

Android Application Development



Session: 19

Activity

Objectives

- ☐ Describe Activities
- ☐ Explain life cycle of Activities
- ☐ Explain the steps to create class extends Activity

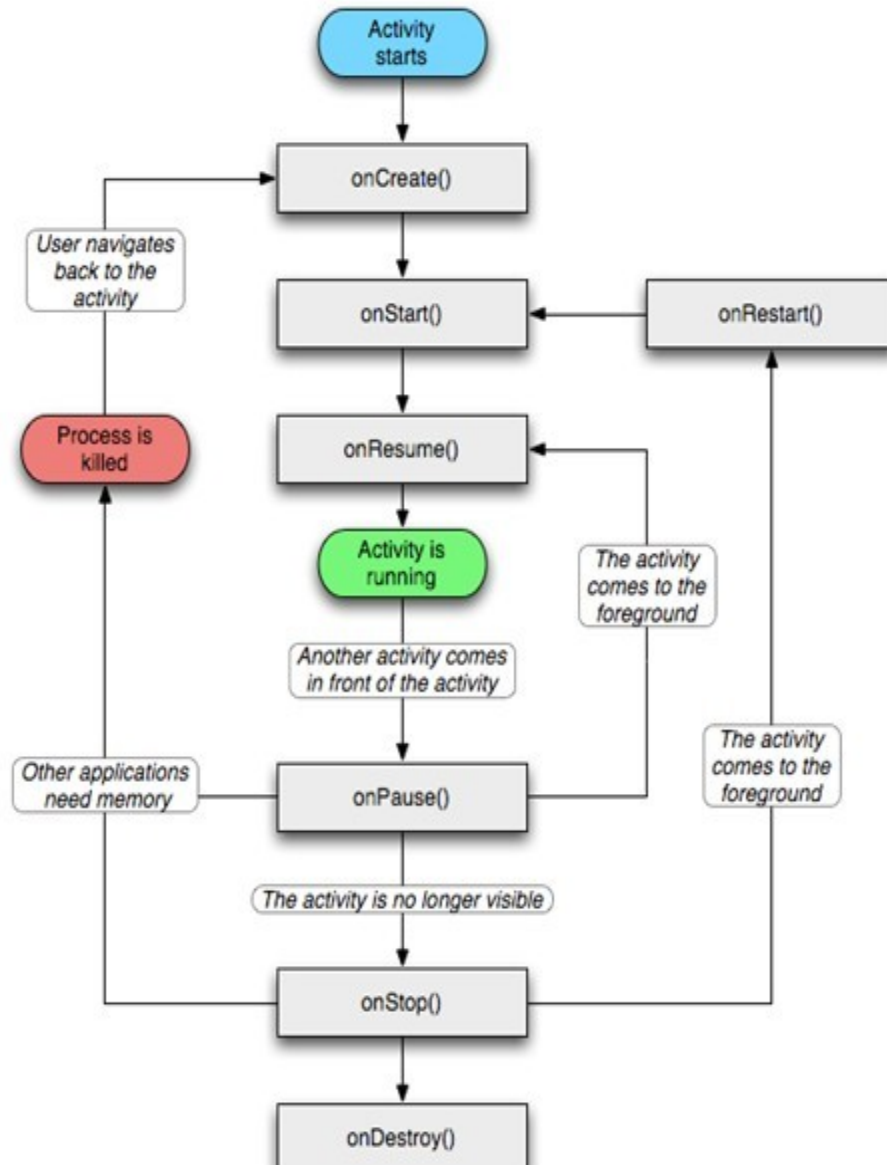
Activity

- An Activity is an application component that provides a screen with which users can interact in order to do something, such as dial the phone, take a photo, send an email, or view a map.
- Each activity is given a window in which to draw its user interface. The window typically fills the screen, but may be smaller than the screen and float on top of other windows.

Activity

- An application usually consists of multiple activities that are loosely bound to each other. Typically, one activity in an application is specified as the "main" activity, which is presented to the user when launching the application for the first time. Each activity can then start another activity in order to perform different actions.
- Each time a new activity starts, the previous activity is stopped, but the system preserves the activity in a stack .
- When a new activity starts, it is pushed onto the back stack and takes user focus.

Activity life cycle



Creating an Activity

- ¬ To create an activity, you must create a subclass of Activity (or an existing subclass of it). In your subclass, you need to implement callback methods that the system calls when the activity transitions between various states of its lifecycle, such as when the activity is being created, stopped, resumed, or destroyed. The two most important callback methods are:
 - ¬ onCreate()
 - ¬ onPause()

Methods in Activity

- `onCreate()` You must implement this method. The system calls this when creating your activity. Within your implementation, you should initialize the essential components of your activity. Most importantly, this is where you must call `setContentView()` to define the layout for the activity's user interface.
- `onPause()` The system calls this method as the first indication that the user is leaving your activity (though it does not always mean the activity is being destroyed). This is usually where you should commit any changes that should be persisted beyond the current user session .

Other Methods

- There are several other lifecycle callback methods that you should use in order to provide a fluid user experience between activities and handle unexpected interruptions that cause your activity to be stopped and even destroyed. All of the lifecycle callback methods discussed later, in the section about Managing the Activity Lifecycle .

Other Methods

Activity Lifecycle.

└onStart()

└onPause()

└onResume()

└onStop()

└onDestroy()

Sample Code

```
public class ExampleActivity extends Activity
{
    Public void onCreate(Bundle savedInstanceState){
        super.onCreate(savedInstanceState); // The activity is being created.
    }
    Protected void onStart(){
        super.onStart(); // The activity is about to become visible.
    }
    Protected void onPause(){
        super.onPause(); // Another activity is taking focus (this activity is
        about to be "paused").
    }
    Protected void onResume(){
        super.onResume(); // The activity has become visible (it is now
        "resumed").
    }
    Protected void onStop(){
        super.onPause(); // The activity is no longer visible (it is now
        "stopped")
    }
    Protected void onDestroy(){
        super.onPause(); // The activity is about to be destroyed.
    }
}
```

This is super class
For all classes

Declaring the activity in the manifest

- ─ You must declare your activity in the manifest file in order for it to be accessible to the system. To declare your activity, open your manifest file and add an `<activity>` element as a child of the

`<application>` element. For example:

```
<manifest ... >  
  <application ... >  
    <activity android: name=".ExampleActivity" />  
    ...  
  </application ... >  
  ...  
</manifest >
```

Using intent filters

- An `<activity>` element can also specify various intent filters—using the `<intent-filter>` element—in order to declare how other application components may activate it.
- When you create a new application using the Android SDK tools, the stub activity that's created for you automatically includes an intent filter that declares the activity responds to the "main" action and should be placed in the "launcher" category. The intent filter looks like this:

Using intent filters

```
<activity android:name=".ExampleActivity"  
  android:icon="@drawable/app_icon">  
  <intent-filter>  
    <action android:name="android.intent.action.MAIN" />  
    <category android:name="android.intent.category.LAUNCHER" />  
  </intent-filter>  
</activity>
```

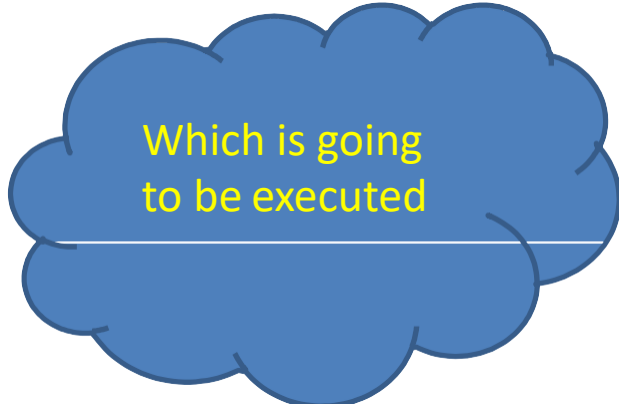
The `<action>` element specifies that this is the "main" entry point to the application.

The `<category>` element specifies that this activity should be listed in the system's application launcher (to allow users to launch this activity).

Starting an Activity

- You can start another activity by calling `startActivity()`, passing it an `Intent` that describes the activity you want to start. An intent can also carry small amounts of data to be used by the activity that is started.

```
Intent intent = new Intent(this, Next.class);  
intent.putExtra("key", value);  
startActivity(intent);
```



Which is going
to be executed



Fragment

How to work with fragments

An Introduction to fragments

A fragment is a class that you can use to define part of a user interface. Then, you can use an activity to display one or more fragments.

Android introduced fragments in version 3.0 (API 11). You can use fragments to create user interfaces that work well on both small and large screens.

Single-panel and multi-panel layout

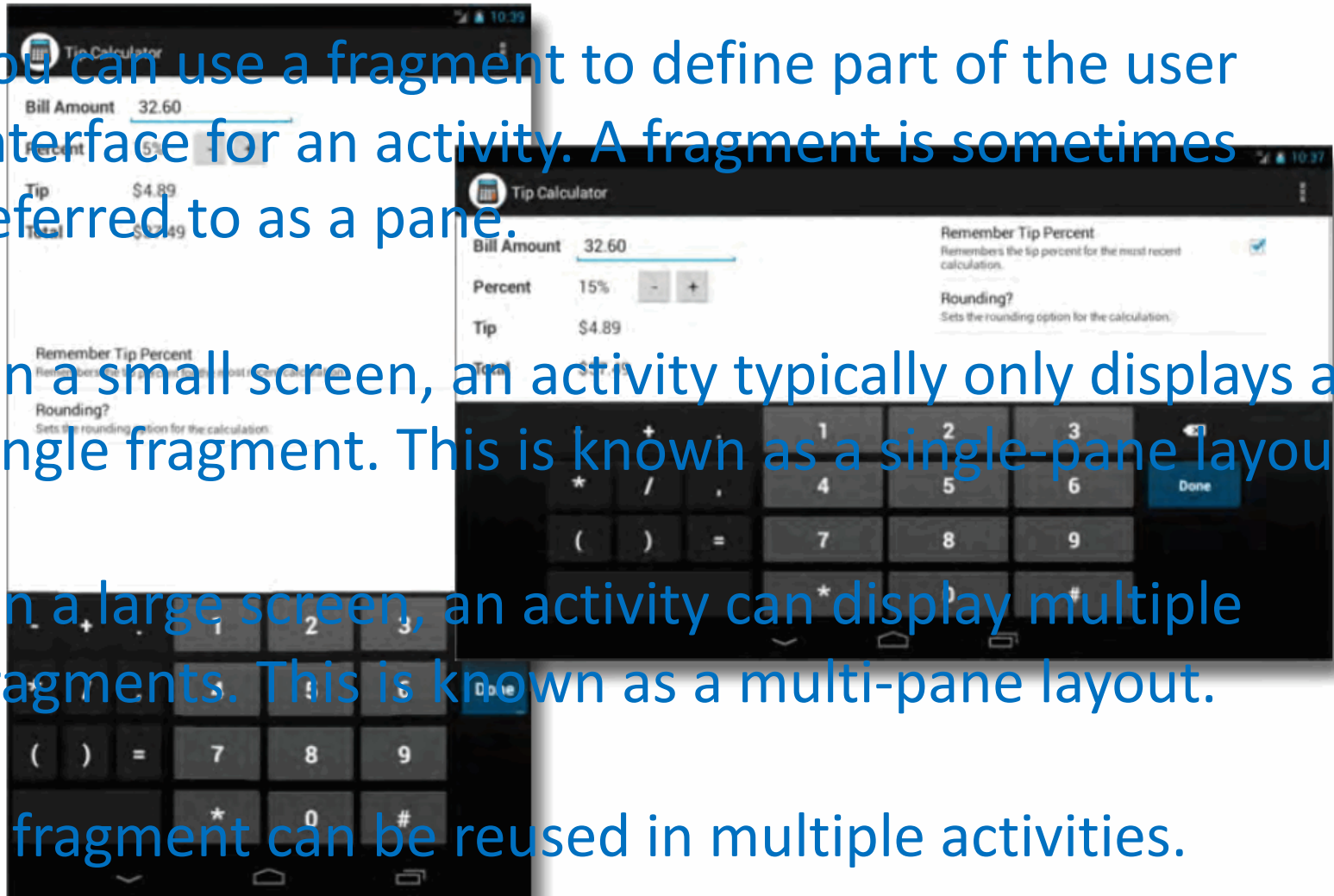
One activity displaying two fragments in both landscape and portrait orientation

You can use a fragment to define part of the user interface for an activity. A fragment is sometimes referred to as a pane.

On a small screen, an activity typically only displays a single fragment. This is known as a single-pane layout.

On a large screen, an activity can display multiple fragments. This is known as a multi-pane layout.

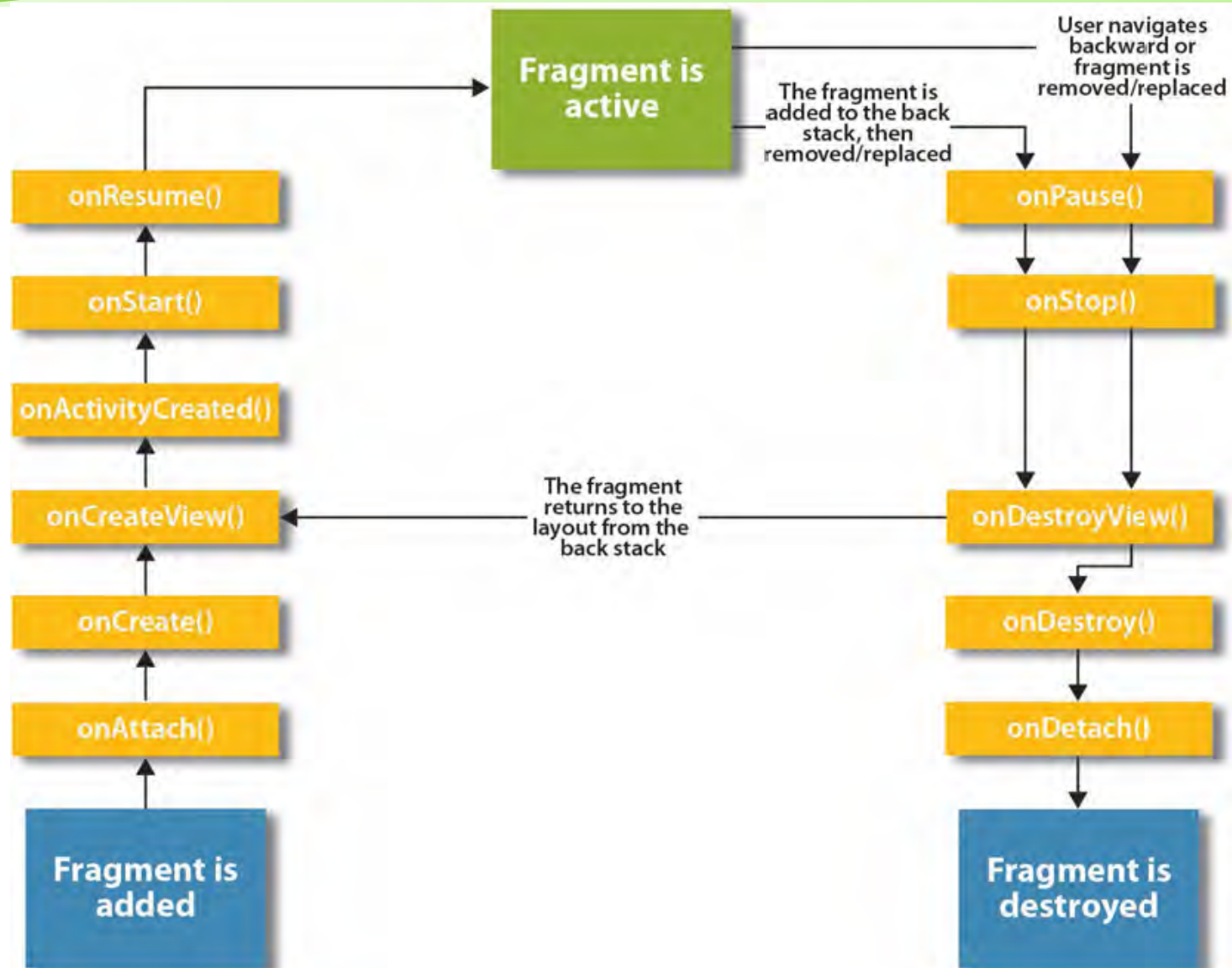
A fragment can be reused in multiple activities.



How to use support libraries

- Fragments are available from Android 3.0 (API 11) and higher.
- **Android provides support libraries that you can add to your project. These libraries make new APIs such as fragments available on older versions of Android.**
 - 1. Start the Android SDK Manager as described in appendix A (PC) or B (Mac).
 - 2. Expand the Extras category and view the Android Support Library package. If the package isn't already installed, install it. This downloads all support files into the directory shown below.
- **The SDK directory that contains the Android support library**
 - `\android-sdk\extras\android\support\`
- **How to make the support library available to a project**
 - 3. Locate the JAR file for the support library and copy it into your project's `libs` directory.

The lifecycle methods of a fragment

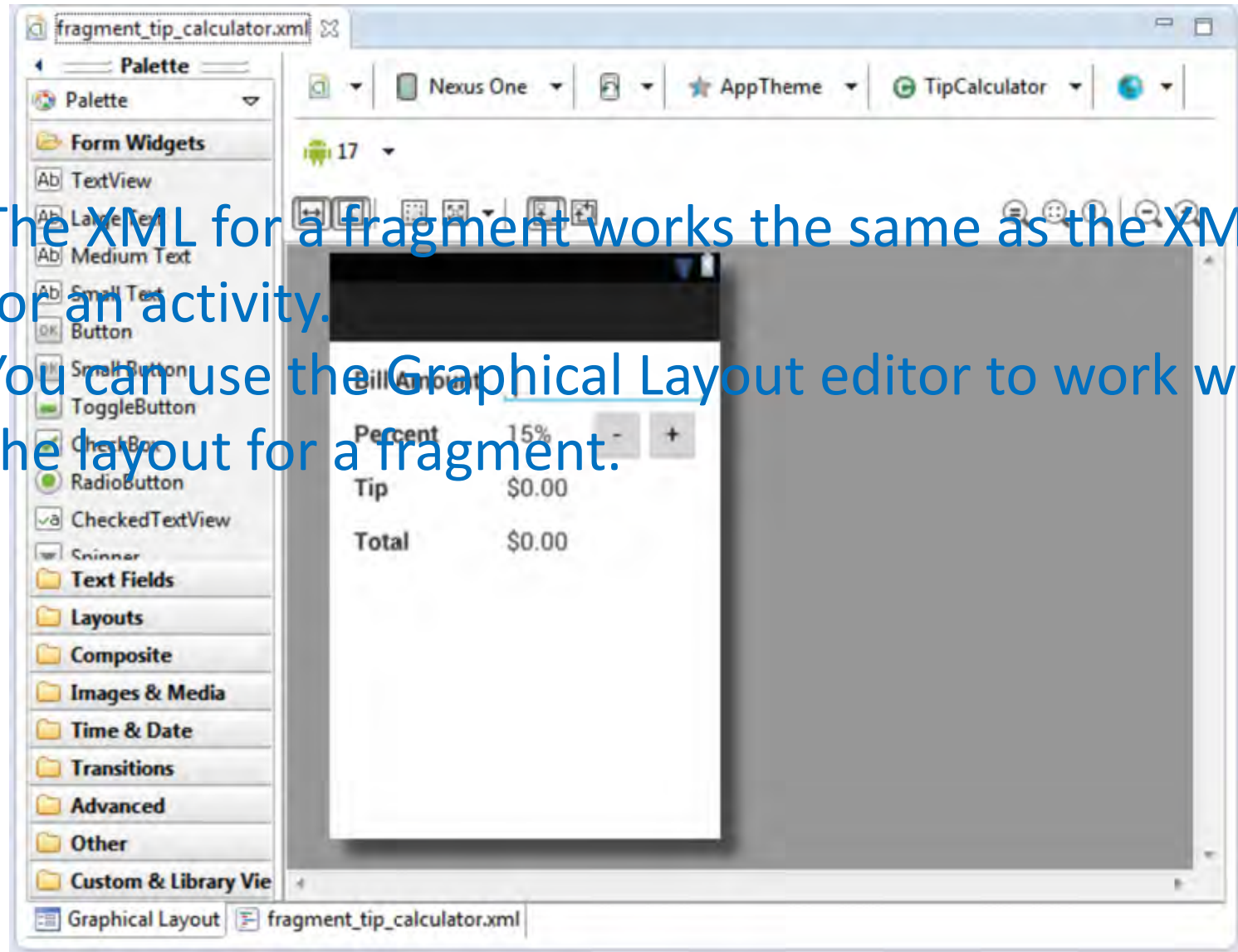


How to use single-pane layouts for small screens

This type of layout is useful for small screen devices such as phones. Since many devices have small screens, you often begin by creating a single-pane layout. Then, once you have that layout working correctly, you can add one or more multi-pane layouts for devices that have larger screens.

How to create the layout for a fragment

- The XML for a fragment works the same as the XML for an activity.
- You can use the Graphical Layout editor to work with the layout for a fragment.



How to create the class for a fragment

The onCreateView method

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

```
    // Inflate the layout for this fragment
```

Description

- The Java code for a fragment works much like the Java code for an activity. However, there are several differences, especially in the onCreate and onCreateView methods.

```
        view.findViewById(R.id.billAmountEditText);  
  
        // set the listeners  
        billAmountEditText.setOnEditorActionListener(this);  
  
        // return the View for the layout  
        return view;  
    }
```

How to display a fragment in an activity

The TipCalculatorActivity class

```
package com.murach.tipcalculator;

import android.app.Activity;
import android.os.Bundle;

public class TipCalculatorActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

Description

- To add a fragment to a layout, add a fragment element and use its name attribute to specify the fully qualified name for the class that defines the fragment.

How to create a preference fragment

Figure 9-7 shows how to create a fragment that displays preferences. Since this user interface is built using Preference objects instead of View objects, your fragment needs to extend the PreferenceFragment class instead of the Fragment class.

The first code example shows a fragment named SettingsFragment that inherits the PreferenceFragment class. This code is the same as the code for the SettingsFragment class shown in the previous chapter. As a result, you should already understand how it works.

How to display a preference fragment in an activity

The SettingsActivity class

```
package com.murach.tipcalculator;

import android.app.Activity;
import android.os.Bundle;

public class SettingsActivity extends Activity {

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

        // set the view for the activity using XML
        setContentView(R.layout.activity_settings);
    }
}
```

Description

- Since the user interface is built using Preference objects instead of View objects, your fragment needs to extend the PreferenceFragment class instead of the Fragment class. Then, it can use the addPreferencesFromResource method to add the preferences defined in the XML file to the fragment.

How to use multi-pane layouts for large screens

Now that you understand how to use fragments in a single-pane layout, you're ready to learn how to use them in a multi-pane layout. This type of layout is commonly used for devices that have large screens such as tablets.

How to add multiple fragments to a layout

res\layout\activity_main_twopane_port.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <fragment android:name="com.murach.tipcalculator.TipCalculatorFragment"
        android:id="@+id/main_fragment"
        android:layout_weight="1"
        android:layout_height="0dp"
        android:layout_width="match_parent" />

    <fragment android:name="com.murach.tipcalculator.SettingsFragment"
        android:id="@+id/settings_fragment"
        android:layout_weight="1"
        android:layout_width="match_parent"
        android:layout_height="0dp" />

</LinearLayout>
```

How to detect large screens

The layout files for devices with large screens

- For devices with small screens, Android uses a layout in the layout directory.
- To detect large screens, you can create a values directory that uses the large or xlarge qualifiers.
- To detect landscape and portrait orientations, you can create a values directory that uses the land or port qualifiers.
- Within a values directory, you can use an *alias* to point to a layout file that's stored in a layout directory. This provides a way to avoid duplicating code in multiple layout files.
- To specify an alias for a layout, code an item element. Then, set its name attribute to the name of the layout that you want to replace and set its type attribute to a value of "layout". Next, use the body of the element to point to the correct XML file in the project's layout directory.
- The large and xlarge qualifiers work with Android versions 3.0 and later. As a result, they aren't available from earlier versions of Android.

</resources>

How to detect screen width

The layout files for devices with a minimum screen size

Description

- The smallest-width qualifier allows you to select screens whose smallest width is at least as wide as specified width.
- To detect devices with screens that have a minimum width, you can create a values directory that uses the smallest-width qualifier. For example, a qualifier of sw600dp specifies a device that has a minimum smallest width of 600dp.
- The smallest-width qualifier was introduced in Android 3.2 (API 13). As a result, it isn't available with earlier versions of Android.
- Since earlier versions of Android don't recognize the smallest-width qualifier, you typically use the large qualifier as well.

```
<item name="activity_main" type="layout" /
    @layout/activity_main_twopane_port
</item>
</resources>
```

How to control the soft keyboard

Attributes of an EditText widget that can control the soft keyboard

```
<EditText
    android:id="@+id/billAmountEditText"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBaseline="@+id/billAmountLabel"
    android:layout_marginLeft="5dp"
    android:layout_toRightOf="@+id/billAmountLabel"
    android:ems="8"
    android:inputType="numberDecimal"
    android:imeOptions="actionDone"
    android:text="@string/bill_amount"
    android:textSize="20sp" >

    <requestFocus />
</EditText>
```

Description

- If you don't want Android to display the soft keyboard when the app starts, you can delete the requestFocus element from the body of the EditText element.
- If Android does not display the correct action button on the soft keyboard, you can use the imeOptions attribute of the EditText element to specify the correct action button on the soft keyboard.

Other skills for working with fragments

So far, this chapter has presented the skills that you typically need for working with fragments. However, in some situations, you may need the skills presented in the next two figures to work with fragments.

How to get a reference to a fragment

The onSharedPreferencesChanged method of the Settings fragment

```
@Override
public void onSharedPreferencesChanged(SharedPreferences prefs,
    String key) {

    // attempt to get the fragment
    TipCalculatorFragment tipFragment =
        (TipCalculatorFragment) getFragmentManager()
            .findFragmentById(R.id.main_fragment);

    // if the fragment is not null, call a method from it
    if (tipFragment != null) {
        tipFragment.onResume();
    }
}
```


How to replace one fragment with other

The SettingsActivity class

```
package com.murach.tipcalculator;

import android.app.Activity;
import android.os.Bundle;

public class SettingsActivity extends Activity {

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

        // display the fragment as the main content
        getFragmentManager().beginTransaction()
            .replace(android.R.id.content, new SettingsFragment())
            .commit();
    }
}
```

Discuss and QA