Lesson 6

The ActionBar, Fragments and the TabHost

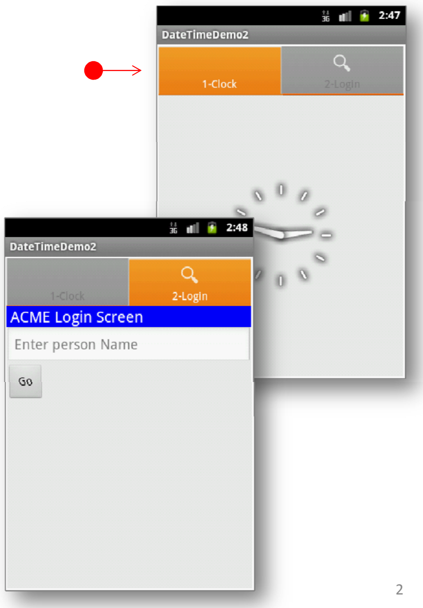
Victor Matos
Cleveland State University

TabHost Selection Widget

TabHost Selector

1. Handheld devices usually offer limited screen space.

2. Complex apps having many visual elements could benefit from the **Tab Host Widget** which maintains the awareness of the many pieces but shows only a few fragments at the time.



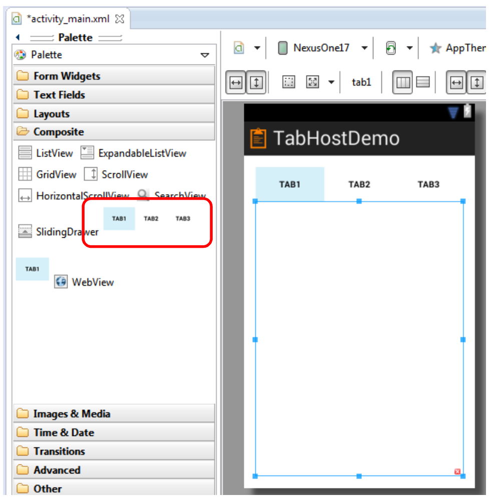
Note:
This is an aging GUI control. It is supported but is running out of favor. TabHosts are still useful for apps on SDK 3.0 or older.

TabHost Selection Widget

TabHost Anatomy

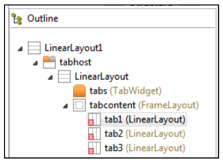
A TabHost control consists of three pieces that you need to set:

1. **TabHost** is the main container for the tab buttons and tab contents
2. **TabSpec** implements the row of tab buttons, which contain text labels (and optionally contain icons)
3. **FrameLayout** is the container for the tab contents

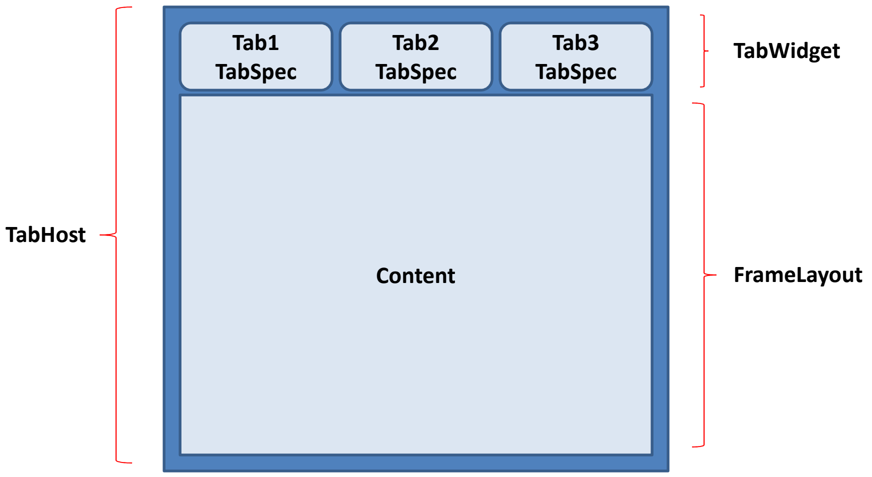


Look for the **Composite** portion of the Eclipse GUI Palette

TabHost Selection Widget

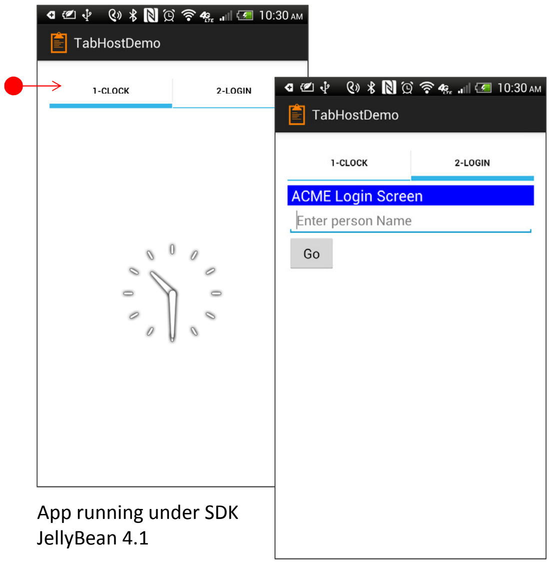


TabHost Selector – Components

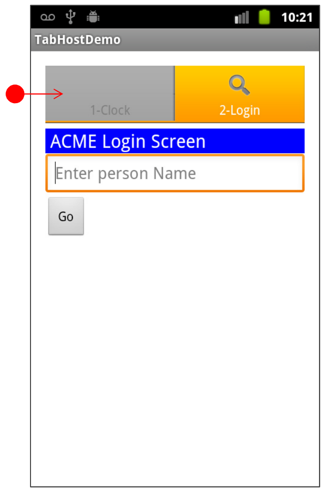


Example 1: TabHost Selection Widget

Using the TabHost GUI control



App running under SDK JellyBean 4.1



App running under SDK Ginger Bread 2.3

Example 1: TabHost Selection Widget

XML Layout – TabHostDemo – main_activity.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:background="#ffffffff"         android:orientation="vertical"
    android:padding="2dp"                  tools:context=".MainActivity" >
    <TabHost
        android:id="@android:id/tabhost"
        android:layout_width="match_parent"
        android:layout_height="match_parent" >
        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:orientation="vertical" >
            <TabWidget
                android:id="@android:id/tabs"
                android:layout_width="match_parent"
                android:layout_height="wrap_content" >
            </TabWidget>

            <FrameLayout
                android:id="@android:id/tabcontent"
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:paddingTop="6dip" >
                <include layout="@layout/main_tab1" />
                <include layout="@layout/main_tab2" />
            </FrameLayout>
        </LinearLayout>
    </TabHost>
</LinearLayout>
```

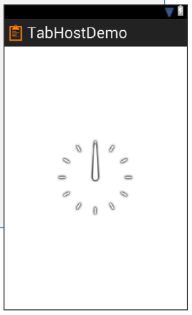
*You may enter here the actual layout specification, or (better) use the **<include>** tag to refer to an external layout assembled and stored in a separate xml file.*

Details in next pages…

Example 1: TabHost Selection Widget

XML Layout – TabHostDemo – main_tab1.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/tab1"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <AnalogClock
        android:id="@+id/tab1Clock"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:layout_gravity="center_horizontal" />

</LinearLayout>
```

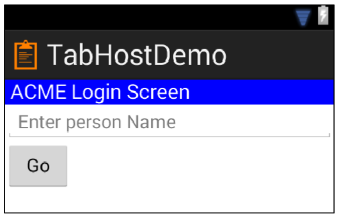


- This is the layout specification for **main_tab1.xml**.
- It is added to `activity_main.xml` using the clause `<include layout="@layout/main_tab1" />`
- This screen holds a centered AnalogClock widget

Example 1: TabHost Widget

This is **main_tab2.xml**.
It defines a *LinearLayout* holding a *label*, a textBox, and a *button*.

Inserted in main.xml using
<include layout=@Layout/... >



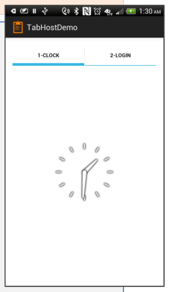
XML Layout – TabHostDemo – main_tab2.xml

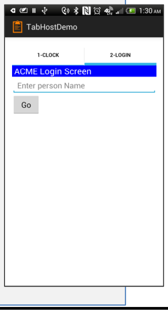
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/tab2"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="#ff0000ff"
        android:text=" ACME Login Screen"
        android:textColor="@android:color/white"
        android:textSize="20sp"  />

    <EditText
        android:id="@+id/tab2TxtPerson"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:hint="Enter person Name"
        android:inputType="textCapWords"
        android:textSize="18sp" />

    <Button
        android:id="@+id/tab2BtnGo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text=" Go " />

</LinearLayout>
```

Example 1: TabHost Selection Widget

TabHostDemo – ActivityMain Class

```java
public class MainActivity extends Activity {

  TabHost tabhost;

  @Override
  public void onCreate(Bundle icicle) {
    super.onCreate(icicle);
    setContentView(R.layout.activity_main);
    // wiring UI widgets shown in the various user-layouts
    // screen-1 components:
    final AnalogClock clock1 = (AnalogClock) findViewById(R.id.tab1Clock);
    // screen-2 components:
    final Button btnGo = (Button) findViewById(R.id.tab2BtnGo);
    final EditText txtPerson = (EditText) findViewById(R.id.tab2TxtPerson);

    // setting up Tabhost selector
    tabhost = (TabHost) findViewById(android.R.id.tabhost);
    tabhost.setup();
    TabHost.TabSpec tabspec;

    tabspec = tabhost.newTabSpec("screen1");
    tabspec.setContent(R.id.tab1);
    tabspec.setIndicator("1-Clock", null);
    tabhost.addTab(tabspec);

    tabspec = tabhost.newTabSpec("screen2");
    tabspec.setContent(R.id.tab2);
    tabspec.setIndicator("2-Login",
                         getResources().getDrawable(R.drawable.ic_menu_search));
    tabhost.addTab(tabspec);
```





Example 1: TabHost Selection Widget

TabHostDemo – ActivityMain Class

```java
    tabhost.setCurrentTab(0);
    // alternatively, you may also say
    // tabhost.setCurrentTabByTag("screen1");

    btnGo.setOnClickListener(new OnClickListener() {
      public void onClick(View v) {
        String theUser = txtPerson.getText().toString();
        txtPerson.setText("Hola " + theUser + " \n" + new Date());
        hideVirtualKeyboard();
      }
    });

    tabhost.setOnTabChangedListener(new OnTabChangeListener() {
      @Override
      public void onTabChanged(String tagId) {
        // do something useful with the selected screen
        String text = "Im currently on: " + tagId + "\nindex: "
            + tabhost.getCurrentTab();

        switch (tabhost.getCurrentTab()) {
        case 0: // do something for layout-0
          hideVirtualKeyboard();
          break;
        case 1: // do something for layout-1
          break;
        }
        Toast.makeText(getApplicationContext(), text, 1).show();
      }
    });
```

React to the clicking of the **GO** *button*

React to the selecting of a tab

Example 1: TabHost Selection Widget

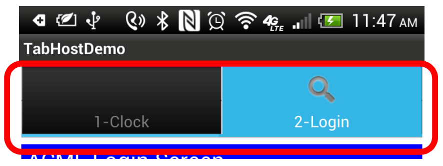
TabHostDemo – ActivityMain Class

```
  }// onCreate

      public void hideVirtualKeyboard() {
       // temporarily remove the virtual keyboard
       ((InputMethodManager) getSystemService(Activity.INPUT_METHOD_SERVICE))
                            .toggleSoftInput(InputMethodManager.SHOW_IMPLICIT, 0);
      }


     public void showVirtualKeyboard() {
       // no used - shown for completeness
       ((InputMethodManager) getSystemService(Activity.INPUT_METHOD_SERVICE))
                            .toggleSoftInput(InputMethodManager.SHOW_IMPLICIT, 0);
     }
}
```

Example 1: TabHost Selection Widget

HINT: Adding Icons to Tabs

You may decorate the tab indicator
Including text and image as shown
below:



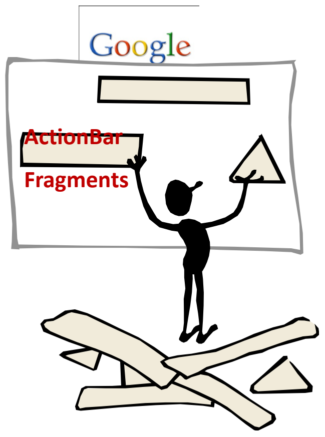
```
tabspec = tabhost.newTabSpec("screen2");

  tabspec.setContent(R.id.tab2);

  tabspec.setIndicator("2-Login",
          getResources().getDrawable(R.drawable.ic_action_search ));

tabhost.addTab(tabspec);
```

Note1:
Open the application's manifest and experiment changing its style. For instance, under the <Application> tag use the clause: `android:theme="@android:style/Theme.Black"`

Note2:
Many icons are available in: android-sdk-folder\docs\images\icon-design
Look also at: http://android-ui-utils.googlecode.com/hg/asset-studio/dist/index.html

Fragments & ActionBars

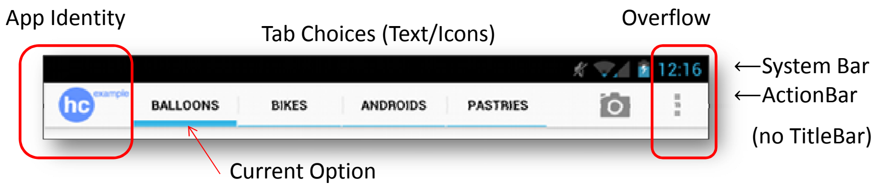
New way of doing things…



- It is very desirable to obtain a more common 'look-&-feel' appeal across applications and devices.
- This 'sameness' should make the user experience simpler and more enjoyable.

ActionBar Widget

The *action bar* is a dedicated strip-selector displayed at the top of each screen that is generally persistent throughout the app.

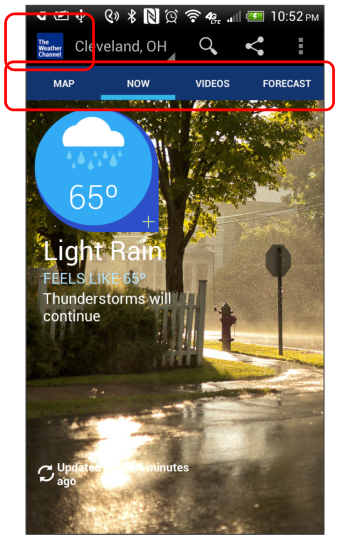


It provides several key functions:

1. Makes important actions prominent and accessible in a predictable way (such as *New* or *Search*).
2. Supports consistent navigation and view switching within apps.
3. Reduces clutter by providing an action overflow for rarely used actions.
4. Provides a dedicated space for giving your app an identity

Statements taken from:
http://developer.android.com/guide/topics/ui/actionbar.html#Tabs
http://developer.android.com/design/patterns/actionbar.html

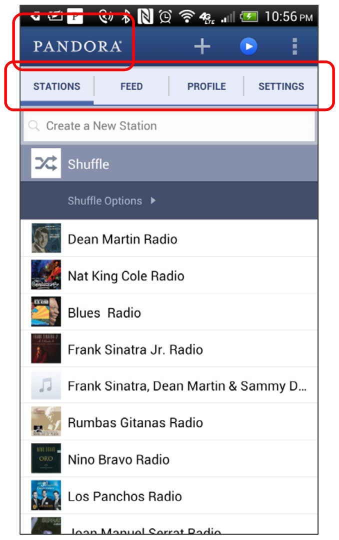
ActionBar Widget



← ActionBar →

Two different apps showing a relatively similar navigation pattern and visual structure.

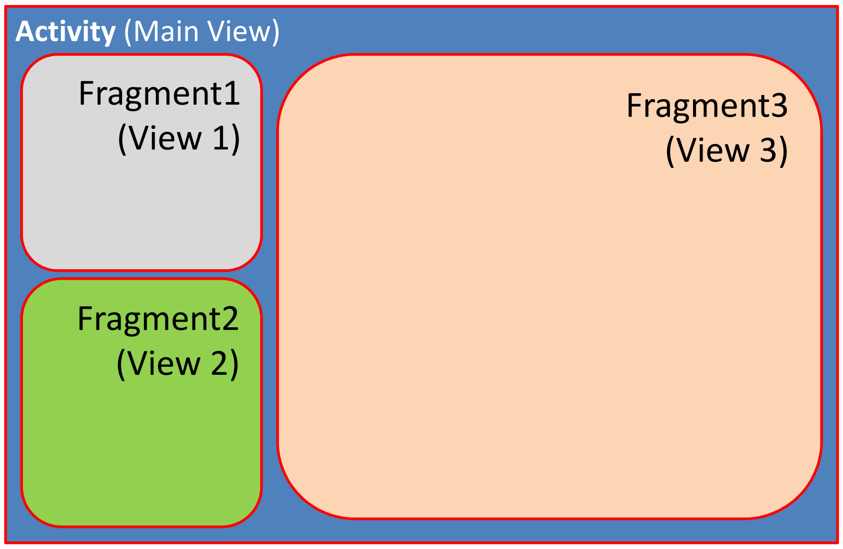
Note:
The ActionBar control requires the app to use the **Holo** theme (or one of its descendents)



Fragments

- A **Fragment** is either an expression of behavior or a portion of user interface in an **Activity**.
- One or more Fragments could attach to the main GUI of the activity in which they exists.
- Notably, all of them could be visible and active at the same time.
- Fragments behave as separate threads each running its on input/outputs, events and business logic.
- Fragments could reach 'global data' held in the main activity to which they belong. Likewise, they could send values of their own to the main activity for potential dissemination to other fragments.

Fragments



A possible arrangement of Fragments attached to the main GUI of an app.

Fragment's Lifecycle

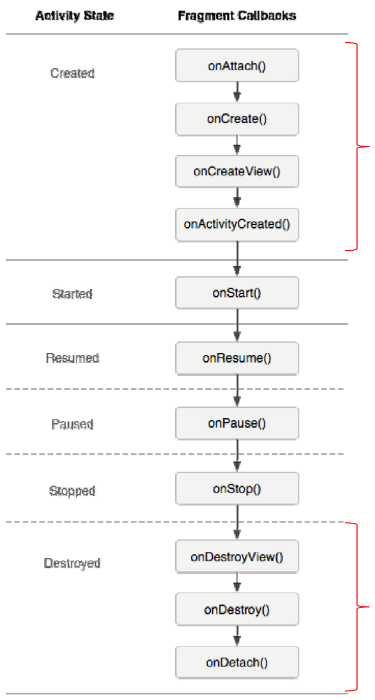
onAttach() Called when the fragment has been associated with the activity

onCreateView() Called to create the view hierarchy associated with the fragment.

onActivityCreated() Called when the activity's onCreate() method has returned.

onDestroyView() Called when the view hierarchy associated with the fragment is being removed.

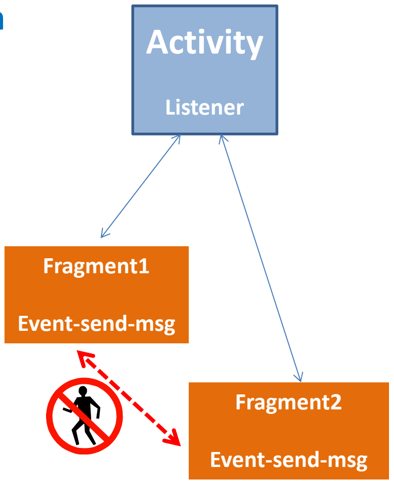
onDetach() Called when the fragment is being disassociated from the activity.



Fragments

Inter-Fragment Communication

- All Fragment-to-Fragment communication is done through the associated Activity.
- Two Fragments should ***never*** communicate directly.
- Activity and fragments interact through listeners and events.
- If a fragment has 'global' data to share, it should trigger an internal event to call the activity listener's attention and pass the global data to it.



Reference:
http://developer.android.com/training/basics/fragments/communicating.html

Example 2: Using Fragments and ActionBars

Example:

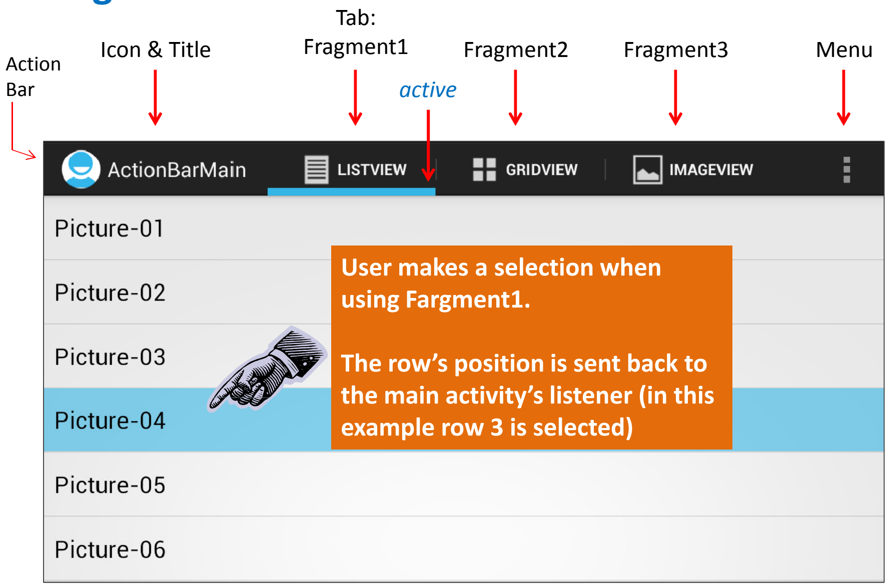
The application shows a multi-tabbed GUI from which a set of images could be examined. The 'look-&-feel' of the app is in line with the notion of standardization across devices /apps.

Individual tabs are implemented as Fragment objects. The screens operate as follows:

Tab1	Displays a list of picture *names*. When the fragment attaches to the main activity, a listener (in the main activity) is set to receive updates from the fragment's onItemSelected event. This strategy keeps the activity aware of selections made in fragment1.
Tab2	A *GridView* depicting all the images whose names were shown in fragment1 (TODO: keep activity informed of user's choices).
Tab3	A large *ImageView* display a 'good quality' version of the picture selected by the user in fragment1.

Example2: Using Fragments and ActionBars

Fragment1

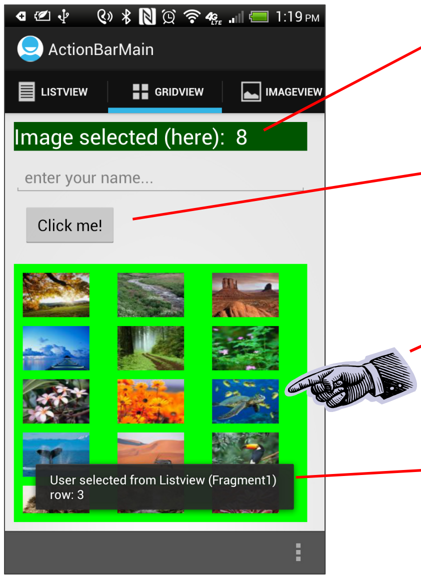


User makes a selection when using Fargment1.

The row's position is sent back to the main activity's listener (in this example row 3 is selected)

Example2: Using Fragments and ActionBars

Fragment2



User selected from Listview (Fragment1) row: 3

While working on this screen you tapped on image number …

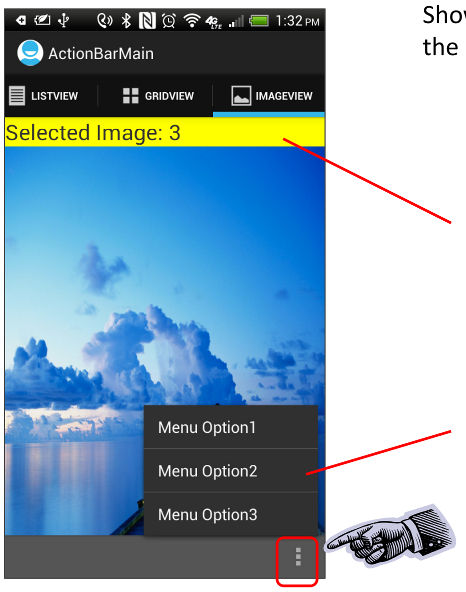
Button
Click to echo your name and current date/time

User makes a selection. Thumbnail position is locally recognized.

Notification of previous action made in Fragment1
User selected in fragment1 row number ….

Example2: Using Fragments and ActionBars

Fragment3

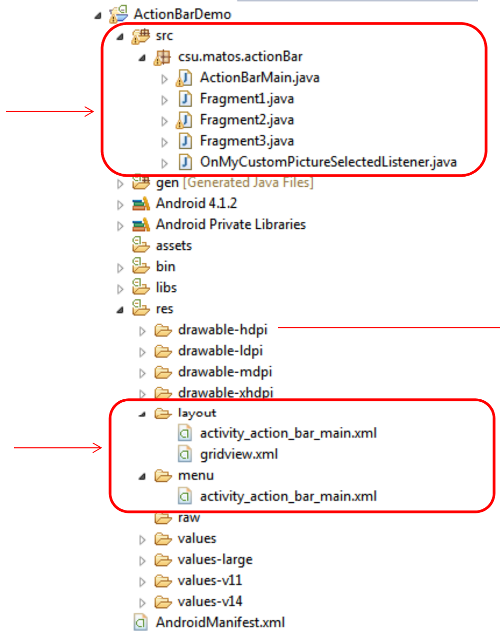


Show a high-quality version of the picture selected in Fragment1

Image selected by the user in Fragment1

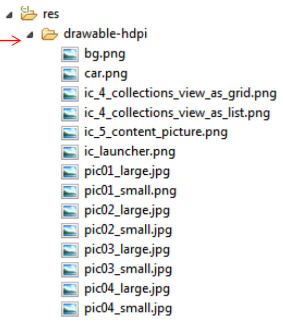
Display the Menu/Overflow Options

Example2: Using Fragments and ActionBars



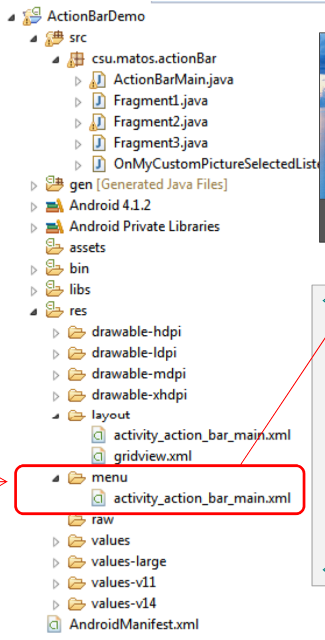
```
ActionBarDemo
  src
    csu.matos.actionBar
      ActionBarMain.java
      Fragment1.java
      Fragment2.java
      Fragment3.java
      OnMyCustomPictureSelectedListener.java
  gen [Generated Java Files]
  Android 4.1.2
  Android Private Libraries
  assets
  bin
  libs
  res
    drawable-hdpi
    drawable-ldpi
    drawable-mdpi
    drawable-xhdpi
    layout
      activity_action_bar_main.xml
      gridview.xml
    menu
      activity_action_bar_main.xml
    raw
    values
    values-large
    values-v11
    values-v14
  AndroidManifest.xml
```

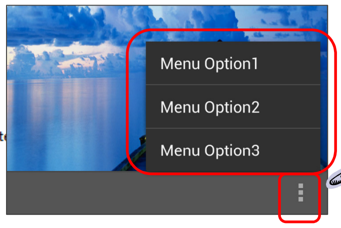
Eclipse's Package Explorer View of Example2



```
res
  drawable-hdpi
    bg.png
    car.png
    ic_4_collections_view_as_grid.png
    ic_4_collections_view_as_list.png
    ic_5_content_picture.png
    ic_launcher.png
    pic01_large.jpg
    pic01_small.png
    pic02_large.jpg
    pic02_small.jpg
    pic03_large.jpg
    pic03_small.jpg
    pic04_large.jpg
    pic04_small.jpg
```

Example2: Using Fragments and ActionBars





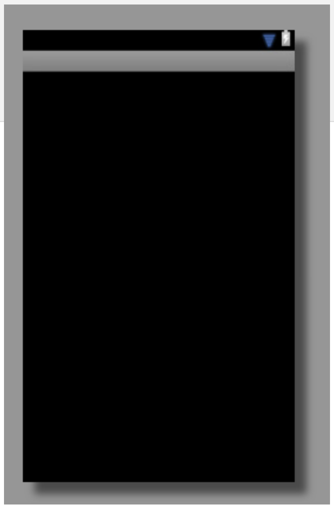
Clicking on the Menu button inflates the XML menu specification

```
<menu xmlns:android="http://schemas.android.com/apk/res/android" >

    <item
        android:id="@+id/menu_settings"
        android:orderInCategory="100"
        android:title="Menu Option1" />
  <item
        android:id="@+id/menu_settings"
        android:orderInCategory="110"
        android:title="Menu Option2" />
  <item
        android:id="@+id/menu_settings"
        android:orderInCategory="120"
        android:title="Menu Option3" />
</menu>
```

Example2: Using Fragments and ActionBars

Example 2 – XML Layout: activity_action_bar_main.xml

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/mainLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

</FrameLayout>
```

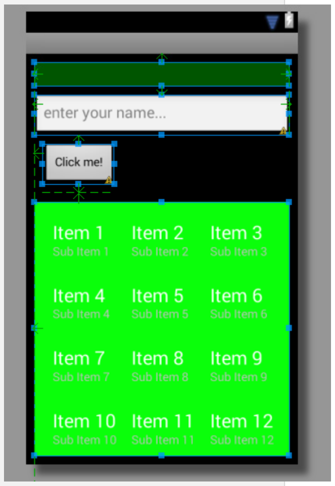


mainLayout provides an empty space in which fragments will place their own GUIs. Choose a FrameLayout or any other type here.

Example2: Using Fragments and ActionBars

Example 2 – XML Layout: gridview.xml 1 of 2

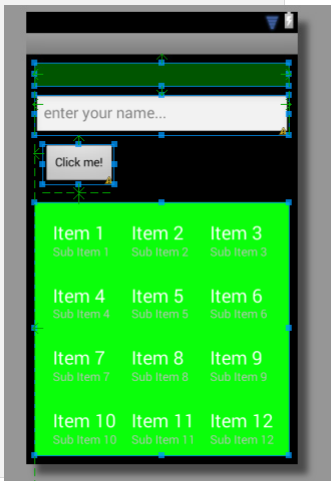
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/LinearLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="3dp" >

    <TextView
        android:id="@+id/editText1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="5dp"
        android:background="#ff005500"
        android:textColor="#ffffffff"
        android:textSize="24sp" />

    <EditText
        android:id="@+id/editText2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:ems="10"
        android:hint="enter your name..."
        android:inputType="textCapWords" />
```



Example2: Using Fragments and ActionBars

Example 2 – XML Layout: gridview.xml 2 of 2

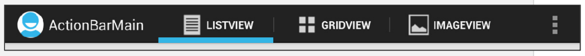
```
    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:text=" Click me! " />

    <GridView
        android:id="@+id/mainGrid"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_margin="10dip"
        android:background="#ff00ff00"
        android:horizontalSpacing="10dp"
        android:numColumns="3"
        android:padding="10dip"
        android:stretchMode="columnWidth"
        android:verticalSpacing="10dp" >
    </GridView>

</LinearLayout>
```



Example2: Using Fragments and ActionBars

MainActivity: ActionBarMain.java 1 of 7

```
public class ActionBarMain extends Activity implements
                                            TabListener,
❶                                           OnMyCustomPictureSelectedListener {

   // this is the row# picked up in Fragment1(ListView)
   Integer selectedRow = 0;
   // host layout where fragments are displayed
   FrameLayout mainLayout;

   // fragment objects
   FragmentTransaction fragTransactMgr = null;
   Fragment currentFragment;

   // tab's captions
   private final String CAPTION1 = "ListView";
   private final String CAPTION2 = "GridView";
   private final String CAPTION3 = "ImageView";
   private final String[] CAPTIONS = new String[] {CAPTION1, CAPTION2, CAPTION3};

   private final String LAST_SELECTED_TAB_INDEX = "LAST_SELECTED_TAB_INDEX";
   private final String SELECTED_ROW = "SELECTED_ROW";

    // use it to remember last tab-index selected by the user
   int lastTabNumber = 0;
```



Example2: Using Fragments and ActionBars

MainActivity: ActionBarMain.java 2 of 7

```
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);

      setContentView(R.layout.activity_action_bar_main);
      try {
         mainLayout = (FrameLayout) findViewById(R.id.mainLayout);

         fragTransactMgr = getFragmentManager().beginTransaction();
❷        ActionBar bar = getActionBar();

         // create fresh tabs adding caption and icon
         bar.addTab(bar.newTab().setText(CAPTION1)
               .setIcon(R.drawable.ic_4_collections_view_as_list)
               .setTabListener(this));

         bar.addTab(bar.newTab().setText(CAPTION2)
               .setIcon(R.drawable.ic_4_collections_view_as_grid)
               .setTabListener(this));

❸        bar.addTab(bar.newTab().setText(CAPTION3)
               .setIcon(R.drawable.ic_5_content_picture)
               .setTabListener(this));

         bar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);
         bar.setDisplayShowHomeEnabled(true);
         bar.setDisplayShowTitleEnabled(true);

         fragTransactMgr.commit();
```

Example2: Using Fragments and ActionBars

MainActivity: ActionBarMain.java 3 of 7

```
        // dealing with device rotation & re-starting
        lastTabNumber = 0;
        selectedRow = 0;
        // if needed bring back previous state info including selected row
        // and last selected tab index, then destroy the bundle
4       if (savedInstanceState != null) {
           lastTabNumber = savedInstanceState.getInt(LAST_SELECTED_TAB_INDEX, 0);
           selectedRow = savedInstanceState.getInt(SELECTED_ROW, 0);
           savedInstanceState = null;
        }
        bar.setSelectedNavigationItem(lastTabNumber);
        bar.show();

     } catch (Exception e) {
        e.getMessage();
     }
  }// onCreate

  @Override
  protected void onSaveInstanceState(Bundle outState) {
     super.onSaveInstanceState(outState);
5    // close to the end (phone was rotated or app was terminated)
     // Save the index of the currently selected tab
     // and the selected row picked up from the listview
     int activeTab = getActionBar().getSelectedTab().getPosition();
     outState.putInt(LAST_SELECTED_TAB_INDEX, activeTab);
     outState.putInt(SELECTED_ROW, selectedRow);
  }
```

Example2: Using Fragments and ActionBars

MainActivity: ActionBarMain.java 4 of 7

```
  @Override
  public boolean onCreateOptionsMenu(Menu menu) {
     // puff the XML menu definition (it shows a few entries)
6    getMenuInflater().inflate(R.menu.activity_action_bar_main, menu);
     return true;
  }

  @Override
  public boolean onOptionsItemSelected(MenuItem item) {
     // let user know about the selectedd menu option
7    Toast.makeText(this, "Option selected: " + item.getTitle(),
           Toast.LENGTH_SHORT).show();
     return true;
  }

  @Override
  public void onTabReselected(Tab tab, FragmentTransaction ft) {
     //TODO - nothing to do, needed by the interface
  }

  @Override
  public void onTabSelected(Tab tab, FragmentTransaction ft) {
     // the user has clicked on a tab - make the corresponding
     // fragment do its job(show a ListView, GridView, ImageView)
     // instantiate a new Fragment, its argument is the
     // selectedRow value. The argument must go in a bundle
```

Example2: Using Fragments and ActionBars

MainActivity: ActionBarMain.java 5 of 7

```
      // create the appropriate fragment based on the tag
8     String tag = (String) tab.getText();
      if (tag.equals(CAPTION1)) {
         currentFragment = addArgsToFragment(new Fragment1(), selectedRow);
      } else if (tag.equals(CAPTION2)) {
         currentFragment = addArgsToFragment(new Fragment2(), selectedRow);
      } else if (tag.equals(CAPTION3)) {
         currentFragment = addArgsToFragment(new Fragment3(), selectedRow);
      }
      // let new fragment be attached to the main GUI
      executeFragment(currentFragment, ft, tag);

   }

   public void executeFragment(Fragment fragment, FragmentTransaction ft, String tag) {
      try {
         // replace any fragment currently attached to the GUI (if needed)
         // with the fragment here provided (identified by tag)
9        ft.replace(mainLayout.getId(), fragment, tag);

      } catch (Exception e) {
         Log.e("ERROR-executeFragment", e.getMessage());
      }
   }// executeFragment

   @Override
   public void onTabUnselected(Tab tab, FragmentTransaction ft) {
      //TODO - nothing to do, needed by the interface
   }
```

Example2: Using Fragments and ActionBars

MainActivity: ActionBarMain.java 7 of 7

```
   // Accept a fragment, and simple arguments, put those arguments
   // into a bundle and bind the fragment with the bundle (only one here).
   // This approach is required for apps running SDK4.x

   public static final <E extends Fragment> E addArgsToFragment (E fragment,
10       int selectedRow) {
      // E represents: Fragment1, Fragment2, or Fragment3 classes
      Bundle bundle = new Bundle();
      bundle.putInt("selectedRow", selectedRow);
      fragment.setArguments(bundle);
      return fragment;
   }


   // this method supports fragment-to-Activity communication. When
   // a row in Fragment1 is selected, this custom callBack is invoked.
   // It updates the valued of 'selectedRow' held in the main activity.

   @Override
11 public void onMyCustomPictureSelected(Integer selectedRow) {
      // as soon as the user picks a row in fragment1,
      // its value (position in the list) is saved here
      this.selectedRow = selectedRow;

   }

}// class
```

Example2: Using Fragments and ActionBars

MainActivity: ActionBarMain.java

COMMENTS

1. The class ActionBarMain (MainActivity) implements two interfaces: **TabListener** and a custom callback mechanism here named **MyCustomPictureSelectedListener**. The first allows the user to trigger a new action after clicking on a tab (or Menu button), the second allows the main activity to hear messages sent by running Fragments.

2. The `getActionBar()` method returns a handle to the GUI ActionBar.

3. The ActionBar is populated, tabs are created, each receiving a caption, an icon, and a listener. Finally you choose the navigation mode (TAB clicking or LIST scrolling), as well as the displaying of a title and icon for the app (HomeEnable, TitleEnable). These actions are processed inside a transaction framework (BeginTransaction, commit)

Example2: Using Fragments and ActionBars

MainActivity: ActionBarMain.java

COMMENTS

4. If this is a fresh execution the state bundle (savedInstanceState) does not exist, and the control variables selectedRow, and lastSelectedTab are set to zero. Otherwise their values are extracted from the bundle. We do this to cope with hardware changes, such as the rotation of the device.

5. Before the app is stopped, we save critical data into a bundle to gracefully recover if necessary. The variables `selectedRow` (a choice from the ListView) and `activeTab` (last tab clicked by the user) are recorded for potential use in the future (onCreate will attempt the reading of those values).

6. An XML menu specification is inflated to provide additional functionality. This is also called “Overflow” options (three dots on the UI either on top or bottom of the screen).

Example2: Using Fragments and ActionBars

MainActivity: ActionBarMain.java

COMMENTS

7. A brief message is displayed after a menu option is chosen.
8. Each fragment class requires its own set of arguments to operate. The convention is to put all those arguments into a single bundle. This approach makes recovery more manageable (better than several overridden constructors with various sets of arguments).
9. When a fragment is executed it asks the FragmentManager to remove from the host layout any other fragment currently occupying the indicated portion of GUI. Afterward, control is transferred to the calling fragment with becomes visible and active
10. The method `addArgsToFragment` accepts a newly created fragment (three possible types) and its parameter `selectedRow`. It creates a bundle, drops the argument inside, and instructs the system (`.setArguments(bundle)`) to wrap the fragment an bundle together.

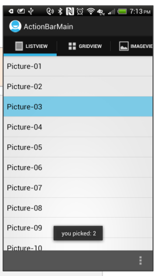
Example2: Using Fragments and ActionBars

MainActivity: ActionBarMain.java

COMMENTS

11. The main activity implements the custom interface `MyCustomPictureSelectedListener`.
    This interface –similar to onClickListener- has only one method: `onMyCustomPictureSelected` which is called by Fragment1 when the user chooses a row from a ListView. The chosen row position is passed in the argument: `selectedRow`.

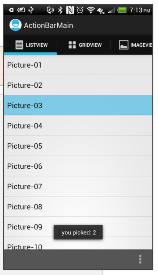
Example2: Using Fragments and ActionBars



FRAGMENTS: Fragment1.java 1of 2

```
// this fragment shows a ListView (offering options to select a picture)
public class Fragment1 extends Fragment {

1   OnMyCustomPictureSelectedListener mListener;

    private String items[] = {
          "Picture-01","Picture-02","Picture-03","Picture-04","Picture-05",
2         "Picture-06","Picture-07","Picture-08","Picture-09","Picture-10",
          "Picture-11","Picture-12","Picture-13","Picture-14","Picture-15" };

    @Override
    public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       // nothing here - see onCreateView
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
          Bundle savedInstanceState) {

       // instead of an XML spec, this view is created with code
       Context context = getActivity();
       ListView listView = new ListView(context);

       ArrayAdapter<String> array = new ArrayAdapter<String>(context,
            android.R.layout.simple_list_item_1, items);
3
       listView.setAdapter(array);
```

Example2: Using Fragments and ActionBars



FRAGMENTS: Fragment1.java 2 of 2

```
       listView.setOnItemClickListener(new OnItemClickListener() {

          @Override
          public void onItemClick(AdapterView<?> parent, View v,
                int position, long id) {

             Toast.makeText(getActivity(), " you picked: " + position, 1)
                   .show();
             // Send the event and clicked item's row ID to the host activity
4            mListener.onMyCustomPictureSelected(position);
          }
       });

       return listView;
    }// onCreateView

    // making sure the MainActivity has implemented the listener
    // and can accept our callback messaging
     @Override
     public void onAttach(Activity activity) {
         super.onAttach(activity);
         try {
5            mListener = (OnMyCustomPictureSelectedListener) activity;
         } catch (ClassCastException e) {
             throw new ClassCastException(activity.toString()
                 + " must implement OnArticleSelectedListener");
         }
     }//onAttach
}//class
```

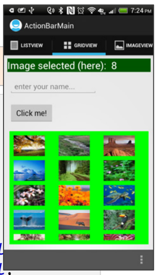
Example2: Using Fragments and ActionBars

FRAGMENTS: Fragment1.java

COMMENTS: This fragment shows a ListView

1. The custom listener defined by the user's supplied interface is needed in order to pass data to the host main activity.
2. A list of String-type values will supply input to the ListView.
3. The ListView and adapter are bound. The adapter uses a pre-defined android layout for the list, and the data items mentioned above.
4. When the user clicks on a ListView row the local ItemClickListener is activated. A brief message is displayed announcing the row selection and the method `mListener.onMyCustomPictureSelected(position)` is invoked to tell the host main activity of the `position` chosen.
5. Before the fragment 's view is created the onAttach method is called. Here we check the host activity has implemented the listener, otherwise the fragment –having no way to pass data to the activity- ends with an error.

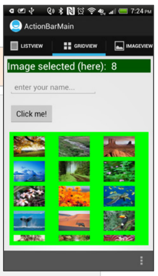
Example2: Using Fragments and ActionBars



FRAGMENTS: Fragment2.java 1 of 4

```
public class Fragment2 extends Fragment implements OnItemClickListener  {
   TextView txtMsg;
   EditText txtName;
   GridView gridview;

   Button btnGo;
   Integer[] smallImages = {
         R.drawable.pic01_small, R.drawable.pic02_small, R.drawable.pic03_small
         R.drawable.pic04_small, R.drawable.pic05_small, R.drawable.pic06_small,
         R.drawable.pic07_small, R.drawable.pic08_small, R.drawable.pic09_small,
         R.drawable.pic10_small, R.drawable.pic11_small, R.drawable.pic12_small,
         R.drawable.pic13_small, R.drawable.pic14_small, R.drawable.pic15_small };
   Integer selectedRow;


   @Override
   public void onCreate(Bundle savedInstanceState) {
1 →   super.onCreate(savedInstanceState);
      this.selectedRow = getArguments().getInt("selectedRow",0);

   }

      // this view is inflated using an XML layout file
   @Override
   public View onCreateView(LayoutInflater inflater,
         ViewGroup container,
         Bundle savedInstanceState) {
```

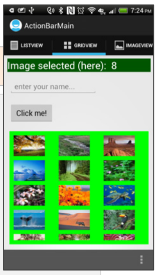
Example2: Using Fragments and ActionBars



FRAGMENTS: **Fragment2.java** 2 of 4

```
2 →     View view = inflater.inflate(R.layout.gridview, null);

        gridview = (GridView) view.findViewById(R.id.mainGrid);
        txtMsg = (TextView) view.findViewById(R.id.editText1);
        txtName = (EditText) view.findViewById(R.id.editText2);

        btnGo = (Button) view.findViewById(R.id.button1);

3 →     btnGo.setOnClickListener(new OnClickListener() {

           @Override
           public void onClick(View v) {
              String text = txtName.getText().toString();
              text += " " + new Date().toString();
              txtName.setText(text);
              hideVirtualKeyboard();
           }
        });

        Adapter myadapter = new Adapter( getActivity() );
4 →     gridview.setAdapter(myadapter);
        gridview.setOnItemClickListener(this);

        // tell here what picture was already selected in fragment1
        String text = "User selected from Listview (Fragment1)\nrow: "
                      + selectedRow;
        Toast.makeText(getActivity(), text, Toast.LENGTH_SHORT).show();

5 →     return view;
     }//onCreate
```

Example2: Using Fragments and ActionBars

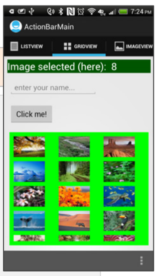


FRAGMENTS: **Fragment2.java** 3 of 4

```
     public void hideVirtualKeyboard() {
         Context context = getActivity().getApplicationContext();
        ((InputMethodManager) context
               .getSystemService(Activity.INPUT_METHOD_SERVICE))
                 .toggleSoftInput(InputMethodManager.HIDE_IMPLICIT_ONLY, 0);
        }

     // $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
6 →  private class Adapter extends BaseAdapter {
        Context ctx;
        public Adapter(Context ctx){
           this.ctx = ctx;
        }
        @Override
        public int getCount() {
           return smallImages.length;
        }

        @Override
        public Object getItem(int position) {
           return null;
        }

        @Override
        public long getItemId(int position) {
           // TODO Auto-generated method stub
           return 0;
        }
```

Example2: Using Fragments and ActionBars

FRAGMENTS: **Fragment2.java** 4 of 4



```
      @Override
7     public View getView(int position,
              View convertView,
              ViewGroup parent) {
          ImageView image;
          if (convertView == null) {
              image = new ImageView(Fragment2.this.getActivity());
              image.setLayoutParams(new GridView.LayoutParams(150, 100));
              image.setScaleType(ScaleType.FIT_XY);
              convertView = image;
          } else {
              image = (ImageView) convertView;
          }

          image.setImageResource(smallImages[position]);
          return image;
      }

  }//ViewAdapter  $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

  //TODO: repeat strategy used in fragment1, when user clicks
  //    on an image you let the callback method in main activity
  //    know what image (position) has been selected

  @Override
  public void onItemClick(AdapterView<?> parent, View v, int position, long id) {
      txtMsg.setText("Image selected (here):  " + position);

  }//onItemClick
}//Activity
```

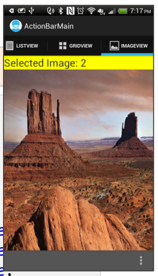
Example2: Using Fragments and ActionBars

FRAGMENTS: **Fragment2.java**

COMMENTS: This fragment shows a GridView

1. The parameter `selectedRow` is extracted from the incoming argument-bundle.
2. The user supplied **file res/layout/gridview.xml** identified as `R.layout.`*`gridview`* is inflated to provide a visual representation of this fragment.
3. The GUI components in the view (a TextView, EditText, Button and GridView) are wired-up to the fragment.
4. A custom adapter –capable of formatting thumbnails- and the GridView are bound together.
5. Once all the fragment's components are created and populated, the entire view is returned. This view is subsequently attached and displayed in the host GUI container.
6. A custom DataAdapter is defined to deal with the GridView images.
7. The method `getView()` –defined in the custom data adapter- provides details about the making/placing of the thumbnail images used to populate the GridView.

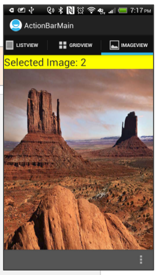
Example2: Using Fragments and ActionBars



FRAGMENTS: **Fragment3.java** 1 of 2

```
public class Fragment3 extends Fragment {

   private Integer selectedRow;

   Integer[] largeImages = {
         R.drawable.pic01_large, R.drawable.pic02_large, R.drawable.pic03_large,
         R.drawable.pic04_large, R.drawable.pic05_large, R.drawable.pic06_large,
         R.drawable.pic07_large, R.drawable.pic08_large, R.drawable.pic09_large,
         R.drawable.pic10_large, R.drawable.pic11_large, R.drawable.pic12_large,
         R.drawable.pic13_large, R.drawable.pic14_large, R.drawable.pic15_large };


   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);

1 →   // this is the index of the picture to be displayed here
      this.selectedRow = getArguments().getInt("selectedRow");

   }

   // This GUI is entirely created by code. It consists of a
   // LinearLayout holding a TextView and an ImageView
   // showing a 'high-quality' version of the selected image.
   @Override
   public View onCreateView(LayoutInflater inflater,
         ViewGroup container,
         Bundle savedInstanceState) {
```

Example2: Using Fragments and ActionBars



FRAGMENTS: **Fragment3.java** 1 of 2

```
2 →   LinearLayout linearlayout = new LinearLayout(getActivity());
      linearlayout.setOrientation(LinearLayout.VERTICAL);

3 →   TextView txtMsg2 = new TextView(getActivity());
      txtMsg2.setBackgroundColor(0xffffff00);
      txtMsg2.setTextSize(25);
      txtMsg2.setText("Selected Image: " + selectedRow);

4 →   ImageView image = new ImageView(getActivity());
      image.setLayoutParams(new RelativeLayout.LayoutParams(
                        LayoutParams.MATCH_PARENT, LayoutParams.MATCH_PARENT));
      image.setBackgroundResource( largeImages[selectedRow] );

      linearlayout.addView(txtMsg2);
5 →   linearlayout.addView(image);

      return linearlayout;
   }
}
```

Example2: Using Fragments and ActionBars

FRAGMENTS: Fragment3.java

COMMENTS: This fragment shows an ImageView

1. The parameter `selectedRow` is extracted from the incoming argument-bundle.
2. The fragment defines a LinearLayout on which its UI components, a TextView and ImageView (see bubbles 3 & 4) will be included.
3. TextView is created, formatted and populated.
4. ImageView is created displaying the image selected in Fragment1.
5. The TextView and ImageView are added to the locally defined LinearLayout. The assembled fragment's view is returned for its attachment to the host UI.

Example2: Using Fragments and ActionBars

INTERFACE: MyCustomPictureSelectedListener

```
package csu.matos.actionBar;


// Note: The MainActivity must implement this interface !!!
// Used to tell the MainActivity what row from ListView the user
// has selected when interacting with Fragment1 (this is
// functionally equivalent to an onClickLister)

public interface OnMyCustomPictureSelectedListener {

    public void onMyCustomPictureSelected(Integer selectedRow);

}
```

TabHost, Fragments, ActionBar

Questions ?