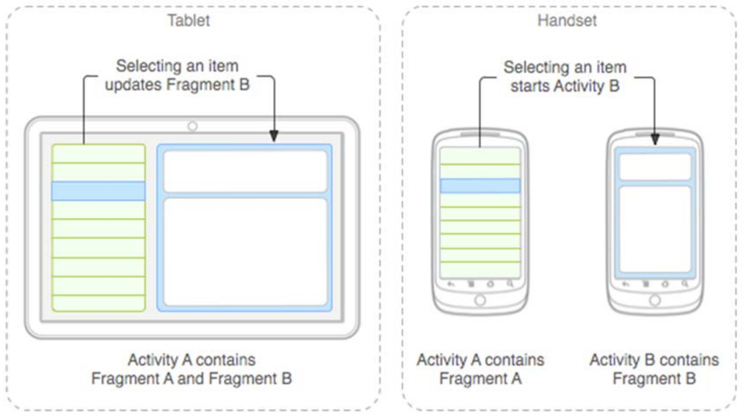
Android Fragments

Akhilesh Tyagi

Fragment

- A *baby* activity – attachable and detachable dynamically.
- A Fragment represents a behavior or a portion of user interface in anActivity
- a modular section of an activity, which has its own lifecycle, receives its own input events, and which you can add or remove while the activity is running.
- A fragment must always be embedded in an activity and the fragment's lifecycle is directly affected by the host activity's lifecycle.

Design Flexibility

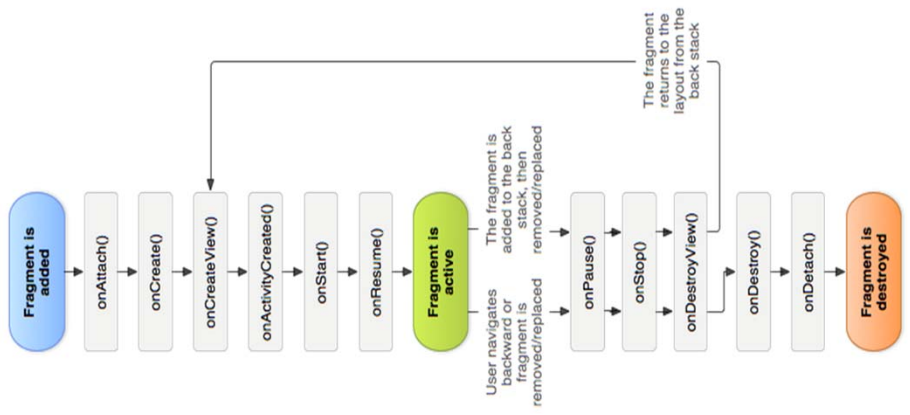

Tablet
Selecting an item updates Fragment B
Activity A contains Fragment A and Fragment B
Handset
Selecting an item starts Activity B
Activity A contains Fragment A
Activity B contains Fragment B

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

Advantages

- Dynamic views/layouts
- Reusable Layouts
- Same layouts can be used in multiple devices

Fragment Lifecycle



```
public class MySkeletonFragment extends Fragment {

 // Called when the Fragment is attached to its parent Activity.
 @Override
 public void onAttach(Activity activity) {
  super.onAttach(activity);
  // Get a reference to the parent Activity.
}
 // Called to do the initial creation of the Fragment.
 @Override
 public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  // Initialize the Fragment.
 }
```

- **onCreate():** The system calls this when creating the fragment. Within your implementation, you should initialize essential components of the fragment that you want to retain when the fragment is paused or stopped, then resumed.
- **onCreateView():** The system calls this when it's time for the fragment to draw its user interface for the first time. To draw a UI for your fragment, you must return a View from this method that is the root of your fragment's layout. You can return null if the fragment does not provide a UI.

```
// Called once the Fragment has been created in order for it to
  // create its user interface.
  @Override
  public View onCreateView(LayoutInflater inflater,
                           ViewGroup container,
                           Bundle savedInstanceState) {
    // Create, or inflate the Fragment's UI, and return it.
    // If this Fragment has no UI then return null.
    return inflater.inflate(R.layout.my_fragment, container, false);
  }
```

```
// Called once the parent Activity and the Fragment's UI have
  // been created.
  @Override
  public void onActivityCreated(Bundle savedInstanceState) {
    super.onActivityCreated(savedInstanceState);
    // Complete the Fragment initialization - particularly anything
    // that requires the parent Activity to be initialized or the
    // Fragment's view to be fully inflated.
  }
```

```
// Called at the start of the visible lifetime.
  @Override
  public void onStart(){
    super.onStart();
    // Apply any required UI change now that the Fragment is visible.
  }

  // Called at the start of the active lifetime.
  @Override
  public void onResume(){
    super.onResume();
    // Resume any paused UI updates, threads, or processes required
    // by the Fragment but suspended when it became inactive.
  }
```

```
// Called at the end of the active lifetime.
 @Override
 public void onPause(){
 // Suspend UI updates, threads, or CPU intensive processes that don't
//need to be updated when the Activity isn't the active foreground
// activity. Persist all edits or state changes as after this call the process is
likely to be killed.
   super.onPause();
 }
 // Called to save UI state changes at the
 // end of the active lifecycle.
 @Override
 public void onSaveInstanceState(Bundle savedInstanceState) {
   // Save UI state changes to the savedInstanceState.
   // This bundle will be passed to onCreate, onCreateView, and
   // onCreateView if the parent Activity is killed and restarted.
   super.onSaveInstanceState(savedInstanceState);
 }
```

```
// Called at the end of the visible lifetime.
 @Override
 public void onStop(){
   // Suspend remaining UI updates, threads, or processing
   // that aren't required when the Fragment isn't visible.
   super.onStop();
 }

 // Called when the Fragment's View has been detached.
 @Override
 public void onDestroyView() {
   // Clean up resources related to the View.
   super.onDestroyView();
 }
```

```
// Called at the end of the full lifetime.
 @Override
 public void onDestroy(){
   // Clean up any resources including ending threads,
   // closing database connections etc.
   super.onDestroy();
 }

 // Called when the Fragment has been detached from its
parent Activity.
 @Override
 public void onDetach() {
   super.onDetach();
 }
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<LinearLayout  android:layout_height="match_parent"
        android:layout_width="match_parent"
        android:orientation="horizontal"
        xmlns:android="http://schemas.android.com/apk/res/android"
>
        <fragment android:layout_height="match_parent"
        android:layout_width="match_parent"
        android:layout_weight="1"
        android:id="@+id/my_list_fragment"
        android:name="com.paad.weatherstation.MyListFragment"/    >
        <fragment android:layout_height="match_parent"
        android:layout_width="match_parent"
        android:layout_weight="3"
        android:id="@+id/details_fragment"
        android:name="com.paad.weatherstation.DetailsFragment"/>
</LinearLayout>
```

XML Fragment layouts

- Fragment element well suited for static layouts
- If the layout is to be dynamically modified in the program, use *containers* to pass to fragment managers.

```
<?xml version="1.0" encoding="UTF-8"?>
<LinearLayout android:layout_height="match_parent"
        android:layout_width="match_parent"
        android:orientation="horizontal"
        xmlns:android="http://schemas.android.com/apk/res/android
">
        <FrameLayout android:layout_height="match_parent"
                android:layout_width="match_parent"
                android:layout_weight="1"
                android:id="@+id/ui_container"/>
        <FrameLayout android:layout_height="match_parent"
                android:layout_width="match_parent"
                android:layout_weight="3"
                android:id="@+id/details_container"/>
</LinearLayout>
```

Fragment Transactions

- To add, remove, and replace fragments within an activity.

```
FragmentManager fragManager =
        getFragmentManager();
FragmentTransaction transaction =
        fragManager.beginTransaction();
transaction.add(R.id.container, firstFragment);
transaction.commit();
```

```
public class MyFragmentActivity extends Activity {

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

    // Inflate the layout containing the Fragment containers
    setContentView(R.layout.fragment_container_layout);

    FragmentManager fm = getFragmentManager();

    // Check to see if the Fragment back stack has been populated
    // If not, create and populate the layout.
    DetailsFragment  detailsFragment =
      (DetailsFragment) fm.findFragmentById(R.id.details_container);
if (detailsFragment == null) {
      FragmentTransaction  ft = fm.beginTransaction();
      ft.add(R.id.details_container, new DetailsFragment());
      ft.add(R.id.ui_container, new MyListFragment());
      ft.commit();
    }
  }
}
```

- Calling commit() does not perform the transaction immediately. Rather, it schedules it to run on the activity's UI thread (the "main" thread) as soon as the thread is able to do so.
- If necessary, however, you may call executePendingTransactions() from your UI thread to immediately execute transactions submitted by commit(). Doing so is usually not necessary unless the transaction is a dependency for jobs in other threads.

Built-in Fragment classes

- **DialogFragment**: Displays a floating dialog. Using this class to create a dialog is a good alternative to using the dialog helper methods in the Activity class, because you can incorporate a fragment dialog into the back stack of fragments managed by the activity, allowing the user to return to a dismissed fragment.
- **ListFragment**: Displays a list of items that are managed by an adapter (such as a SimpleCursorAdapter), similar to ListActivity. It provides several methods for managing a list view, such as the onListItemClick() callback to handle click events.
- **PreferenceFragment:** Displays a hierarchy of Preference objects as a list, similar to PreferenceActivity. This is useful when creating a "settings" activity for your application.

Communicating with the Activity

- the fragment can access the Activity instance with getActivity() and easily perform tasks such as find a view in the activity layout:

```
View listView = getActivity().findViewById(R.id.list);
```

- activity can call methods in the fragment by acquiring a reference to the Fragment from FragmentManager, using findFragmentById()

```
ExampleFragment fragment = (ExampleFragment)
getFragmentManager().findFragmentById(R.id.exampl
e_fragment);
```

Fragment Event Handling

- Say a *Button* in a fragment; Which class receives the *onClick* event – Fragment class or Activity class? Depends!
- *button.setOnClickListener(new Button.OnClickListener() { public void onClick(View v) {…}})* sends the event to the fragment
- *android:onClick="onClick“* in layout resource file sends the event to the Activity class.

To-DO-LIST with Fragments

- Two views in a linear layout: *EditText* and *ListView*.
- Create EditText as a separate XML layout resource new_item_fragment.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<EditText android:layout_height="wrap_content"
      android:layout_width="match_parent"
      android:contentDescription=
            "@string/addItemContentDescription"
      android:hint="@string/addItemHint"
      android:id="@+id/myEditText"/>
```

Fragment classes

```
 public class NewItemFragment extends
Fragment {
@Override
  public View onCreateView(LayoutInflater inflater,
                        ViewGroup container,
                        Bundle savedInstanceState) {
  return inflater.inflate(R.layout.new_item_fragment, container,
false);
  }
```

- Encapsulate functionality a fragment provides. Let us start with event listeners interface.

```
public interface OnNewItemAddedListener {
  public void onNewItemAdded(String newItem);
}
```

- Var to store reference to parent ToDoListActivity. This ref can be assigned in *onAttach*.

```
private OnNewItemAddedListener onNewItemAddedListener;
@Override
public void onAttach(Activity activity) {
   super.onAttach(activity);
   try {
     onNewItemAddedListener = (OnNewItemAddedListener) activity;} catch
(ClassCastException e) {
            throw new ClassCastException(activity.toString() +
    "must implement OnNewItemAddedListener");
```

- Move the *editText.onClickListener* from toDoListActivity to this fragment. When user adds a new item, rather than adding it to an array, pass it to the parent activity's *OnNewItemAddedListener.onNewItemAdded*

@Override public View **onCreateView(…)**

View view = inflater.inflate(R.layout.new_item_fragment, container, false);

```
final EditText myEditText =  (EditText)
          view.findViewById(R.id.myEditText);
myEditText.setOnKeyListener(myEditText.setOnKeyListener(ne
w View.OnKeyListener() {
     public boolean onKey(View v, int keyCode, KeyEvent event) {
```

```
if (event.getAction() == KeyEvent.ACTION_DOWN)
          if ((keyCode ==
KeyEvent.KEYCODE_DPAD_CENTER) ||
             (keyCode == KeyEvent.KEYCODE_ENTER)) {
           String newItem = myEditText.getText().toString();
         onNewItemAddedListener.onNewItemAdded(newItem
);
          myEditText.setText("");
          return true;
          }
  return false;
}
});

return view;

}
```

List Fragment

- New fragment for the list of to-do items. Android provides a native ListFragment class.

```
Import android.app.ListFragment
public class ToDoListFragment extends ListFragment{ }
```

- In Activity, update the XML layout file:

```
<?xml version="1.0" encoding="UTF-8"?>
<LinearLayout android:layout_height="match_parent"
        android:layout_width="match_parent"
        android:orientation="vertical"
        xmlns:android="http://schemas.android.com/apk/res/androi
d">
        <fragment android:layout_height="wrap_content"
                android:layout_width="match_parent"
                android:id="@+id/NewItemFragment"
        android:name="com.paad.todolist.NewItemFragment"/>
        <fragment android:layout_height="wrap_content"
                android:layout_width="match_parent"
                android:id="@+id/TodoListFragment"
        android:name="com.paad.todolist.ToDoListFragment"/>
</LinearLayout>
```

- In Activity ToDoListActivity, in onCreate, get fragment manager

```
public class ToDoListActivity extends Activity implements
NewItemFragment.OnNewItemAddedListener {

  private ArrayAdapter<String> aa;
  private ArrayList<String> todoItems;

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

    // Inflate your view
    setContentView(R.layout.main);
```

```
// Get references to the Fragments
    FragmentManager fm = getFragmentManager();
    ToDoListFragment todoListFragment =

(ToDoListFragment)fm.findFragmentById(R.id.TodoListFragment);

    // Create the array list of to do items
    todoItems = new ArrayList<String>();

    // Create the array adapter to bind the array to the listview
  aa = new
ArrayAdapter<String>(this,android.R.layout.simple_list_item_1,
                         todoItems);
    // Bind the array adapter to the listview.
    todoListFragment.setListAdapter(aa); }
```

- ListView is now connected to the Array with adapter aa.
- Now implement the listener:

```
public void onNewItemAdded(String newItem) {
   todoItems.add(newItem);
   aa.notifyDataSetChanged();
  }
```