Lesson 6

ActionBar, Fragments and the TabHost

Victor Matos
Cleveland State University

Portions of this page are reproduced from work created and shared by Google and used according to terms described in the Creative Commons 3.0 Attribution License.
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 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

TabHost

TabWidget

Tab1
TabSpec

Tab2
TabSpec

Tab3
TabSpec

Content

FrameLayout

Outline
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

```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="#ffffff"
    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
<?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
<?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>
```
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 tabhost = (TabHost) findViewById(android.R.id.tabhost);
// 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 tabhost = (TabHost) findViewById(android.R.id.tabhost);

tabhost.setOnTabChangedListener(new OnTabChangeListener() {
    @Override
    public void onTabChanged(String tagId) {
        // do something useful with the selected screen
        String text = "I'm currently on: " + tagId + "index: " + 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();
    }
});
```
Example 1: TabHost Selection Widget

TabHostDemo – ActivityMain Class

```java
// 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);
}
```
HINT: Adding Icons to Tabs

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

```java
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
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.
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
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, more than one of them could be visible and active at the same time.

• Fragments behave as separate threads each running its own 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.

• First introduced in the Honeycomb SDK (API 11)
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.

**onCreate()** Called when creating the fragment. Generally used for initializing those essential components that you want to retain when the fragment is paused or stopped, then resumed.

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

**onPause()** Called when the user leaves the fragment. Here you should commit state changes that are needed in case the fragment is re-executed.

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

How to use Fragments to support GUI design?

1. Decide what is the overall organization of your activity’s GUI. Include in your decision factors such as: screen size (small/large), rendition mode (portrait/landscape), device events (rotation, interruption).

2. Choose many fragments you want to use in the activity (Typically, one for landscape mode and one for portrait mode)

3. For each of your fragments create new classes extending the `Fragment` class.

4. Override the callback methods of your new fragments. Usually the `onCreate`, `onCreateView` and `onPause` are needed.

5. For each fragment, you will need to provide an XML layout file specifying how the view returned by its corresponding `onCreateView` method should be displayed.
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.
Example 2: Using Fragments and ActionBars

User makes a selection when using Fragment 1.

The row’s position is sent back to the main activity’s listener (in this example row 3 is selected).
Example 2: Using Fragments and ActionBars

Fragment 2

- 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 Fragment 1
  
  User selected in fragment1 row number ....
Example 2: Using Fragments and ActionBars

Show a high-quality version of the picture selected in Fragment 1

Image selected by the user in Fragment 1

Display the Menu/Overflow Options
Example 2: Using Fragments and ActionBars

Eclipse’s Package Explorer View of Example 2
Example 2: Using Fragments and ActionBars

Clicking on the Menu button inflates the XML menu specification.

```xml
<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>
```
Example 2: Using Fragments and ActionBars

Example 2 – XML Layout:  activity_action_bar_main.xml

```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"/>
```

`mainLayout` provides an empty space in which fragments will place their own GUIs. Choose a FrameLayout or any other type here.
Example 2 – XML Layout:  

<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"/>

Example 2:  Using Fragments and ActionBars
Example 2 – XML Layout:  gridview.xml  2 of 2

```xml
<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="10dp"
    android:background="#ff00ff00"
    android:horizontalSpacing="10dp"
    android:numColumns="3"
    android:padding="10dp"
    android:stretchMode="columnWidth"
    android:verticalSpacing="10dp" />
</GridView>
</LinearLayout>
```
public class ActionBarMain extends Activity implements TabListener,
        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;
Example 2: 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();
        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));

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

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // let user know about the selected menu option
    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
```

---

**MainActivity:** ActionBarMain.java

---

Example2: Using Fragments and ActionBars
// create the appropriate fragment based on the tag
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)
        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
}
// 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, 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
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;
}
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).
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).
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.
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.
// this fragment shows a ListView (offering options to select a picture)
public class Fragment1 extends Fragment {

    OnMyCustomPictureSelectedListener mListener;

    private String items[] = {
        "Picture-01", "Picture-02", "Picture-03", "Picture-04", "Picture-05",
        "Picture-06", "Picture-07", "Picture-08", "Picture-09", "Picture-10",

    @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);

        listView.setAdapter(array);
    }
}
Example 2: Using Fragments and ActionBars

FRAGMENTS: Fragment1.java 2 of 2

```java
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
        mListener.onMyCustomPictureSelected(position);
    }
});
return listView;
} // onCreateView

@Override
public void onAttach(Activity activity) {
    super.onAttach(activity);
    // Activities containing this fragment must implement its callbacks.
    if (!(activity instanceof OnMyCustomPictureSelectedListener)) {
        throw new IllegalStateException("Activity must implement OnMyCustomPictureSelectedListener.");
    }
    mListener = (OnMyCustomPictureSelectedListener) activity;
} // onAttach
} // class
```
Example2: Using Fragments and ActionBars

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 \texttt{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’s callbacks, otherwise the fragment –having no way to pass data to the activity- ends with an error.
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) {
        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) {
        return null;  // return null to indicate the view was inflated
    }
}
Example 2: Using Fragments and ActionBars

FRAGMENTS: Fragment2.java 2 of 4

```java
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);

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());
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();

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

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;
    }
}
Example 2: Using Fragments and ActionBars

FRAGMENTS: Fragment2.java 4 of 4

```java
@Override
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;
}
```
Example2: Using Fragments and ActionBars

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.
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);

        // 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) {
        //FRAGMENTS:  Fragment3.java  1 of 2

        return null;
    }
}
LinearLayout linearlayout = new LinearLayout(getActivity());
linearlayout.setOrientation(LinearLayout.VERTICAL);

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

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

linearlayout.addView(txtMsg2);
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.
Example 2: Using Fragments and ActionBars

**INTERFACE:** MyCustomPictureSelectedListener

```java
default 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 onClickListener)

default public interface OnMyCustomPictureSelectedListener {
    
    default public void onMyCustomPictureSelected(Integer selectedRow);

}
```
1. Although relatively new, Android tablets are becoming a preferred choice for many users. Their acceptance goes in hand with the continuous decrease in sales of laptop computers.

2. App designers now need to contend with deploying their apps in devices of very different levels of resolution. Clearly tablets can display more visual elements than handsets, however users should feel the same experience in dealing with the app regardless of the actual device on which it runs.

3. Fragments can be used to alleviate this problem. The multi-pane or Master/Detail model has been included to the set of basic templates created by the ADT.

The next example discusses a plain Master/Detail app generated by the Eclipse-ADT wizard.
Fragments

Multi-Pane Apps

An example of how two UI modules defined by fragments can be combined into one activity for a tablet design, but separated for a handset design

Example 3: A Multi-pane App

Master/Detail App

A two-panes app shows in the first pane a list of items from which a selection is made. The details associated to the selected choice are shown in the second pane. The app should be deployed on tablets and handsets.
Example 3: A Multi-pane App

Master/Detail App - Screenshots

Images from the two-pane app’s execution are shown when running on a tablet and a handset.
Example 3: A Multi-pane App

Master/Detail App – Creating the App

This solution is made using the **Master/Detail Flow** wizard (Eclipse-ADT)

![Image of Eclipse-ADT interface showing Master/Detail Flow creation]

**Warning:** Minimum SDK is API-11
Example 3: A Multi-pane App

Master/Detail App - Project Structure

This solution is made using the Master/Detail Flow wizard (Eclipse-ADT)

ItemListActivity is the main activity.

“It represents a list of Items. This activity has different presentations for handset and tablet-size devices. On handsets, the activity presents a list of items, which when touched, lead to a ItemDetailActivity representing item details. On tablets, the activity presents the list of items and item details side-by-side using two vertical panes.

The activity makes heavy use of fragments. The list of items is a ItemListFragment and the item details (if present) is a ItemDetailFragment.

This activity also implements the required ItemListFragment.Callbacks interface to listen for item selections.”
Example 3: A Multi-pane App

Master/Detail App – XML Layouts

<!-- activity_item_detail.xml -->

```xml
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/item_detail_container"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".ItemDetailActivity"
    tools:ignore="MergeRootFrame" />
```

<!-- fragment_item_detail.xml -->

```xml
<TextView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/item_detail"
    style="?android:attr/textAppearanceLarge"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    android:textIsSelectable="true"
    tools:context=".ItemDetailFragment"/>
```
Example 3: A Multi-pane App

Master/Detail App – XML Layouts

```
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
          xmlns:tools="http://schemas.android.com/tools"
          android:id="@+id/item_list"
          android:name="com.example.fragmentdemo3.ItemListFragment"
          android:layout_width="match_parent"
          android:layout_height="match_parent"
          android:layout_marginLeft="16dp"
          android:layout_marginRight="16dp"
          tools:context="ItemListActivity"
          tools:layout="@android:Layout/list_content"/>
```
Example 3: A Multi-pane App

Master/Detail App – XML Layouts

```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:layout_marginLeft="16dp"
    android:layout_marginRight="16dp"
    android:baselineAligned="false"
    android:divider="?android:attr/dividerHorizontal"
    android:orientation="horizontal"
    android:showDividers="middle"
    tools:context=".ItemListActivity">
    <fragment
        android:id="@+id/item_list"
        android:name="com.example.fragmentdemo3.ItemListFragment"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1"
        tools:layout="@android:layout/list_content" />
    <FrameLayout
        android:id="@+id/item_detail_container"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="3" />
</LinearLayout>
```
Example 3: A Multi-pane App

Main: ItemListActivity.java

```java
package com.example.fragmentdemo3;

import android.content.Intent;
import android.os.Bundle;
import android.support.v4.app.FragmentActivity;

/**
 * An activity representing a list of Items. This activity has different
 * presentations for handset and tablet-size devices. On handsets, the activity
 * presents a list of items, which when touched, lead to a
 * {link ItemDetailActivity} representing item details. On tablets, the
 * activity presents the list of items and item details side-by-side using two
 * vertical panes.
 * <p>
 * The activity makes heavy use of fragments. The list of items is a
 * {link ItemListFragment} and the item details (if present) is a
 * {link ItemDetailFragment}.
 * <p>
 * This activity also implements the required {link ItemListFragment_Callbacks}
 * interface to listen for item selections.
 */

public class ItemListActivity extends FragmentActivity
    implements ItemListFragment.Callbacks {
    /**
     * Whether or not the activity is in two-pane mode, i.e. running on a tablet
     * device.
     */
    private boolean mTwoPane;
```
Example 3: A Multi-pane App

Main: ItemListActivity.java

```java
@override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_item_list);

    if (findViewById(R.id.item_detail_container) != null) {
        // The detail container view will be present only in the
        // large-screen layouts (res/values-large and
        // res/values-sw600dp). If this view is present, then the
        // activity should be in two-pane mode.
        mTwoPane = true;

        // In two-pane mode, list items should be given the
        // 'activated' state when touched.
        ((ItemListFragment) getSupportFragmentManager().
            findFragmentById(R.id.item_list)).
            setActivateOnItemClick(true);
    }

    // TODO: If exposing deep links into your app, handle intents here.
}
```
/**
 * Callback method from {@link ItemListFragment.Callables} indicating that
 * the item with the given ID was selected.
 */

@Override
public void onItemSelected(String id) {
    if (mTwoPane) {
        // In two-pane mode, show the detail view in this activity by
        // adding or replacing the detail fragment using a
        // fragment transaction.
        Bundle arguments = new Bundle();
        arguments.putString(ItemDetailFragment.ARG_ITEM_ID, id);
        ItemDetailFragment fragment = new ItemDetailFragment();
        fragment.setArguments(arguments);
        getSupportFragmentManager().beginTransaction().replace(R.id.
                item_detail_container, fragment)
                .commit();
    } else {
        // In single-pane mode, simply start the detail activity
        // for the selected item ID.
        Intent detailIntent = new Intent(this, ItemDetailActivity.class);
        detailIntent.putExtra(ItemDetailFragment.ARG_ITEM_ID, id);
        startActivity(detailIntent);
    }
}
package com.example.fragmentdemo3;

import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;
import com.example.fragmentdemo3.dummy.DummyContent;

/**<p>A fragment representing a single Item detail screen. This fragment is either contained in a {@link ItemListActivity} in two-pane mode (on tablets) or a {@link ItemDetailActivity} on handsets.</p>*/
public class ItemDetailFragment extends Fragment {
    /**
     * The fragment argument representing the item ID that this fragment represents.
     */
    public static final String ARG_ITEM_ID = "item_id";

    /**
     * The dummy content this fragment is presenting.
     */
    private DummyContent.DummyItem mItem;
public ItemDetailFragment() {
}

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

    if (getArguments().containsKey(ARG_ITEM_ID)) {
        // Load the dummy content specified by the fragment arguments. In a real-world
        // scenario, use a Loader to load content from a content provider.
        mItem = DummyContent.ITEM_MAP.get(getArguments().getString(ARG_ITEM_ID));
    }
}
```java
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {

    View rootView = inflater.inflate(R.layout.fragment_item_detail,
            container, false);

    // Show the dummy content as text in a TextView.
    if (mItem != null) {
        ((TextView) rootView.findViewById(R.id.item_detail))
                .setText(mItem.content);
    }

    return rootView;
}
```
Example 3: A Multi-pane App

Fragment2: ItemDetailActivity.java

```java
package com.example.fragmentdemo3;

import android.content.Intent;
import android.os.Bundle;
import android.support.v4.app.FragmentActivity;
import android.support.v4.app.NavUtils;
import android.view.MenuItem;

/* An activity representing a single Item detail screen. This activity is only
 * used on handset devices. On tablet-size devices, item details are presented
 * side-by-side with a list of items in a {@link ItemListActivity}.
 * <p>
 * This activity is mostly just a 'shell' activity containing nothing more
 * than a {@link ItemDetailFragment}.
 */
public class ItemDetailActivity extends FragmentActivity {

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

        // Show the Up button in the action bar.
        getSupportActionBar().setDisplayHomeAsUpEnabled(true);
    }
}
```
// savedInstanceState is non-null when there is fragment state
// saved from previous configurations of this activity
// (e.g. when rotating the screen from portrait to landscape).
// In this case, the fragment will automatically be re-added
// to its container so we don't need to manually add it.
// For more information, see the Fragments API guide at:

if (savedInstanceState == null) {
    // Create the detail fragment and add it to the activity
    // using a fragment transaction.
    Bundle arguments = new Bundle();
    arguments.putString(ItemDetailFragment.ARG_ITEM_ID, getIntent().getStringExtra(ItemDetailFragment.ARG_ITEM_ID));

    ItemDetailFragment fragment = new ItemDetailFragment();
    fragment.setArguments(arguments);

    getSupportFragmentManager().beginTransaction().add(R.id.item_detail_container, fragment).commit();
}
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
    case android.R.id.home:
        // This ID represents the Home or Up button. In the case of this
        // activity, the Up button is shown. Use NavUtils to allow users
        // to navigate up one level in the application structure. For
        // more details, see the Navigation pattern on Android Design:
        //
        // http://developer.android.com/design/patterns/navigation.html#up-vs-back
        //
        NavUtils.navigateUpTo(this,
                          new Intent(this, ItemListActivity.class));
        return true;
    }
    return super.onOptionsItemSelected(item);
}
package com.example.fragmentdemo3;

import android.app.Activity;
import android.os.Bundle;
import android.support.v4.app.ListFragment;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import com.example.fragmentdemo3.dummy.DummyContent;

/**
 * A list fragment representing a list of Items. This fragment also supports
 * tablet devices by allowing list items to be given an 'activated' state upon
 * selection. This helps indicate which item is currently being viewed in a
 * {@link ItemDetailFragment}.
 * <p>
 * Activities containing this fragment MUST implement the {@link Callbacks}
 * interface.
 */

public class ItemListFragment extends ListFragment {

    /**
     * The serialization (saved instance state) Bundle key representing the
     * activated item position. Only used on tablets.
     */
    private static final String STATE_ACTIVATED_POSITION = "activated_position";
/* The fragment's current callback object, which is notified of list item clicks. */
private Callbacks mCallbacks = sDummyCallbacks;

/**
 * The current activated item position. Only used on tablets.
 */
private int mActivatedPosition = ListView.INVALID_POSITION;

/**
 * A callback interface that all activities containing this fragment must
 * implement. This mechanism allows activities to be notified of item
 * selections.
 */
public interface Callbacks {
    /**
     * Callback for when an item has been selected.
     */
    public void onItemSelected(String id);
}

/**
 * A dummy implementation of the {@link Callbacks} interface that does
 * nothing. Used only when this fragment is not attached to an activity.
 */
private static Callbacks sDummyCallbacks = new Callbacks() {
    @Override
    public void onItemSelected(String id) {
    }
};
/**
 * Mandatory empty constructor for the fragment manager to instantiate the
 * fragment (e.g. upon screen orientation changes).
 */

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

        // TODO: replace with a real list adapter.
        setListAdapter(new ArrayAdapter<DummyContent.DummyItem>(getActivity(),
            android.R.layout.simple_list_item_activated_1,
            android.R.id.text1, DummyContent.ITEMS));
    }

    @Override
    public void onViewCreated(View view, Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);

        // Restore the previously serialized activated item position.
        if (savedInstanceState != null && savedInstanceState.containsKey(STATE_ACTIVATED_POSITION)) {
            setActivatedPosition(savedInstanceState.getInt(STATE_ACTIVATED_POSITION));
        }
    }
}
```java
@Override
public void onAttach(Activity activity) {
    super.onAttach(activity);
    // Activities containing this fragment must implement its callbacks.
    if (!(activity instanceof Callbacks)) {
        throw new IllegalStateException("Activity must implement fragment's callbacks.");
    }
    mCallbacks = (Callbacks) activity;
}

@Override
public void onDetach() {
    super.onDetach();
    // Reset the active callbacks interface to the dummy implementation.
    mCallbacks = sDummyCallbacks;
}

@Override
public void onListItemClick(ListView listView, View view, int position, long id) {
    super.onListItemClick(listView, view, position, id);
    // Notify the active callbacks interface (the activity, if the
    // fragment is attached to one) that an item has been selected.
    mCallbacks.onItemSelected(DummyContent.ITEMS.get(position).id);
}
```
@Override
public void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);
    if (mActivatedPosition != ListView.INVALID_POSITION) {
        // Serialize and persist the activated item position.
        outState.putInt(STATE_ACTIVATED_POSITION, mActivatedPosition);
    }
}

/* Turns on activate-on-click mode. When this mode is on, list items will be
* given the 'activated' state when touched. */
public void setActivateOnItemClick(boolean activateOnItemClick) {
    // When setting CHOICE_MODE_SINGLE, ListView will automatically
    // give items the 'activated' state when touched.
    getListView().setChoiceMode(activateOnItemClick ? ListView.CHOICE_MODE_SINGLE
                                                      : ListView.CHOICE_MODE_NONE);
}

private void setActivatedPosition(int position) {
    if (position == ListView.INVALID_POSITION) {
        getListView().setItemChecked(mActivatedPosition, false);
    } else {
        getListView().setItemChecked(position, true);
    }
    mActivatedPosition = position;
}
package com.example.fragmentdemo3.dummy;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

/* Helper class for providing sample content for user interfaces created by
 * Android template wizards.
 * TODO: Replace all uses of this class before publishing your app. */
public class DummyContent {

    /* An array of sample (dummy) items. */
    public static List<DummyItem> ITEMS = new ArrayList<DummyItem>();

    /* A map of sample (dummy) items, by ID. */
    public static Map<String, DummyItem> ITEM_MAP = new HashMap<String, DummyItem>();

    static {
        // Add 3 sample items.
        addItem(new DummyItem("1", "Item 1"));
        addItem(new DummyItem("2", "Item 2"));
        addItem(new DummyItem("3", "Item 3"));
    }

    private static void addItem(DummyItem item) {
        ITEMS.add(item);
        ITEM_MAP.put(item.id, item);
    }
}
/**
 * A dummy item representing a piece of content.
 */

public static class DummyItem {
    public String id;
    public String content;

    public DummyItem(String id, String content) {
        this.id = id;
        this.content = content;
    }

    @Override
    public String toString() {
        return content;
    }
}

Example 3: A Multi-pane App
Example 3: A Multi-pane App

Manifest

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.fragmentdemo3"
    android:versionCode="1"
    android:versionName="1.0">

    <uses-sdk
        android:minSdkVersion="16"
        android:targetSdkVersion="16" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme">

        <activity
            android:name="com.example.fragmentdemo3.ItemListActivity"
            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>
</manifest>
```
Example 3: A Multi-pane App

Manifest

```xml
<activity>
    android:name="com.example.fragmentdemo3.ItemDetailActivity"
    android:label="@string/title_item_detail"
    android:parentActivityName=".ItemListActivity"
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value=".ItemListActivity" />
</activity>
</application>
</manifest>
```
TabHost, Fragments, ActionBar

Questions ?