Activities and Fragments

236503
Dr. Oren Mishali
Based on the Official Android Development Guide

An Activity

• A “screen” in an android app is implemented by a class inheriting from Activity
• All activity classes must have a corresponding <activity> declaration in AndroidManifest.xml

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

Declaration of a launcher activity

Actually, a “screen” may also be a Fragment
Activity lifecycle – 3 states

- **Resumed** – the activity is in the foreground and the user can interact with it
- **Paused** – the activity is partially obscured by another activity; does not receive user input and cannot execute any code
- **Stopped** – the activity is completely hidden and not visible to the user; it is considered to be in the background. State information is retained

If an activity is paused or stopped, the system can drop the activity from memory. When it is displayed again, it must be completely restarted and restored.
Activity lifecycle methods

• The entire lifecycle of an activity is defined by the following Activity methods

```java
public class Activity extends ApplicationContext {
    protected void onCreate(Bundle savedInstanceState);
    protected void onStart();
    protected void onRestart();
    protected void onResume();
    protected void onPause();
    protected void onStop();
    protected void onDestroy();
}
```

• These are “callback” methods (hooks) that you can override to do appropriate work when the activity changes state
• almost all subclasses of Activity will implement:
  - `onCreate(Bundle)` – activity is initialized,
  - `setContentView()` sets the UI layout,
  - `findViewById(int)` is used to retrieve UI widgets
  - `onPause()` – here we deal with the user leaving the activity, user changes should be committed

Should we implement all lifecycle methods?

• You probably don’t need to implement all lifecycle methods

• Implement those that ensure the app behaves as expected, i.e., that it does not crash or lose user’s progress when
  - the user receives a phone call or switches to another app
  - the user leaves the app and returns to it at a later time
  - when the screen rotates between landscape and portrait orientation
Activity.onCreate()

• The system creates every new instance of an Activity by calling its `onCreate()` method
• `onCreate()` is called only once for the entire lifecycle of the activity
• Within `onCreate()` we place setup operations such as declaring the user interface and defining member variables
• Once `onCreate()` finishes execution, the system calls `onStart()` and `onResume()` in quick succession
• `onCreate()` has a parameter (next slides)

Activity.onPause()

• When the system calls `onPause()` for your activity, it technically means your activity is still partially visible
• Most often it is an indication that the user is leaving the activity and it will soon enter the Stopped state
When to use `onPause()`?

- To stop animations or other ongoing actions that could consume CPU
- Commit unsaved changes, but only if users expect such changes to be permanently saved when they leave (such as a draft email)
- Release system resources, e.g., Camera

Generally, you should not use `onPause()` to store user changes to permanent storage because it can slow the visible transition to the next activity. Instead, perform heavy-load shutdown operations during `onStop()`

Activity. `onResume()`

- When the user resumes the activity from the Paused state, the system calls `onResume()`
- Note that the system calls this method every time the activity comes into the foreground
- As such, you should implement `onResume()` to initialize components released during `onPause()`, any other initializations that must occur each time the activity enters the Resumed state
**onStop(), onRestart()**

- When the user leaves your activity, the system calls `onStop()` to stop the activity.
- If the user returns, the system calls `onRestart()`, quickly followed by `onStart()` and `onResume()`.
- Note that `onPause()` is always called before `onStop()`.

---

**How to implement `onStop()`?**

- Use `onStop()` to write information to a database, e.g., saving the content of a draft note to persistent storage.
- It's uncommon that an app needs to use `onRestart()` to restore the activity's state.
- However, based on your implementation of `onStop()`, you may re-create resources, but `onStart()` is usually better for that purpose.
Stopping & restarting – example scenarios

• The user performs an action that starts a new activity
  • The current activity is stopped
  • The second activity is created
  • The user presses the Back button
  • The first activity is restarted
• The user receives a phone call while using the app
• While using the app, the user opens the Recent Apps window and switches to another app
  • The user then returns to the app

Stopped state – what happens to the instance?

• When stopped, the Activity’s object is kept resident in memory and is recalled when the activity resumes
• The system also keeps track of the current state for each View in the layout so you don’t need to save and restore it
  • E.g., if the user entered text into an EditText, that content is retained

**Note:** Once stopped, the system might destroy the activity’s instance if it needs to recover system memory. However even if the system destroys your activity while it’s stopped, it still retains the state of the View objects in a Bundle (key-value pairs)
**Activity.onCreate(Bundle savedInstanceState)**

- The system uses the `Bundle` instance to save information about each `View` object in the layout.
- If the activity instance is destroyed and recreated, the state of the layout is restored with no additional code required.
- However, the activity might have more state information that we'd like to restore, e.g., member variables that track the user's progress in the activity.

**Note:** In order for the Android system to restore the state of the views in your activity, each view must have a unique ID.

---

**How to save additional data?**

- To save additional data about the activity state, override `onSaveInstanceState()` callback.
- To restore the data, use `onCreate()` or `onRestoreInstanceState()`.
Saving additional data – an Example

- Always call `super.onSaveInstanceState()`
- `onCreate()` may create a totally new instance hence `savedInstanceState` may be null
- This check is not needed in `onRestoreInstanceState()`
- Additional reading in Handling Runtime Changes

```java
static final String STATE_SCORE = "playerScore";
static final String STATE_LEVEL = "playerLevel";
...

@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
    // Save the user's current game state
    savedInstanceState.putInt(STATE_SCORE, mCurrentScore);
    savedInstanceState.putInt(STATE_LEVEL, mCurrentLevel);

    // Always call the superclass so it can save the view hierarchy state
    super.onSaveInstanceState(savedInstanceState);
}

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState); // Always call the superclass first

    // Check whether we're recreating a previously destroyed instance
    if (savedInstanceState != null) {
        // Restore value of members from saved state
        mCurrentScore = savedInstanceState.getInt(STATE_SCORE);
        mCurrentLevel = savedInstanceState.getInt(STATE_LEVEL);
    } else {
        // probably initialize members with default values for a new instance
    }
    ...
}
```

---

Final notes: use your app regularly!

- “Play” with your app regularly
  - Rotate the screen
  - Switch to another app and return etc.
- Notice unexpected behavior
  - Crash?
  - Unsaved state?

You may need to implement some callback methods...
Why Fragments?

- A fragment is a modular section of an activity, a “sub activity”
  - It has its own lifecycle
  - receives its own input events
  - and it can be added or removed while the activity is running
- Fragments provide more flexibility in designing the UI
- Fragments are useful, e.g., in supporting a wide range of screens
- The same fragments may be reused to create different layout configurations

Note that some UI patterns such as Tabs require the use of Fragments.

A fragment “lives” within a parent activity.

Required Setup

- We will extend the Fragment class using the Support Library
- Set up the project to use the v4 library
  - Or the v7 appcompat library

This is the project created by Android Studio that by default includes both libraries
How To Create a Fragment

- Extend the `Fragment` class
- Override key lifecycle methods to insert your app logic
- Define the layout in XML and connect within `onCreateView()`

```java
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.ViewGroup;

public class ArticleFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.article_view, container, false);
    }
}
```

- Like an activity, a fragment may implement other lifecycle callbacks
- E.g., when the activity's `onPause()` is called, any fragments in the activity also receive a call to `onPause()`
- More info about the fragment’s life cycle

---

Adding Fragments to an Activity Via XML

- Each instance of a Fragment class must be associated with a parent Activity

```xml
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:orientation="Horizontal"
              android:layout_width="fill_parent"
              android:layout_height="fill_parent">

    <Fragment android:name="com.example.android.fragments.HeadlinesFragment"
              android:id="@+id/headlines_fragment"
              android:layout_weight="1"
              android:layout_width="wpd"
              android:layout_height="match_parent" />

    <Fragment android:name="com.example.android.fragments.ArticleFragment"
              android:id="@+id/article_fragment"
              android:layout_weight="2"
              android:layout_width="wpd"
              android:layout_height="match_parent" />

</LinearLayout>
```

```java
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.news_articles);
    }
}
```

- If you're using the v7 appcompat library, extend `AppCompatActivity` instead
Adding a Fragment at runtime

- The fragments are used differently in a small screen
- Here, the fragments are not declared in XML, but created and added at runtime

res/layout/news_articles.xml

```xml
<fragmentLayout xmlns:android="http://schemas.android.com
android:id="@+id/fragment_container"
android:layout_width="match_parent"
android:layout_height="match_parent" />
```

The activity layout must include a container view

---

Replace One Fragment with Another

- To replace a fragment use `replace()` instead of `add()`
- To allow backward navigation, call `addToBackStack()` before commit

```java
import android.os.Bundle;
import android.support.v4.app.FragmentActivity;

public class MainActivity extends FragmentActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.news_articles);

        // Check that the activity is using the layout version with
        // the fragment_container FrameLayout
        if (findViewById(R.id.fragment_container) != null) {
            // However, if we’re being restored from a previous state,
            // then we don’t need to do anything and should return or else
            // we could end up with overlapping fragments.
            if (savedInstanceState != null) {
                return;
            }

            // create a new Fragment to be placed in the activity layout
            HeadlinesFragment firstFragment = new HeadlinesFragment();

            // In case this activity was started with special instructions from an
            // Intent, pass the Intent’s extras to the fragment as arguments
            firstFragment.setArguments(getIntent().getExtras());

            // Add the fragment to the ‘fragment_container’ FrameLayout
            getSupportFragmentManager().beginTransaction()
                .add(R.id.fragment_container, firstFragment)
                .commit();
        }
    }
}
```

Note: the method `addToBackStack()` may be provided with a unique name for the transaction. The name is only needed to perform advanced operations, see here for more details
Communication Between Fragments

• A Fragment may need to communicate with another Fragment
  • E.g., change the content based on a user event
• Fragment-to-Fragment communication should be done through the associated Activity
  • Fragments should not communicate directly
• See here an example for a possible communication pattern

The host activity can deliver messages to a fragment by capturing the Fragment instance with `findFragmentById()`, and then call its public methods

Additional reading: Fragments API Guide
  Note the example at the end of the document