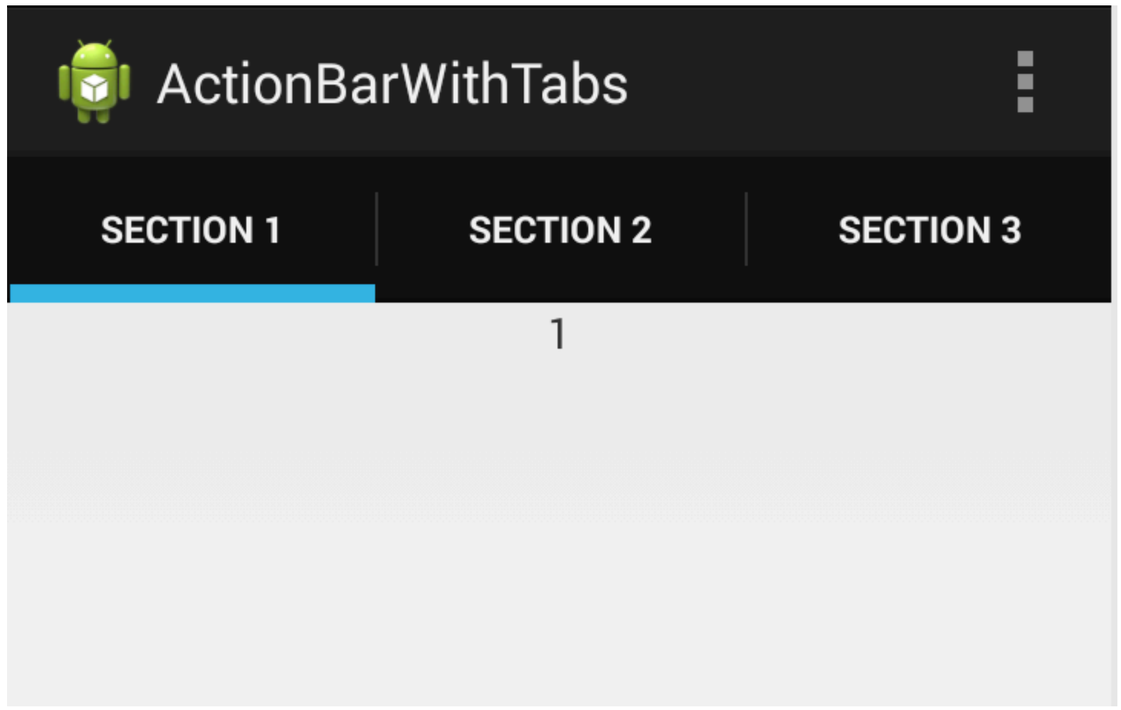
Define a generic Fragment with some additional parameters through a Bundle

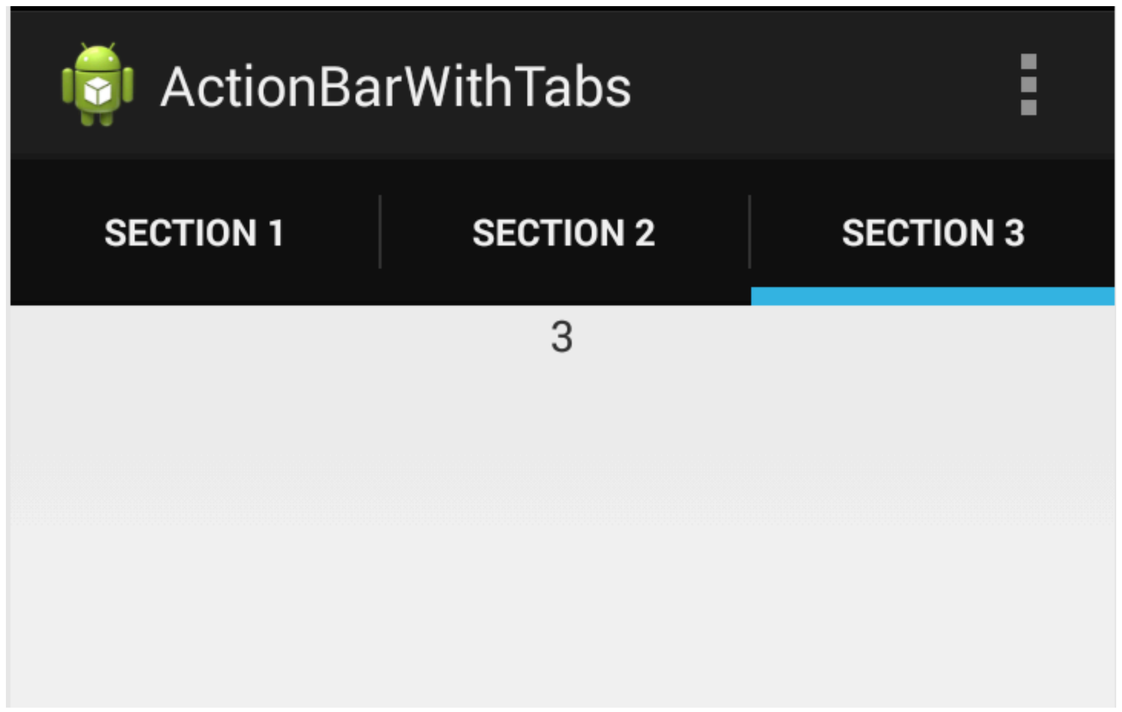
ActionBar & Navigation Tabs

```
public class DemoSectionFragment extends Fragment {

   public static final String ARG_SECTION_NUMBER = "placeholder_text";

   @Override
   public View onCreateView(LayoutInflater inflater, ViewGroup container,Bundle savedInstanceState) {
      TextView textView = new TextView(getActivity());
      textView.setGravity(Gravity.CENTER);
      textView.setText(Integer.toString(getArguments().getInt(ARG_SECTION_NUMBER)));
      return textView;

   }
}
```

A simple DemoFragment that shows only a TextView with a number specified through the Bundle.

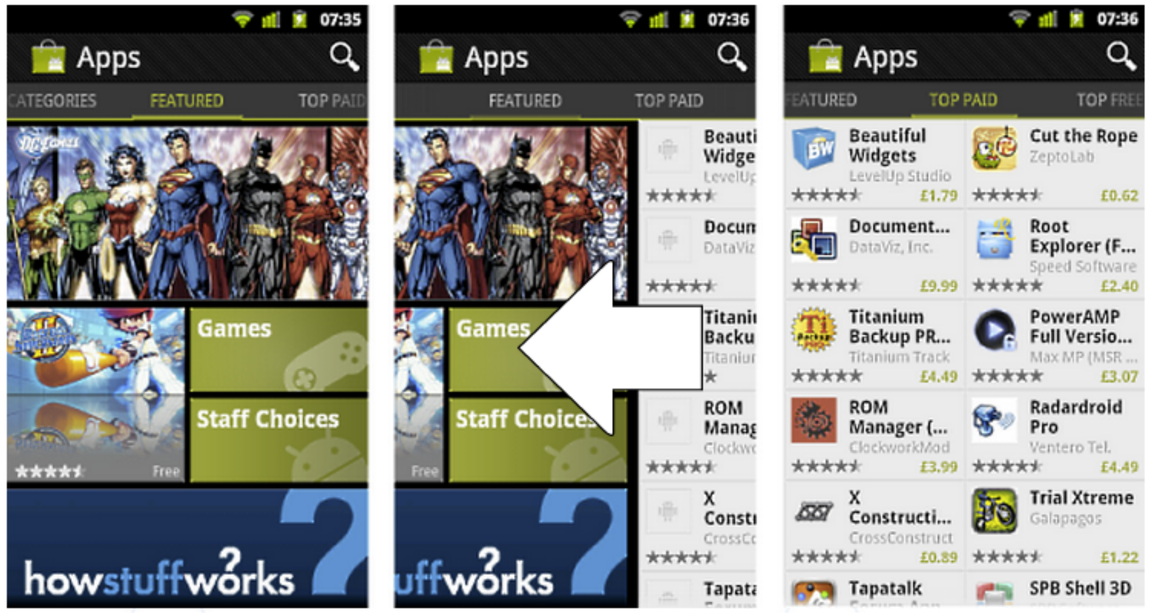
ActionBar & Navigation Tabs





ViewPager

- ViewPagers have built-in swipe gestures to transition through pages.
- They display screen slide animations by default.
- ViewPagers use PagerAdapters as a supply for new pages to display.
- PagerAdapter will use the Fragments to define the content of each page.
- ViewPager is a ViewGroup that works in a similar way to AdapterViews (like ListView and Gallery)
- ViewPager was released as part of the Compatibility Package revision 3 and works with Android 1.6 upwards.
- **[Note]** If you use the ViewPager in an xml layout, be sure to use the full class reference.



Additional Info (E.g Customized Animation)
http://developer.android.com/training/animation/screen-slide.html

ViewPager - Implementation

- Sets the content view to be the layout with the ViewPager
- Creates a class that extends the **FragmentStatePagerAdapter** abstract class and implements the getItem() method to supply instances of ScreenSlidePageFragment as new pages. The pager adapter also requires that you implement the getCount() method, which returns the amount of pages the adapter will create (five in the example).
- Hooks up the PagerAdapter to the ViewPager.
- Handles the device's back button by moving backwards in the virtual stack of fragments. If the user is already on the first page, go back on the activity back stack.

```
<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" />
```

ViewPager - Implementation

```
public class AppSectionsPagerAdapter extends FragmentPagerAdapter {

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

    /**
     * Returns the Fragment associated to the element in position 'i'
     */
    @Override
    public Fragment getItem(int i) {
        switch (i) {
        case 0:
            return new BookmarkGridFragment();
        case 1:
            return new BookmarkListFragment();
        default:
            return new BookmarkGridFragment();
        }
    }
```

```
    /**
     * Returns the number of element in the ViewPager
     */
    @Override
    public int getCount() {
        return 2;
    }

    /**
     * Returns the title (as CharSequence) of the Page at position 'position'
     */
    @Override
    public CharSequence getPageTitle(int position) {
        switch (position) {
        case 0:
            return "Add";
        case 1:
            return "Top";
        case 2:
            return "My List";
        default:
            return "No Title !";
        }
    }
}
```

ViewPager - Implementation

Create the ViewPager Adapter

```
mAppSectionsPagerAdapter = new AppSectionsPagerAdapter(getSupportFragmentManager());
```

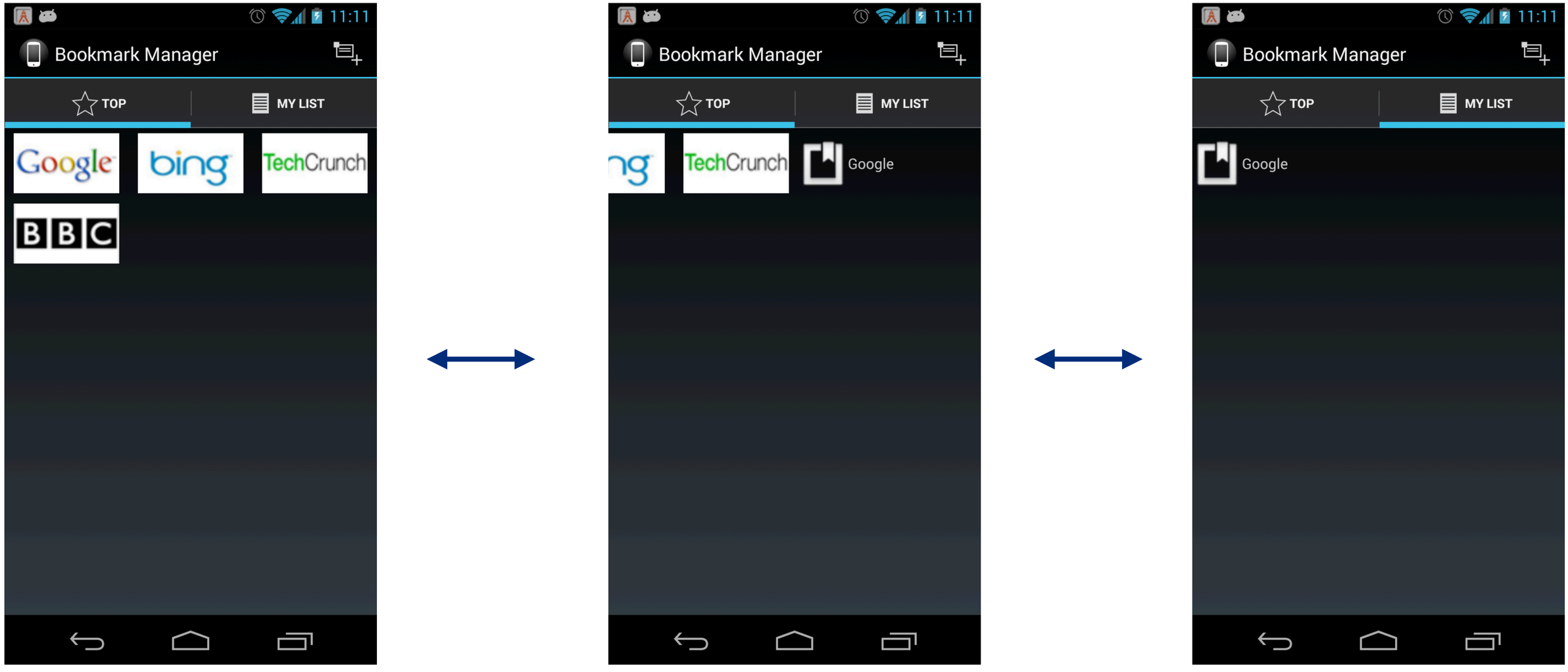
```
// Set up the ViewPager, attaching the adapter and setting up a listener for when the
// user swipes between sections.
mViewPager = (ViewPager) findViewById(R.id.pager);

mViewPager.setAdapter(mAppSectionsPagerAdapter);
```

Retrieve the Pager and set the adapter

```
mViewPager.setOnPageChangeListener(new ViewPager.SimpleOnPageChangeListener() {
    @Override
    public void onPageSelected(int position) {
        ...
    }
});
```

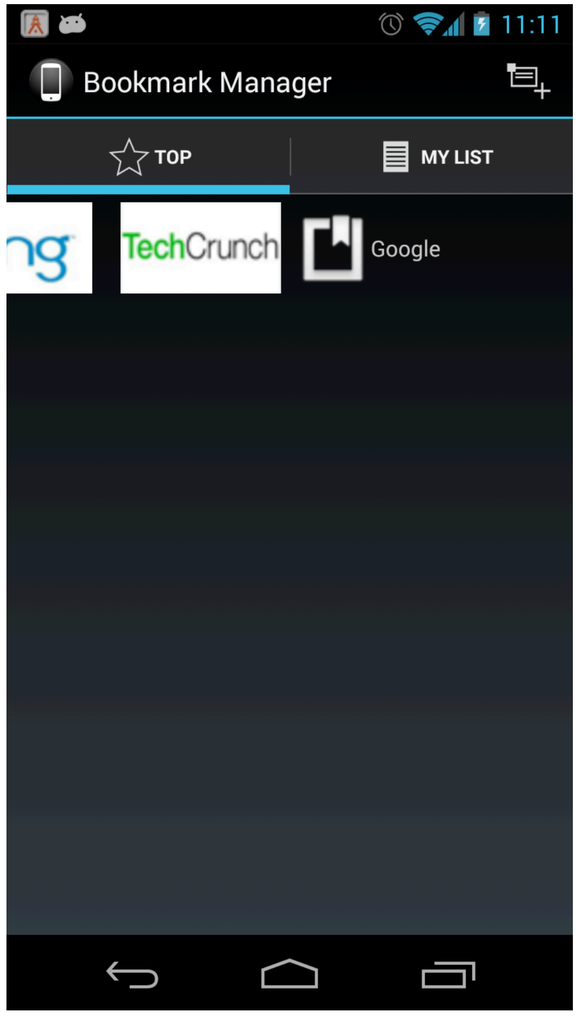
Listener to react when a specific page has been selected by the user

ActionBar & ViewPager



ActionBar & ViewPager

- The idea is to combine the functionalities of the ActionBar with a ViewPager to easily manage multiple sections in your applications.
- The ActionBar with Tabs allows to view the available sections and click on the selected Tab to switch to the associated View/Fragment
- The ViewPager allows to navigate left and right through the built-in swipe gestures
- The challenging part is to properly manage the behavior of one component according to the controller of the other one.
- E.g: React to a swipe gesture changing the Page in the ViewPager and show the right selected Tab in the ActionBar



→ ActionBar

→ View Pager

ActionBar & ViewPager - Implementation

AndroidManifest with the right Theme to support the ActionBar

```
<application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/Theme.AppCompat">
        <activity
            android:name="it.unipr.mobdev.MainActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        ...

    </application>
```

XML with the ViewPager for the MainActivity layout

```
<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"
/>
```

ActionBar & ViewPager - Implementation

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

    private AppSectionsPagerAdapter mAppSectionsPagerAdapter;

    private ViewPager mViewPager;

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

        mAppSectionsPagerAdapter = new AppSectionsPagerAdapter(getSupportFragmentManager());

        final ActionBar actionBar = getSupportActionBar();
        actionBar.setHomeButtonEnabled(false);
        actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);
```

```
mViewPager = (ViewPager) findViewById(R.id.pager);
mViewPager.setAdapter(mAppSectionsPagerAdapter);
mViewPager.setOnPageChangeListener(new ViewPager.SimpleOnPageChangeListener() {
    @Override
    public void onPageSelected(int position) {
        actionBar.setSelectedNavigationItem(position);
    }
});
```

ViewPager -> ActionBar

```
        Tab topTab = actionBar.newTab();
        topTab.setText(mAppSectionsPagerAdapter.getPageTitle(1)).setTabListener(this);
        topTab.setIcon(R.drawable.ic_action_rating_not_important);
        actionBar.addTab(topTab);

        Tab myTab = actionBar.newTab();
        myTab.setText(mAppSectionsPagerAdapter.getPageTitle(2)).setTabListener(this);
        myTab.setIcon(R.drawable.ic_action_collections_view_as_list);
        actionBar.addTab(myTab);
    }
```

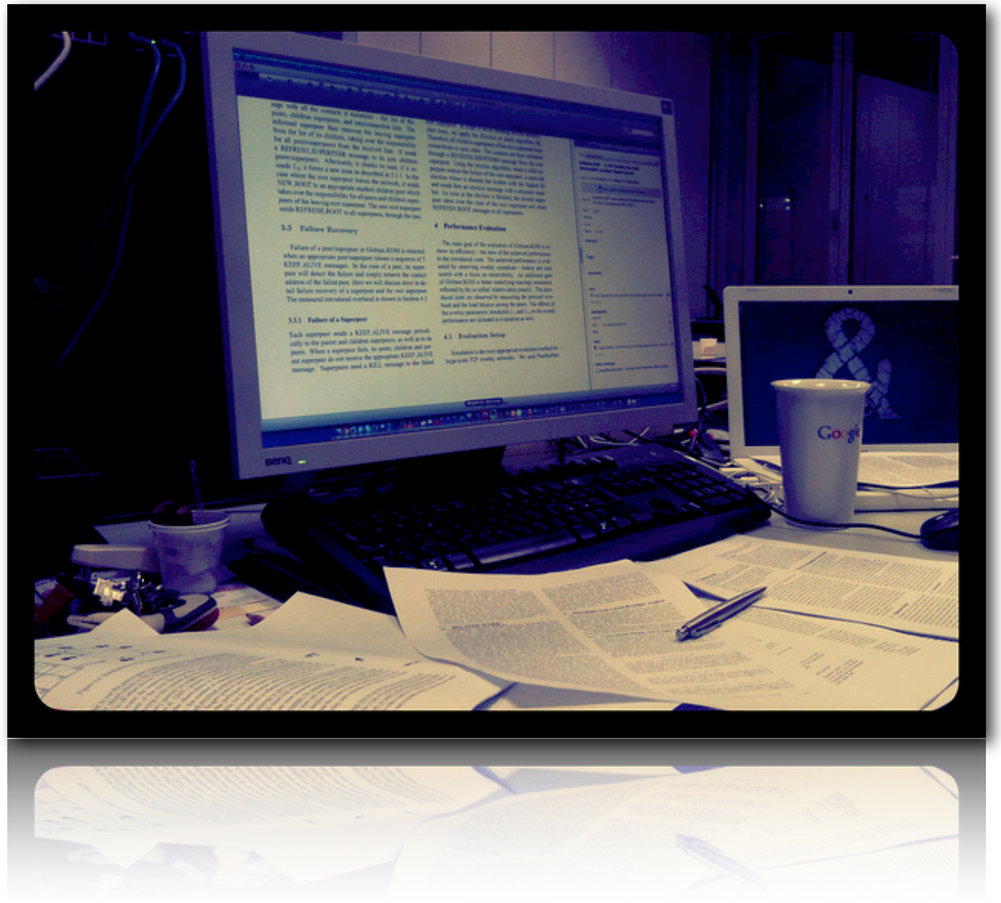
ActionBar & ViewPager - Implementation

```
@Override
public void onTabUnselected(ActionBar.Tab tab, android.support.v4.app.FragmentTransaction fragmentTransaction) {
}

@Override
public void onTabSelected(ActionBar.Tab tab, android.support.v4.app.FragmentTransaction fragmentTransaction) {
    // When the given tab is selected, switch to the corresponding page in the ViewPager.
    mViewPager.setCurrentItem(tab.getPosition());
}

@Override
public void onTabReselected(ActionBar.Tab tab, android.support.v4.app.FragmentTransaction fragmentTransaction) {
}
```

ActionBar -> ViewPager

Coming Up

- Next Lecture
  - Android Graphical User Interface 3
- Homework
  - Review
    - Bookmark Application 1.0 & 2.0 with different libraries and compatibility support
    - Additional Provided Applications (ActionBar & Tabs)



Android Development

Lecture 4
Android Graphical User Interface 2