Fragments

Fragments

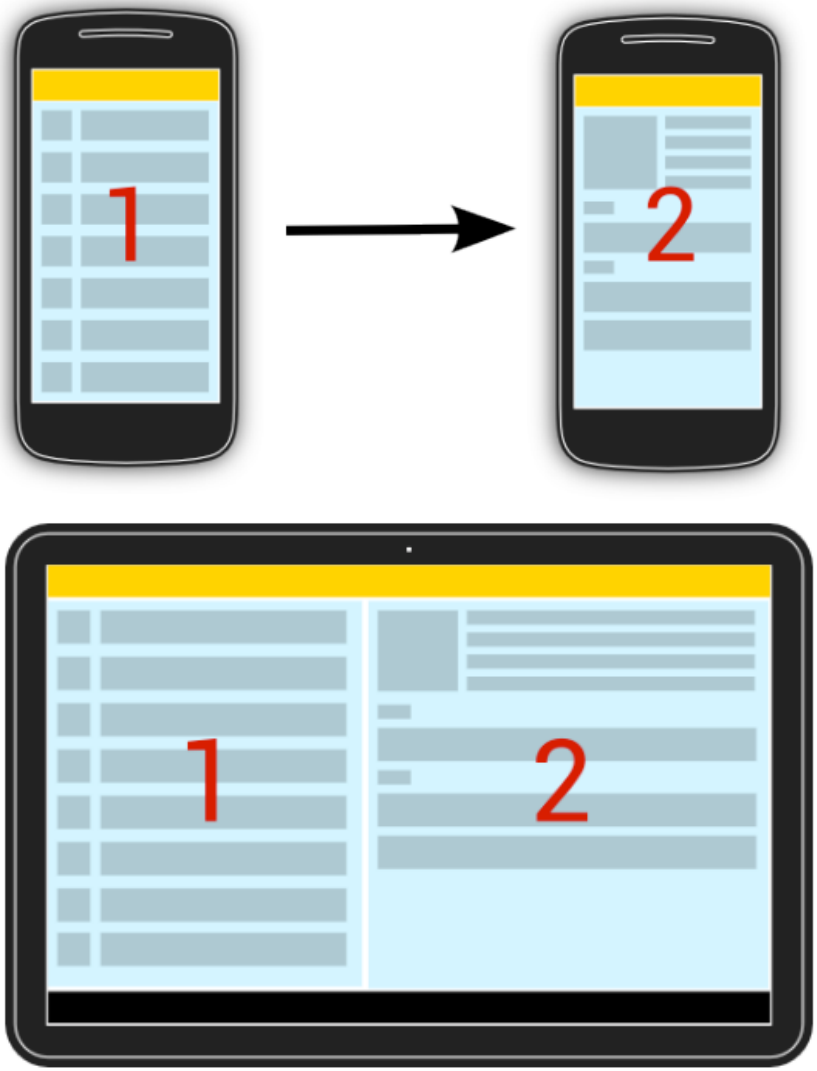
- A fragment is a self-contained, modular section of an application's user interface and corresponding behavior that can be embedded within an activity.

- Fragments can be assembled to create an activity during the application design phase, and added to, or removed from an activity during application runtime to create a dynamically changing user interface.

- Fragments may only be used as part of an activity and cannot be instantiated as standalone application elements.

- A fragment can be thought of as a functional "sub-activity" with its own lifecycle similar to that of a full activity.

Using fragments



Fragments lifecycle

Method	Description
onAttach()	The fragment instance is associated with an activity instance.The activity is not yet fully initialized
onCreate()	Fragment is created
onCreateView()	The fragment instance creates its view hierarchy. The inflated views become part of the view hierarchy of its containing activity.
onActivityCreated()	Activity and fragment instance have been created as well as thier view hierarchy. At this point, view can be accessed with the `findViewById()` method. example.
onResume()	Fragment becomes visible and active.
onPause()	Fragment is visibile but becomes not active anymore, e.g., if another activity is animating on top of the activity which contains the fragment.
onStop()	Fragment becomes not visible.

Defining a new fragment (from code)

To define a new fragment you either extend the android.app.Fragment class or one of its subclasses, for example, ListFragment, DialogFragment, PreferenceFragment or WebViewFragment.

Defining a new fragment (from code)

```
public class DetailFragment extends Fragment {
 @Override
 public View onCreateView(LayoutInflater inflater,
        ViewGroup container, Bundle savedInstanceState) {
    View view = inflater.inflate(R.layout.fragment_rssitem_detail,
         container, false);
   return view;
 }
 public void setText(String item) {
    TextView view = (TextView)
        getView().findViewById(R.id.detailsText);
    view.setText(item);
 }
}
```

XML-based fragments

```
<RelativeLayout 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"
tools:context=".FragmentDemoActivity" >
<fragment android:id="@+id/fragment_one"
android:name="com.example.myfragmentdemo.Fragmen
tOne"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_alignParentLeft="true"
android:layout_centerVertical="true"
tools:layout="@layout/fragment_one_layout" />
</RelativeLayout>
```

Adding-removing fragments at runtime

- The FragmentManager class and the FragmentTransaction class allow you to add, remove and replace fragments in the layout of your *activity*.

- Fragments can be dynamically modified via transactions. To dynamically add fragments to an existing layout you typically define a container in the XML layout file in which you add a *Fragment*.

```
FragmentTransaction ft = getFragmentManager().beginTransaction();
ft.replace(R.id.your_placehodler, new YourFragment());
ft.commit();
```

- A new *Fragment* will replace an existing *Fragment* that was previously added to the container.

Finding if a fragment is already part of your Activity

```
DetailFragment fragment = (DetailFragment) getFragmentManager().
findFragmentById(R.id.detail_frag);

if (fragment==null || ! fragment.isInLayout()) {
        // start new Activity
} else {
        fragment.update(...);
}
```

Communication: activity -> fragment

- In order for an activity to communicate with a fragment, the activity must identify the fragment object via the ID assigned to it using the findViewById() method. Once this reference has been obtained, the activity can simply call the public methods of the fragment object.

Communication: fragment-> activity

- Communicating in the other direction (from fragment to activity) is a little more complicated.

A) the fragment must define a listener interface, which is then implemented within the activity class.

```
public class MyFragment extends Fragment {
  AListener activityCallback;
  public interface AListener {
       public void someMethod(int par1, String par2);
  }
  …
```

Communication: fragment-> activity

B. the onAttach() method of the fragment class needs to be overridden and implemented. The method is passed a reference to the activity in which the fragment is contained. The method must store a local reference to this activity and verify that it implements the interface.

```
public void onAttach(Activity activity) {
  super.onAttach(activity);
  try { activityCallback = (AListener) activity;
  } catch (ClassCastException e) {
       throw new ClassCastException(activity.toString()
            + " must implement ToolbarListener");
} }
```

Communication: fragment-> activity

C. The next step is to call the callback method of the activity from within the fragment. When and how this happens is entirely dependent on the circumstances under which the activity needs to be contacted by the fragment. For the sake of an example, the following code calls the callback method on the activity when a button is clicked:

```
public void buttonClicked(View view) {
    activityCallback.someMethod(arg1, arg2);
}
```

Communication: fragment-> activity

- All that remains is to modify the activity class so that it implements the ToolbarListener interface.

```
public class MyActivity extends FragmentActivity
implements MyFragment.AListener {
   public void someMethod(String arg1, int arg2) {
         // Implement code for callback method
   }
.
.
}
```

Esempio

- vedi http://www.vogella.com/tutorials/AndroidFragments/article.html

sez. 10