Android Platform Overview

Applications

Home Screen, Contacts, Phone, Browser, etc

Application Framework

Activity Manager, Window Manager, Content Providers, etc

Libraries

Media Framework, SQLite, OpenGL, WebKit etc.

Android RunTime

Core libraries

Linux Kernel

Display driver, camera driver, audio drivers, power management, WiFi driver, USB driver, Flash memory driver, etc.

Main Building Blocks of Android

Main Building Blocks in Android App

- All building blocks live in Application Context
- All 4 building blocks are implemented as Java classes:
  - Activities (this is the only one that has GUI)
  - Services
  - Content Providers
  - Broadcast Receivers
- Activities and Services extend the Context class.
  Therefore, they can be used directly to access the Context.

Activity

Changing Activities (i.e. "flipping pages")

- Add the destination activity to the project
- Use "Intent" to move from one activity to another

Intent

- Intents are messages passed among Android building blocks.
- Intents are Asynchronous
- Some uses of Intent:
  - Navigate among different activities, either within the same app or to another app
  - Pass data from one activity to another
  - Broadcast and receive messages

Intent – Navigation from one Activity to Another

To switch to a new screen (Activity):

```
Intent   myIntent;

myIntent = new Intent(this, Activity1.class);

startActivity(myIntent);
```

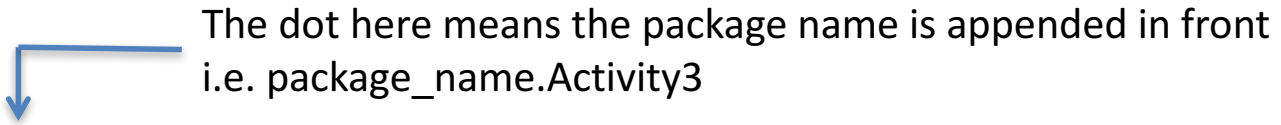
To return to parent screen (Activity):

```
finish();
```

Note: Every time you add an activity, Android Studio will automatically declare it in AndroidManifest.xml. Example:

```
<!-- A child of the main activity -->
<activity
   android:name=    ".Activity1"
   android:label="@string/activity1_label"
   android:parentActivityName="com.example.androidintent.MainActivity" >
</activity>
```

Note: Not every activity need to be a child activity. Can define an activity at the same level in the hierarchy as MainActivity by




```
<activity
   android:name=".Activity3"
   android:label="@string/title_activity_activity3" >
</activity>
```

Intent – Pass Data from one Activity to Another

To switch to a new screen and also pass data over:

Method 1:

```
myIntent = new Intent(this, Activity3.class);
myIntent.putExtra("sports", "football");
myIntent.putExtra("teamSize", 11);
startActivity(myIntent);
```

Method 2:

```
myIntent = new Intent(this, Activity3.class);
Bundle bundleObject = new Bundle();
bundleObject.putString("sports", "volleyball");
bundleObject.putInt("teamSize", 6);
myIntent.putExtras(bundleObject);
startActivity(myIntent);
```

To request child activity to pass data back to parent:

```
int requestCode = 1;

startActivityForResult(myIntent, requestCode);
```

Note: if we use startActivityForResult, need to define onActivityResult to process the returned data:

```
public void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (requestCode == 1) {
            if (resultCode == RESULT_OK) {
              // access the returned data using data.getIntExtra("teamSize",0)
            }
        }
}
```

Note: In the child activity, need to define the following to return data to parent activity:

```
myIntent = new Intent();
myIntent.putExtra("teamSize", 22);
setResult(RESULT_OK, myIntent);
finish();
```

Note: In the child activity, use the following in onCreate method to access the data passed to it from the parent activity:

```
getIntent().getStringExtra("sports")

getIntent().getIntExtra("teamSize", 0)));
```

Default value to be returned if no int value is stored in “teamSize”

Another Way to Specify Intent for Activity Flipping:

In the destination Activity (eg. Activity4), add the following:

```
private static String key_name  = "Name";
private static String key_np       = "Number of players";

public static Intent getNewIntent(String name, int np, Context ctx) {
   Intent intent = new Intent(ctx, Activity4.class);
   intent.putExtra(key_name, name);
   intent.putExtra(key_np, np);
   return intent;
}

@Override
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_4);
   Toast.makeText(this, ”name = "+getIntent().getStringExtra(key_name) + "  " +
                                 getIntent().getIntExtra(key_np, 0) + " ",
                                 Toast.LENGTH_LONG).show();
}
```

Then in the source Activity, do the following:

```
public void onClick_GoToActivity4(View view) {

    Intent myIntent = Activity4.getNewIntent("Pokemon Go", 3, this );

    startActivity(myIntent);
}
```

Starting Activities with Intents

- Need to specify context:
    - Context is used to access global application information.
    - Since Activity is subclass of Context, we can also use activity.this as context but need to be aware of memory leak issues (see the next 3 slides).

Application Context or Activity Context ?

- One of the common mistakes that lead to memory leak problems in Android:
  - inappropriate referencing to Context

- In Android, there are two types of context:
  - Activity
  - Application

- Avoid situations whereby a strong reference having a different life cycle as Activity is made on the Activity context. When Activity dies, the strong reference will still point to its Context thus causing memory leak.

- Another solution is to use Application Context instead

An example from http://android-developers.blogspot.sg/2009/01/avoiding-memory-leaks.html

```
private static Drawable sBackground;

@Override
protected void onCreate(Bundle state) {
  super.onCreate(state);

  TextView label = new TextView(this);
  label.setText("Leaks are bad");

  if (sBackground == null) {
    sBackground = getDrawable(R.drawable.large_bitmap);
  }
  label.setBackgroundDrawable(sBackground);

  setContentView(label);
}
```

static object is tied to class, not instance

sBackground callback references TextView label which lives in Application Context

When Activity is killed (such as for example when user changes the mobile phone from landscape to portrait, or vice versa), the static object still strongly reference the Context of the Activity, therefore causing memory leak.

Note:

- can clear stack of all previous activities by preceding a startActivity(intent) with

- *intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);*

  startActivity(intent);

This is useful if you want to get out at any level in the hierarchy of activities.

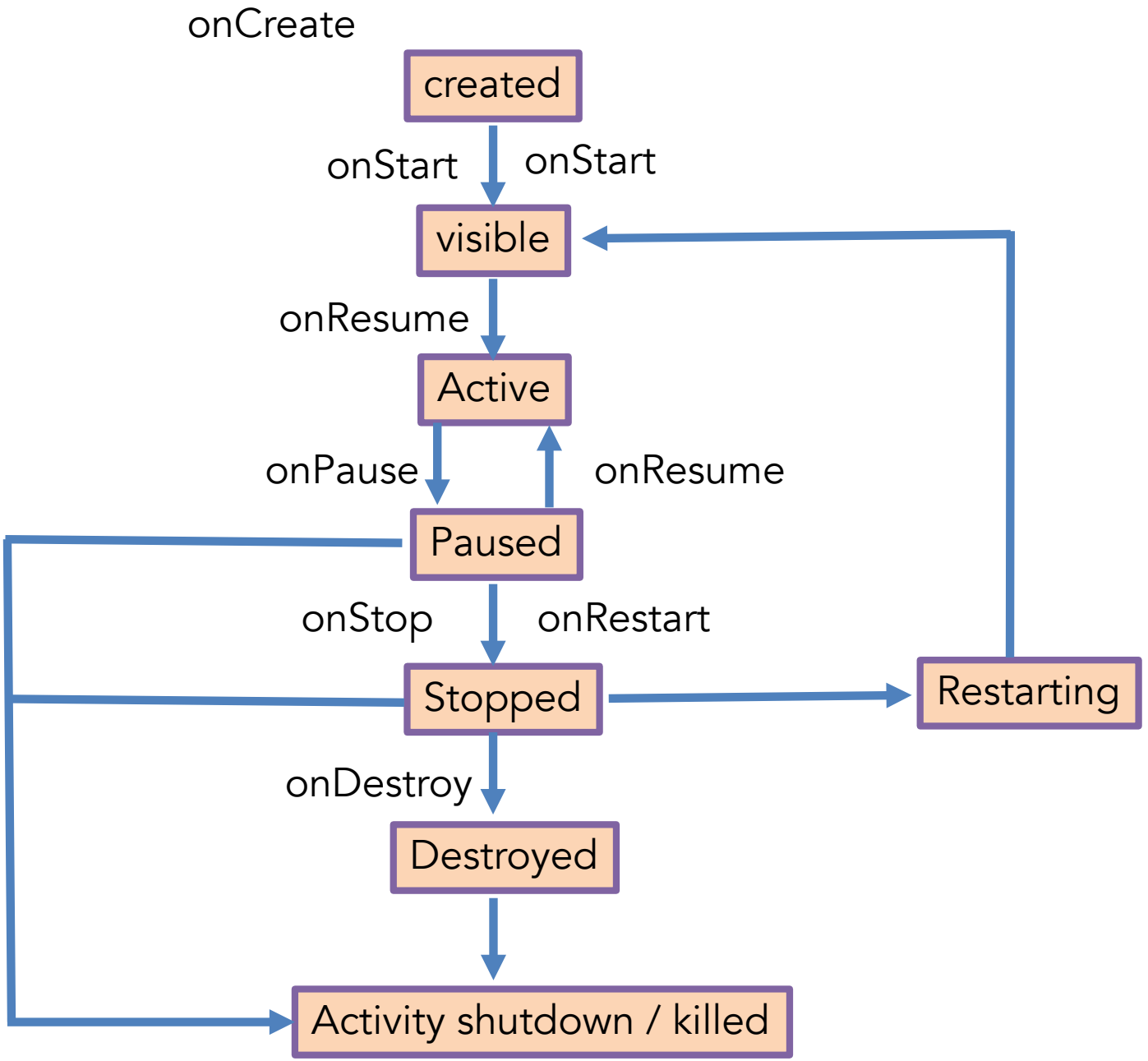
Priority

- If more than one activity can accept the same intent, can use priority
- Priority between -1000 and 1000. Higher values means higher priority

Activity Life Cycle

Android announces activity lifecycle state changes to activity by calling specific activity methods

- Eg. onCreate, onStart, onResume, onPause, onStop, onDestroy


onCreate
created
onStart
onStart
visible
onResume
Active
onPause
onResume
Paused
onStop
onRestart
Stopped
Restarting
onDestroy
Destroyed
Activity shutdown / killed

Note: if you need to save your activity data, safer not to do it in onStop or onDestroy as they may not be called at all

- Eg. onStop may not be called if Android killed the application process when it has low memory. So don't save persistence data in onStop. Should do that in onPause instead. Same thing for onDestroy. onDestroy may not be called at all when Android kills the application due to, for example, low memory

But for post-Honeycomb, it is ok as onStop is guaranteed to be the last to be executed.

From
http://stackoverflow.com/questions/9625920/should-the-call-to-the-superclass-method-be-the-first-statement

Methods that you override that are part of component creation (`onCreate()`, `onStart()`, `onResume()`, etc.), you should chain to the superclass as the first statement, to ensure that Android has its chance to do its work before you attempt to do something that relies upon that work having been done.

Methods you override that are part of component destruction (`onPause()`, `onStop()`, `onDestroy()`, etc.), you should do your work first and chain to the superclass as the last thing. That way, in case Android cleans up something that your work depends upon, you will have done your work first.

Methods that return something other than `void` (`onCreateOptionsMenu()`, etc.), sometimes you chain to the superclass in the return statement, assuming that you are not specifically doing something that needs to force a particular return value.

Everything else -- such as `onActivityResult()` -- is up to you, on the whole. I tend to chain to the superclass as the first thing, but unless you are running into problems, chaining later should be fine.

Note: after onStart, it is visible but user cannot interact yet. After onResume, then user can interact

- starting state: when an activity does not exist in memory, it is in a starting state.

- The transition from starting state to running state is very computationally intensive and affects battery life. This is why we don't automatically destroy activities in case user needs them later.

- running state: The activity in a running state is the one currently on the screen and interacting with the user. There is only one running activity at any given time

- The running activity is the one that has priority in terms of getting memory and resources

- paused state: when an activity is not in focus (i.e. not interacting with the user) but still visible on the screen, we say it is in a paused state (imagine a dialog box scenario). All activities go through a paused state before being stopped. Paused activities still have high priority in terms of getting memory and other resources

- stopped state: when an activity is not visible, but still in memory, we say it is in a stopped state.

- restarting a stopped activity is much cheaper than starting an activity from scratch

- destroyed state: a destroyed activity is no longer in memory

Summary of Activity Life Cycle

Starting State :

- Activity about to occupy memory
- Activity about to go through a set of callback methods

Computationally Intensive

Eats up Battery

Running State:

- Activity “in Focus”
- Only 1 running activity at any one time
- Enjoys priority in using memory and resources

Paused State :

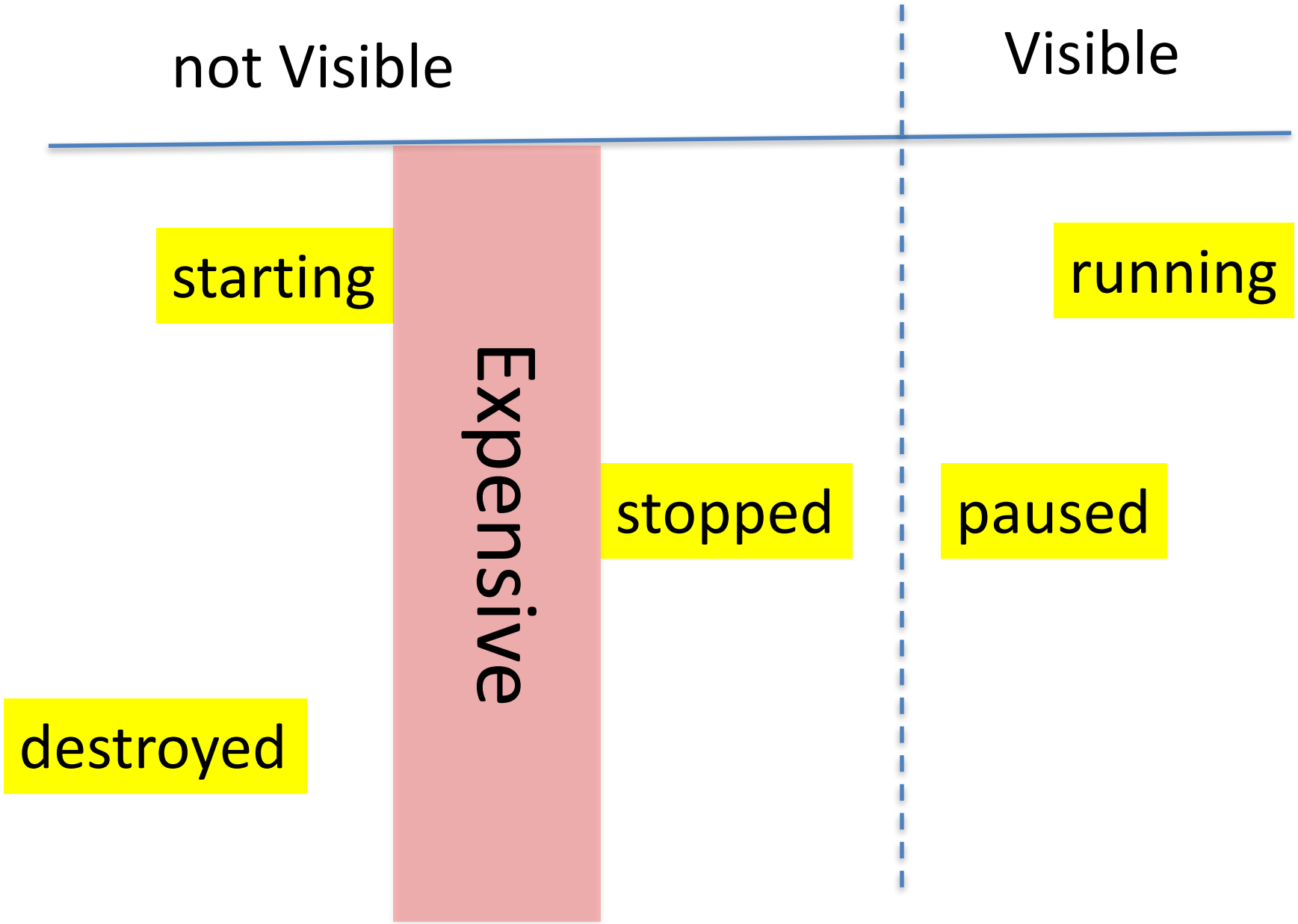
- Activity visible on screen but not in focus
- eg. paused by dialog boxes
- All activities must go through paused state before being stopped
- Still enjoys high priority in using memory and other resources

Stopped State:

- Activity not visible but still in memory
- Could be brought back to running again
- Restarting stopped activity much cheaper than starting an activity from scratch
- Can be removed from memory

Destroyed State :

- no longer in memory
- not necessarily must pass through stopped state before destroy
- can be destroyed directly from paused state


not Visible
Visible
starting
running
Expensive
stopped
paused
destroyed

Fragment

Fragments

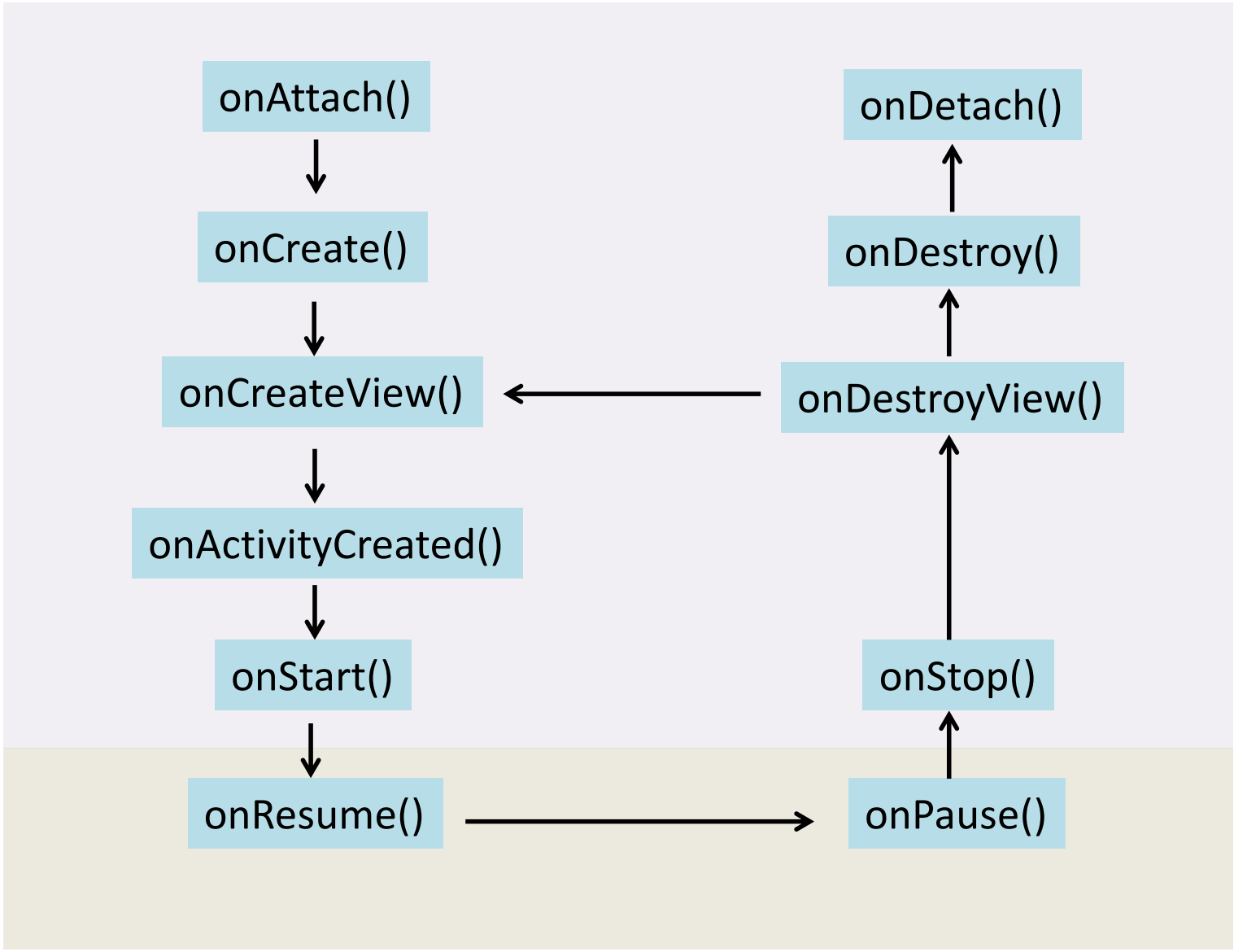
- Fragments are re-usable components hosted by an Activity
- Fragments do not subclass Context. Use context.getActivity() to get the hosting activity.

- Subclasses of Fragment class:
  - ListFragment
  - DialogFragment
  - PreferenceFragment
  - WebViewFragment

- Fragments should not communicate with each other directly. They should communicate through the host activity.

  - Therefore, fragments should define an interface and require that the hosting activity implements this interface
  - A Fragment can check whether a hosting activity correctly implements this interface in the Fragment's onAttach() method.

Fragment Life Cycle



- onAttach()
  - fragment instance associate with activity instance
- onCreate()
  - onCreate() of fragment is called after onCreate() of hosting activity
- onCreateView()
  - Fragment creates its user interface. inflate() the layout here.
  - Cannot interact with hosting activity yet, as the activity is not yet fully initialized
  - Note: no need to implement onCreateView() if the view is already defined in the hosting activity.

- onActivityCreated()
  - Called after onCreateView() **and** after the hosting activity is created.
  - You can now use Context (since the activity has been created)
- onStart()
  - Called once the fragment is visible
- onResume()
  - Fragment is now active
- onPause()
  - Fragment is visible but not active.

- onStop()
  - Fragment becomes invisible
- onDestroyView()
  - Destroys the fragment's view but the fragment is still in the stack
  - From onDestroyView(), can recreate the fragment relatively cheaply by calling onCreateView.
- onDestroy()

Important:

- If you want to attach fragments dynamically, make sure the fragment class import from

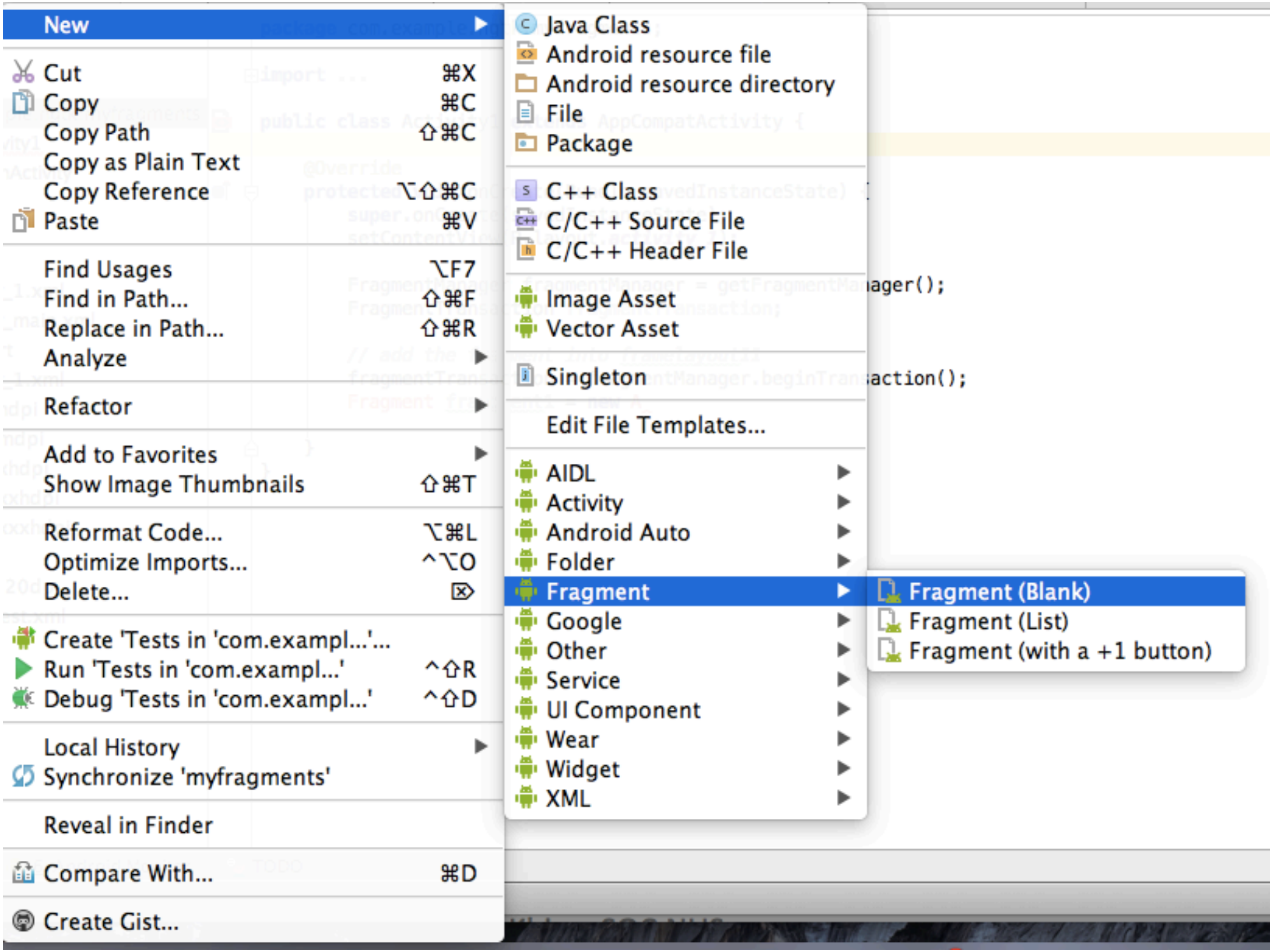
  import android.app.Fragment **and not**

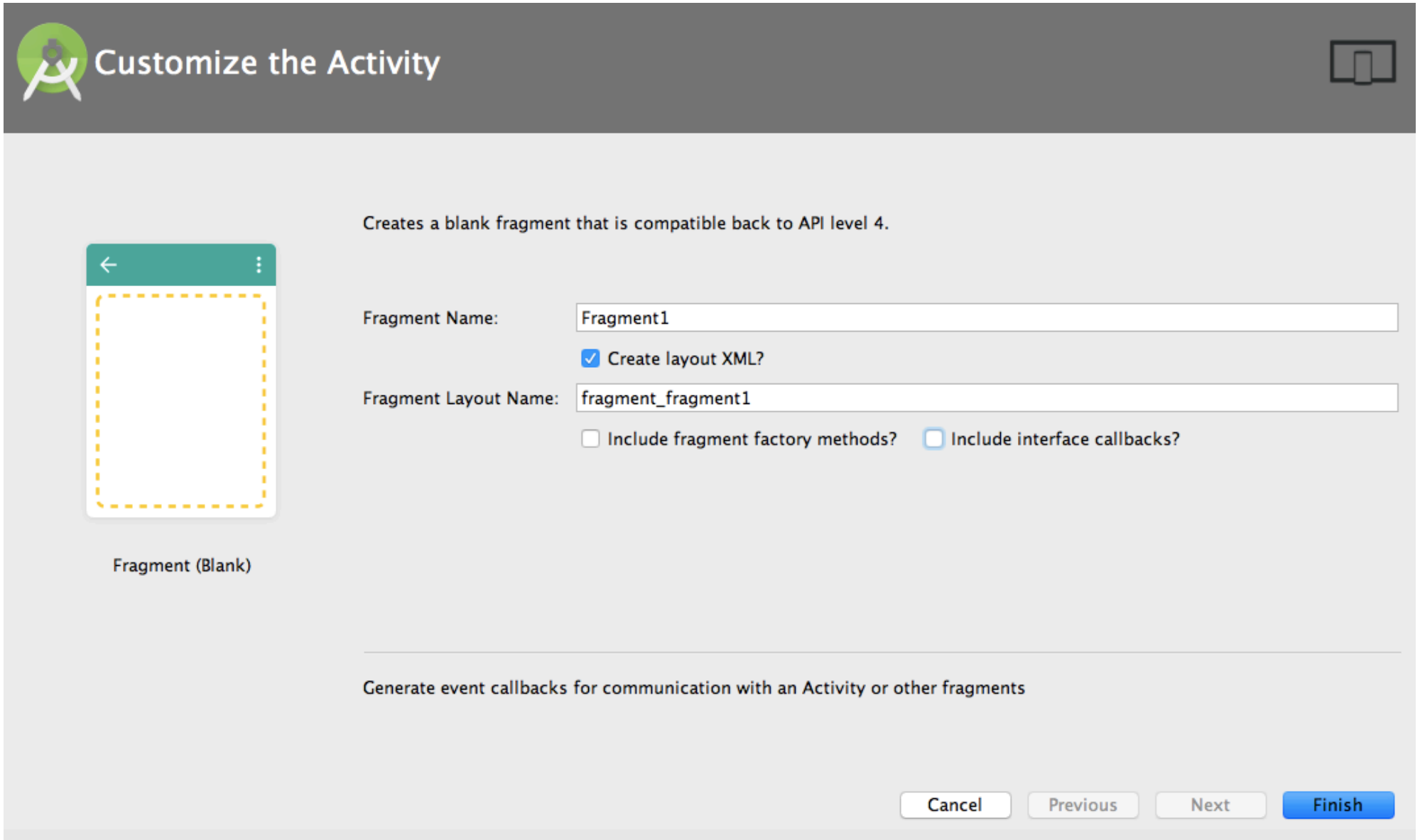
  import android.support.v4.app.Fragment (this could have been auto-imported)

Otherwise, the statement (in the activity that wants to attach the fragment dynamically) will complain of error:

```
fragmentTransaction.add(R.id.frameLayout21, new Fragment1());
```

Creating Fragments (empty fragment)


New
Cut ⌘X
Copy ⌘C
Copy Path ⇧⌘C
Copy as Plain Text
Copy Reference ⌥⇧⌘C
Paste ⌘V
Find Usages ⌥F7
Find in Path... ⇧⌘F
Replace in Path... ⇧⌘R
Analyze
Refactor
Add to Favorites
Show Image Thumbnails ⇧⌘T
Reformat Code... ⌥⌘L
Optimize Imports... ^⌥O
Delete...
Create 'Tests in 'com.exampl...'...
Run 'Tests in 'com.exampl...' ^⇧R
Debug 'Tests in 'com.exampl...' ^⇧D
Local History
Synchronize 'myfragments'
Reveal in Finder
Compare With... ⌘D
Create Gist...
Java Class
Android resource file
Android resource directory
File
Package
C++ Class
C/C++ Source File
C/C++ Header File
Image Asset
Vector Asset
Singleton
Edit File Templates...
AIDL
Activity
Android Auto
Folder
Fragment
Google
Other
Service
UI Component
Wear
Widget
XML
Fragment (Blank)
Fragment (List)
Fragment (with a +1 button)


Customize the Activity
Creates a blank fragment that is compatible back to API level 4.
Fragment Name:
Fragment1
Create layout XML?
Fragment Layout Name:
fragment_fragment1
Include fragment factory methods?
Include interface callbacks?
Fragment (Blank)
Generate event callbacks for communication with an Activity or other fragments
Cancel
Previous
Next
Finish

Defining Fragments Statically

- Let us create Activity3.java, which is a blank activity (without fragment)

- Insert the following in activity_3.xml to reuse the fragments created earlier:

```
<fragment
  android:layout_width="0dp"
  android:layout_height="match_parent"
  android:layout_weight="1"
  android:id="@+id/fragment31"
  android:name="com.example.ngtk.myfragments.Fragment1"
  android:layout="@layout/fragment_fragment1"/>

<fragment
  android:layout_width="0dp"
  android:layout_height="match_parent"
  android:layout_weight="1"
  android:id="@+id/fragment32"
  android:name="com.example.ngtk.myfragments.Fragment2"
  android:layout="@layout/fragment_fragment2"/>
```

Defining Fragments Dynamically

- We can “attach” fragments to activities dynamically.
- We will illustrate by creating a blank activity (i.e. without fragment) called Activity1 that will dynamically attach the two fragments that we created earlier.
- We will define two layout files:
  - activity_1.xml in res/layout
  - activity_1.xml in res/layout-port

Insert the following into activity_1.xml in layout folder:

```
<FrameLayout
   android:layout_width="0dp"
   android:layout_height="match_parent"
   android:layout_weight="1"
   android:id="@+id/frameLayout11">
</FrameLayout>

<FrameLayout
   android:layout_width="0dp"
   android:layout_height="match_parent"
   android:layout_weight="1"
   android:id="@+id/frameLayout12">
</FrameLayout>
```

Insert the following into activity_1.xml in layout-port folder:

```
<FrameLayout
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:id="@+id/frameLayout11">
</FrameLayout>
```

In onCreate() of Activity1.java, add the following after the default codes:

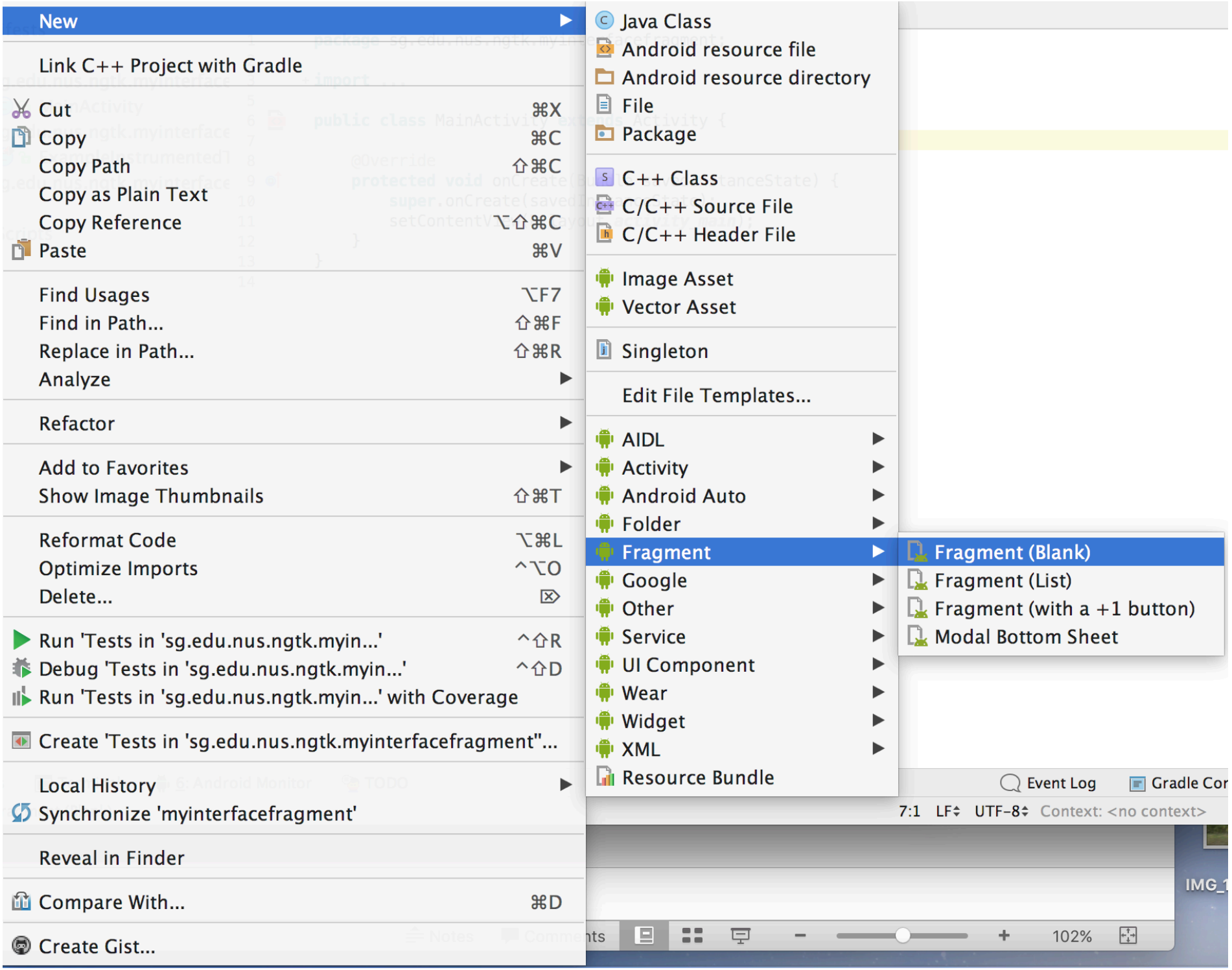
```
FragmentManager fragmentManager = getFragmentManager();
FragmentTransaction fragmentTransaction;

// add the fragment into framelayout11
fragmentTransaction = fragmentManager.beginTransaction();
Fragment fragment1 = new Fragment1();
fragmentTransaction.add(R.id.frameLayout11, fragment1);
fragmentTransaction.commit();

if (getResources().getConfiguration().orientation ==
Configuration.ORIENTATION_LANDSCAPE) {
   // add the fragment into framelayout12
   fragmentTransaction = fragmentManager.beginTransaction();
   Fragment fragment2 = new Fragment2();
   fragmentTransaction.add(R.id.frameLayout12, fragment2);
   fragmentTransaction.commit();
}
```

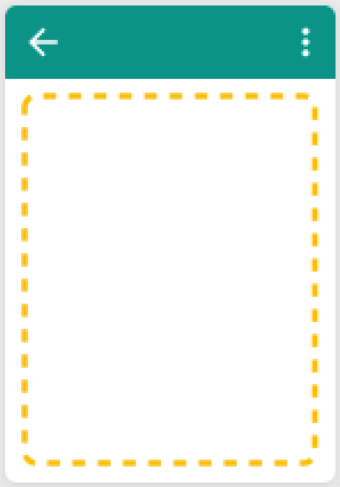
Note:

FragmentTransaction.commit() finishes off a transaction, so need to call fragmentManager.beginTransaction() before you can use it to add a new fragment and commit the addition.

Communicating between Fragments

- Fragments should be stand alone and modular. Fragments should not communicate with each other directly.
- Communication between fragments should be done through the hosting Activity.
- The hosting Activity should implement an interface to communicate with the fragments.


New
Link C++ Project with Gradle
Cut ⌘X
Copy ⌘C
Copy Path ⇧⌘C
Copy as Plain Text
Copy Reference ⌥⇧⌘C
Paste ⌘V
Find Usages ⌥F7
Find in Path... ⇧⌘F
Replace in Path... ⇧⌘R
Analyze
Refactor
Add to Favorites
Show Image Thumbnails ⇧⌘T
Reformat Code ⌥⌘L
Optimize Imports ^⌥O
Delete...
Run 'Tests in 'sg.edu.nus.ngtk.myin...' ^⇧R
Debug 'Tests in 'sg.edu.nus.ngtk.myin...' ^⇧D
Run 'Tests in 'sg.edu.nus.ngtk.myin...' with Coverage
Create 'Tests in 'sg.edu.nus.ngtk.myinterfacefragment"...
Local History
Synchronize 'myinterfacefragment'
Reveal in Finder
Compare With... ⌘D
Create Gist...
Java Class
Android resource file
Android resource directory
File
Package
C++ Class
C/C++ Source File
C/C++ Header File
Image Asset
Vector Asset
Singleton
Edit File Templates...
AIDL
Activity
Android Auto
Folder
Fragment
Google
Other
Service
UI Component
Wear
Widget
XML
Resource Bundle
Fragment (Blank)
Fragment (List)
Fragment (with a +1 button)
Modal Bottom Sheet
Event Log
Gradle Cor
7:1 LF UTF-8 Context: <no context>
102%



Configure Component

Android Studio

Creates a blank fragment that is compatible back to API level 4.

Fragment Name: FragmentWithInterface

☑ Create layout XML?

Fragment Layout Name: fragment_fragment_with_interface

☐ Include fragment factory methods?

☑ Include interface callbacks?

Generate static fragment factory methods for easy instantiation

Cancel | Previous | Next | Finish

```
import android.content.Context;
import android.net.Uri;
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
```

Change the import

```
import android.app.Fragment;
import android.content.Context;
import android.net.Uri;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
```

```java
public class FragmentWithInterface extends Fragment {

    private OnFragmentInteractionListener mListener;

    public FragmentWithInterface() {
        // Required empty public constructor
    }


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

    // TODO: Rename method, update argument and hook method into UI event
    public void onButtonPressed(Uri uri) {
        if (mListener != null) {
            mListener.onFragmentInteraction(uri);
        }
    }

    @Override
    public void onAttach(Context context) {
        super.onAttach(context);
        if (context instanceof OnFragmentInteractionListener) {
            mListener = (OnFragmentInteractionListener) context;
        } else {
            throw new RuntimeException(context.toString()
                    + " must implement OnFragmentInteractionListener");
        }
    }
}
```

Remove onButtonPressed (optional)

```
/**
 * This interface must be implemented by activities that contain this
 * fragment to allow an interaction in this fragment to be communicated
 * to the activity and potentially other fragments contained in that
 * activity.
 * <p>
 * See the Android Training lesson <a href=
 * "http://developer.android.com/training/basics/fragments/communicating.html"
 * >Communicating with Other Fragments</a> for more information.
 */
public interface OnFragmentInteractionListener {
    // TODO: Update argument type and name
    void onFragmentInteraction(Uri uri);
}
```

Any Activity that wishes to host this fragment must implement this interface

In the hosting Activity :

```
public class FragmentHostingActivity extends Activity implements
        FragmentWithInterface.OnFragmentInteractionListener {

                    •
                    •
                    •

 public void onFragmentInteraction(Uri uri) {

    }
}
```

If you add a button in a fragment layout, its callback function will, by default, handled by the hosting activity. This is not ideal, as we want the fragment to handle its own button callback i.e. we want the fragment to be standalone.

To do that, need to modify the default fragment java codes.

Default onCreateView

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

Change the onCreateView to the following:

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

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

    Button button1 = (Button)view.findViewById(R.id.buttonInFragment);
    button1.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            mListener.onFragmentInteraction("button is clicked!!!");

        }
    });

    return view;
}
```

id of button in layout xml file of fragment

String argument

```
public interface OnFragmentInteractionListener {
    // TODO: Update argument type and name
    void onFragmentInteraction(String str);
}
```

Change to String argument

In the hosting Activity :

```
public class FragmentHostingActivity extends Activity implements
      FragmentWithInterface.OnFragmentInteractionListener {

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

   public void onFragmentInteraction(String str) {
      Toast.makeText(this, str, Toast.LENGTH_SHORT).show();
   }
}
```