ANDROID APPS DEVELOPMENT FOR MOBILE AND TABLET DEVICE (LEVEL II)

Lecture 1: Content Provider and Fragment

Peter Lo

Who am I?

Lo Chi Wing, Peter

- **Email:** Peter@Peter-Lo.com
- **Facebook:** http://www.facebook.com/PeterLo111

Course Outline

Lesson 1	Content Provider and Fragment
Lesson 2	Data Storage
Lesson 3	Location Based Services
Lesson 4	Multimedia and Open GL
Lesson 5	Sensor and Game Development
Lesson 6	Web Services

Are you take the Right Course?

- This course is intended for experienced Android programmers who would like to enrich their mobile development skill.
- The code examples are not particularly complex, but it is assumed that you have basic Android programming knowledge.

Where can you find the material?

- Workshop Notes and Exercises
  - http://www.Peter-Lo.com/Teaching/4T025-2-A/
- Android Developer Official Site
  - http://developer.android.com/
- Microsoft DreamSpark
  - http://www.dreamspark.com

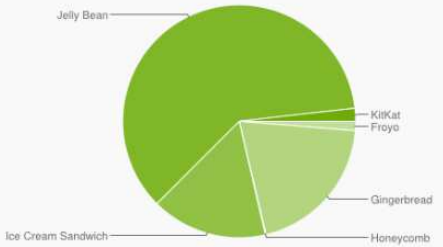
What is Android?

- A Linux-based operating system for mobile devices.
- An open-source project and is distributed free of charge.
- Developed by the Open Handset Alliance and Google Inc.
- Supporting telephony, messaging, emailing, contact management, calendar, entertainment, multimedia experience, location services, mapping, social interaction, etc.

Android Family

Platform Versions

Version	Codename	API	Distribution
2.2	Froyo	8	1.3%
2.3.3 - 2.3.7	Gingerbread	10	20.0%
3.2	Honeycomb	13	0.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	16.1%
4.1.x	Jelly Bean	16	35.5%
4.2.x		17	16.3%
4.3		18	8.9%
4.4	KitKat	19	1.8%

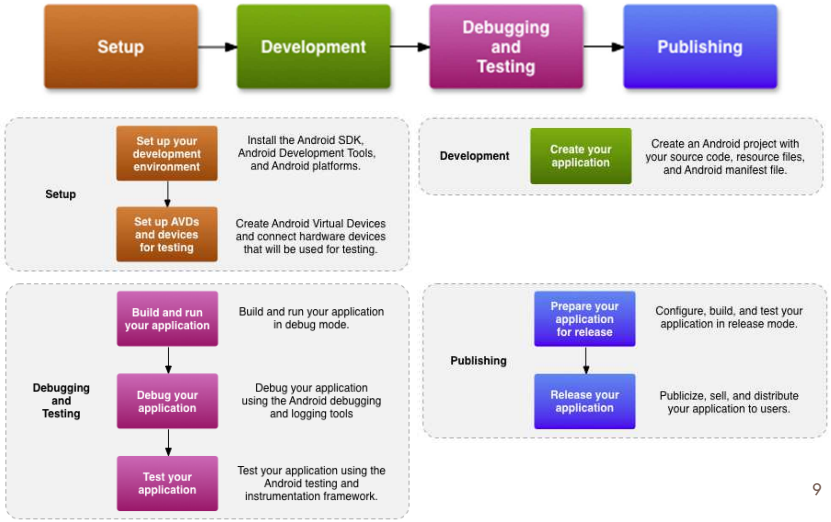


Data collected during a 7-day period ending on February 4, 2014.
Any versions with less than 0.1% distribution are not shown.

Note: Because this data is gathered from the new Google Play Store app, which supports Android 2.2 and above, devices running older versions are not included. However, in August, 2013, versions older than Android 2.2 accounted for about 1% of devices that *checked in* to Google servers (not those that actually visited Google Play Store).

Source: http://developer.android.com/about/dashboards/index.html

Development Process for Android Apps



Application Components

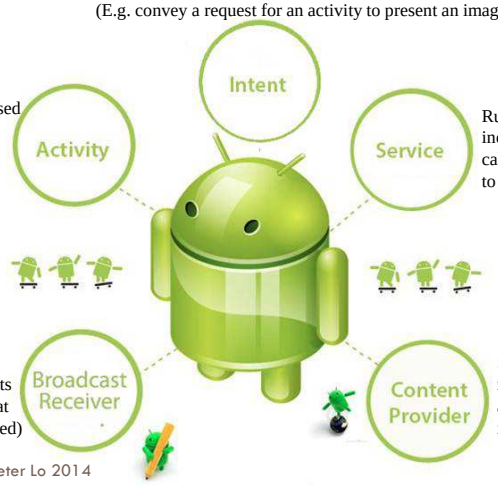
Hold the content of a message
(E.g. convey a request for an activity to present an image)

Present a visual user interface for one focused endeavor the user can undertake (E.g. list of menu items)

Run in the background for an indefinite period of time (E.g. calculate and provide the result to activities that need it)

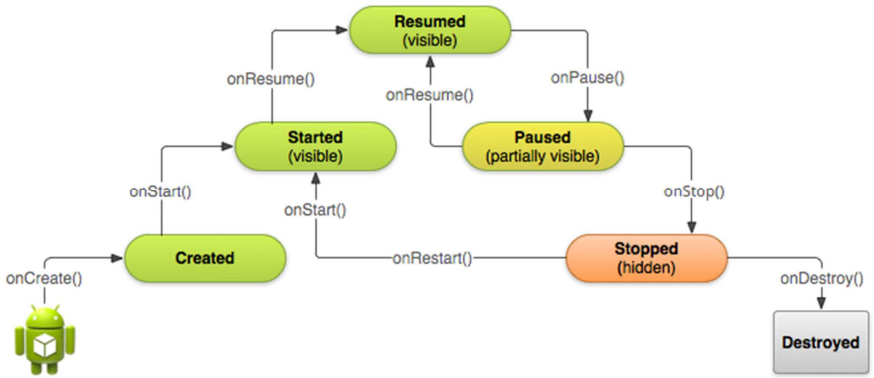
Receive and react to broadcast announcements (E.g. announcements that the time zone has changed)

Store and retrieve data and make it accessible to all applications (E.g. audio, video, images, etc.)



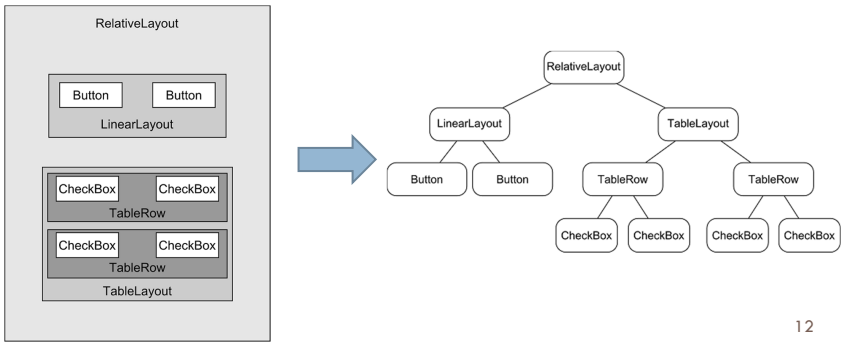
Android Lifecycle

- An activity begins its lifecycle when entering the ***onCreate()*** state
- If not interrupted or dismissed, the activity performs its job and finally terminates and releases its acquired resources when reaching the ***onDestroy()*** event



The View Hierarchy

- The view hierarchy diagram gives probably the clearest overview of the relationship between the various views that make up the user interface shown. When a user interface is displayed to the user, the Android runtime walks the view hierarchy, starting at the root view and working down the tree as it renders each view.

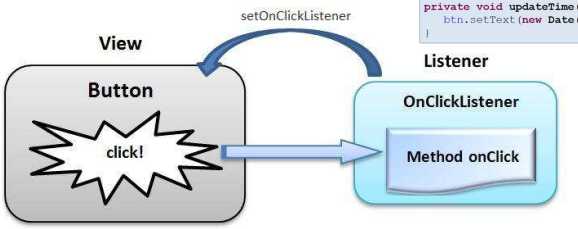


Event Listener

- An object cannot process event on its own, it needs a listener for this.
  - E.g. When a button is clicked, listener reacts and runs the code from **onClick** method.

```
btn.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        updateTime();
    }
});

private void updateTime() {
    btn.setText(new Date().toString());
}
```

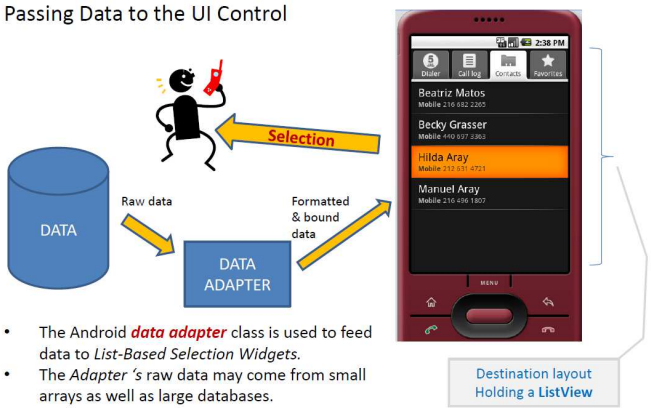


Common Event

- onClick()
- onLongClick()
- onFocusChange()
- onKey()
- onTouch()
- onCreateContextMenu()

ListView

- ListView provides a simple way to present a scrolling list of rows with a built-in style or customized format extensively.
- It requires an adapter to feed it with data contained in row views.



- The Android ***data adapter*** class is used to feed data to *List-Based Selection Widgets.*
- The *Adapter 's* raw data may come from small arrays as well as large databases.

Creating Simple List

- *ListActivity* makes it simple to create an activity whose main purpose is to show a list of elements.

```
public class MainActivity extends ListActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        String ListItems[] = { "Item 1", "Item 2", "Item 3", "Item 4"};

        ArrayAdapter<String> adapter = new ArrayAdapter<String>
            (this,android.R.layout.simple_list_item_1, ListItems);

        setListAdapter(adapter);
    }

    protected void onListItemClick(ListView list, View view, int position, long id) {
        switch(position) {
            case 0:
                // Do something
            case 1:
                // Do something
            default:
                // Handle other case
        }
    }
}
```

Extends ListActivity rather than Activity

Defined Array values

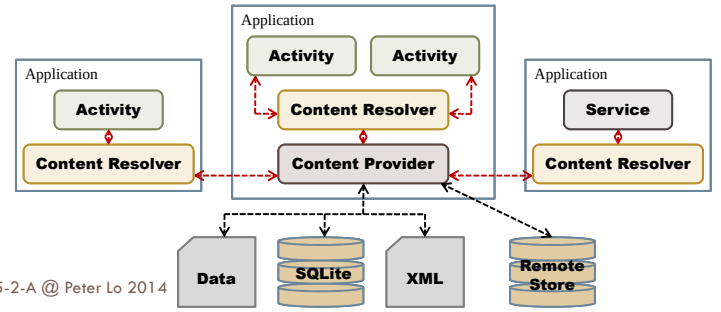
Define a new Adapter

Assign adapter to ListView

Response to user selection

Content Provider

- Android provides a way for you to expose even your private data to other applications with a content provider.
- A content provider is an optional component that exposes read/write access to your application data, subject to whatever restrictions you want to impose.



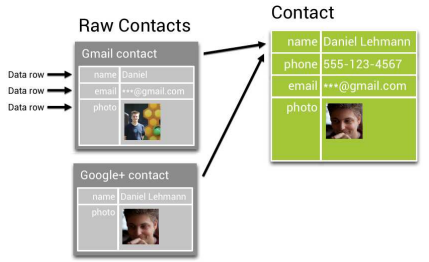
Content Provider Overview

- A content provider presents data to external applications as tables.
- A row represents an instance of some type of data the provider collects, and each column in the row represents an individual piece of data collected for an instance.
  - E.g. one of the built-in providers in the Android platform is the user dictionary, which stores the spellings of non-standard words that the user wants to keep.

word	app id	frequency	locale	_ID
mapreduce	user1	100	en_US	1
precompiler	user14	200	fr_FR	2
applet	user2	225	fr_CA	3
const	user1	255	pt_BR	4
int	user5	100	en_UK	5

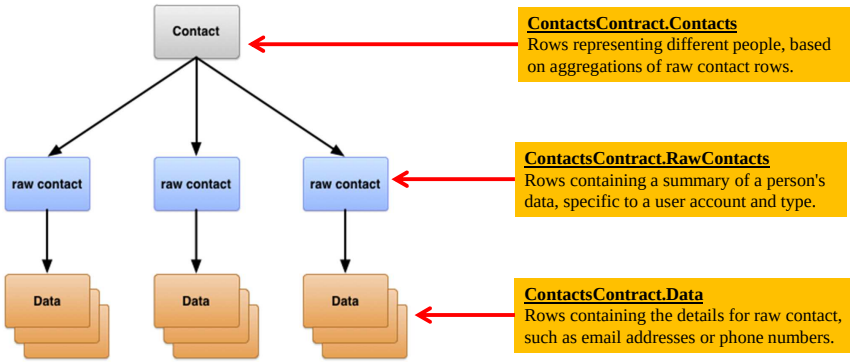
Contacts Provider

- Contacts Provider is a powerful and flexible Android component that manages the device's central repository of data about people.
- Contacts Provider is the source of data you see in the device's contacts application, and you can also access its data in your own application and transfer data between the device and online services.



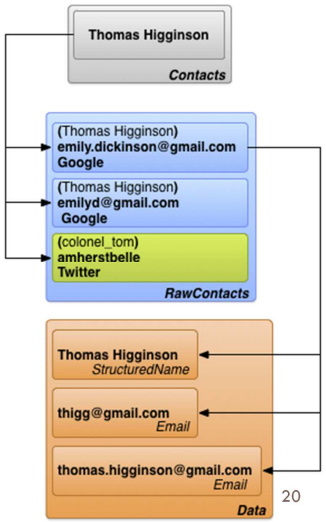
Contacts Provider Table Structure.

- The Contacts Provider is an Android content provider component.
- It maintains three types of data about a person, each of which corresponds to a table offered by the provider



Contacts, Raw Contacts, and Details table Relationships

- The Contacts Provider links a contact row to its raw contact rows with the contact row's *_ID* column in the Contacts table.
  - The *CONTACT_ID* column of the table *ContactsContract.RawContacts* contains *_ID* values for the contacts row associated with each raw contacts row.
- The *ContactsContract.Contacts* table also has the column LOOKUP_KEY that is a "permanent" link to the contact row.
  - Because the Contacts Provider maintains contacts automatically, it may change a contact row's *_ID* value in response to an aggregation or sync.



Request Permissions

- To read from the Contacts Provider, your app must have *READ_CONTACTS* permission.
- To request this permission, add the following child element of *<manifest>* to your manifest file:

```
<uses-permission android:name="android.permission.READ_CONTACTS" />
```


Retrieving a List of Contacts

- To retrieve the contact row that contains the user's profile, call *ContentResolver.query()*. Set the content URI to *CONTENT_URI* and don't provide any selection criteria.

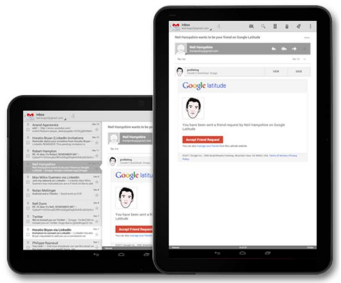
Construct the Content Resolver for accessing Content Provider

```
ContentResolver contentResolver = getContentResolver();
Cursor cursor = contentResolver.query(
        ContactsContract.Contacts.CONTENT_URI, null, null, null, null);
```

Query the content provider, returning a cursor over the result set.
- *Uri* – The URI for the content to retrieve.
- *projection* – A list columns to return. Passing null will return all columns.
- *selection* – Define 'where' part of query. Passing null will return all rows.
- *selectionArgs* – Specify arguments in placeholder order
- *sortOrder* – Define the sorting order for the results.

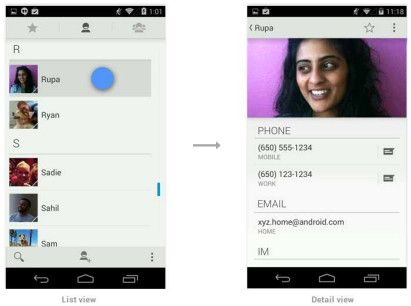
Multi-pane Layouts

- Android devices come in many different screen sizes and types.
- The app should provides a balanced and aesthetically pleasing layout consistently by adjusting its content to varying screen sizes and orientations.
- Panels are a great way for your app to achieve this by combining multiple views into one compound view when a lot of horizontal screen real estate is available and by splitting them up when less space is available



Multiple Views on Handset Device

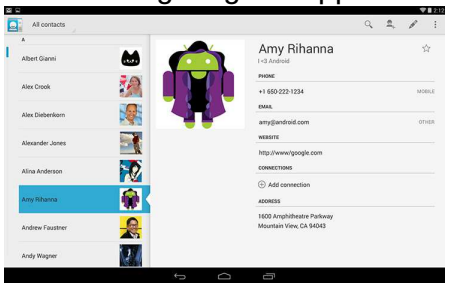
- On smaller devices your content is typically divided into a master grid or list view and a detail view.
- Touching an item in the master view opens a different screen showing that item's detail information.



List view Detail view

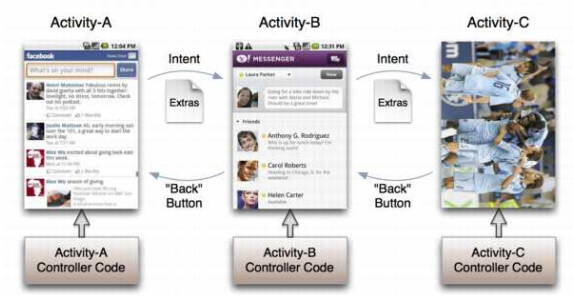
Multiple Views on Tablet Device

- Tablets have more screen real estate than phones, you can use panels to combine the related list and detail views into a single compound view.
- This uses the additional space more efficiently and makes navigating the app easier.



Fragments

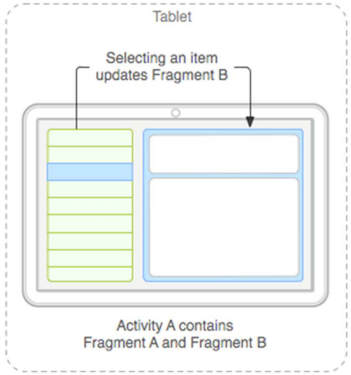
- A Fragment represents a behavior or a portion of user interface in an Activity. Can combine multiple fragments in a single activity to build a multi-pane UI and reuse a fragment in multiple activities.



You can think of a fragment as 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

Approach for Creating Fragment

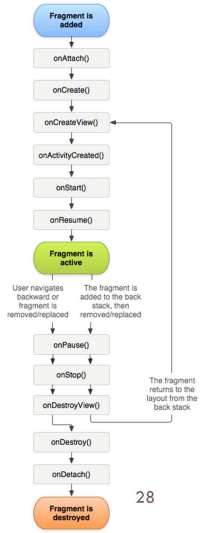
Use One Activities



Use Separate Activities

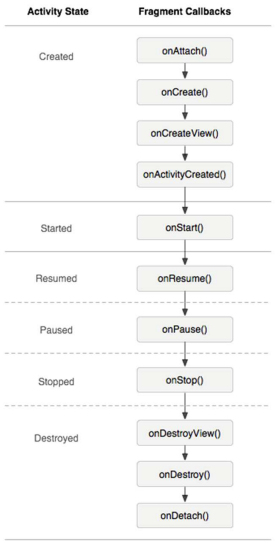
Fragment Lifecycle

- You should implement at least the following lifecycle methods:
  - **onCreate()** – The system calls this when creating the fragment. 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.
  - **onPause()** – The system calls this method as the first indication that the user is leaving the fragment. This is usually where you should commit any changes that should be persisted beyond the current user session.



Handling the Fragment Lifecycle

- Managing the lifecycle of a fragment is a lot like managing the lifecycle of an activity. Like an activity, a fragment can exist in three states:
  - **Resumed** – The fragment is visible in the running activity.
  - **Paused** – Another activity is in the foreground and has focus, but the activity in which this fragment lives is still visible.
  - **Stopped** – The fragment is not visible. Either the host activity has been stopped or the fragment has been removed from the activity but added to the back stack. A stopped fragment is still alive. However, it is no longer visible to the user and will be killed if the activity is killed.



Create a Fragment Class

- To provide a layout for a fragment, you must implement the *onCreateView()* callback method, which the Android system calls when it's time for the fragment to draw its layout.

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

public class TestingFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {

        return inflater.inflate(R.layout.fragment_detail, container, false);
    }
}
```

To create a fragment, extend the Fragment class, then override key lifecycle methods to insert your app logic, similar to the way you would with an Activity class

You must use the onCreateView() callback to define the layout. This is the only callback you need in order to get a fragment running

To return a layout from onCreateView(), you can inflate it from a layout resource defined in XML. To help you do so, onCreateView() provides a LayoutInflater object

Communicating between Fragments

- Often you will want one Fragment to communicate with another, for example to change the content based on a user event.
- All Fragment-to-Fragment communication is done through the associated Activity.
- Two Fragments should never communicate directly.