APPFORUM2013

APPFORUM2013

Android Fundamentals

Ritesh Gupta

Agenda

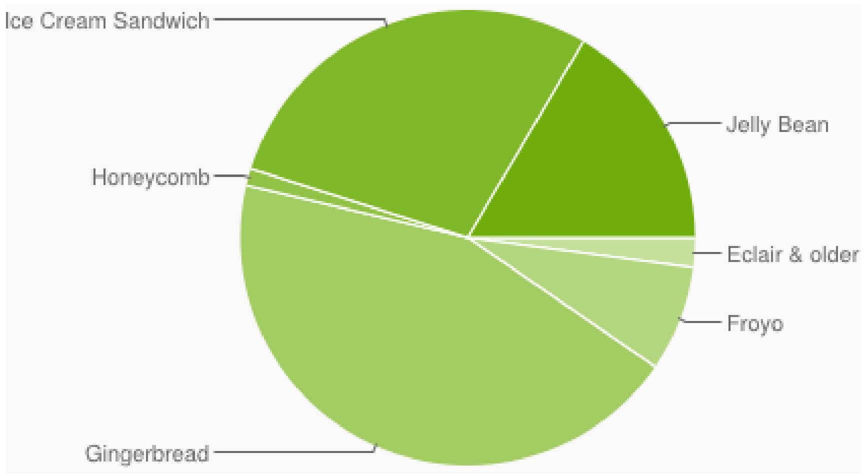
- What is Android
- Why Android
- Android Architecture
  - Applications
  - Application Framework
  - Android Runtime
  - Native Libraries
  - Linux Kernel
- Application Security Sandbox
- APK files
- Manifest File
- Activity
- Intent
- Service
- Broadcast Receiver
- Notification
- Content Provider

What is Android

- Software stack for mobile devices that includes an OS, middleware, and key applications
- Linux-based Operating System
    - "adb shell" command opens Linux shell
- Open source

Android Versions

Version	Codename	API	Distribution	Release Date
1.6	Donut	4	0.1%	September 15, 2009
2.1	Eclair	7	1.7%	January 12, 2010
2.2	Froyo	8	4.0%	May 20, 2010
2.3 - 2.3.2		9	0.1%	December 6, 2010
2.3.3 - 2.3.7	Gingerbread	10	39.7%	February 9, 2011
3.2	Honeycomb	13	0.2%	July 15, 2011
4.0.3 - 4.0.4	Ice Cream Sandwich	15	29.3%	December 16, 2011
4.1.x		16	23.0%	July 9, 2012
4.2.x	Jelly Bean	17	2.0%	November 13, 2012



Data collected over a 14 day period ending on April 2, 2013.

From http://developer.android.com

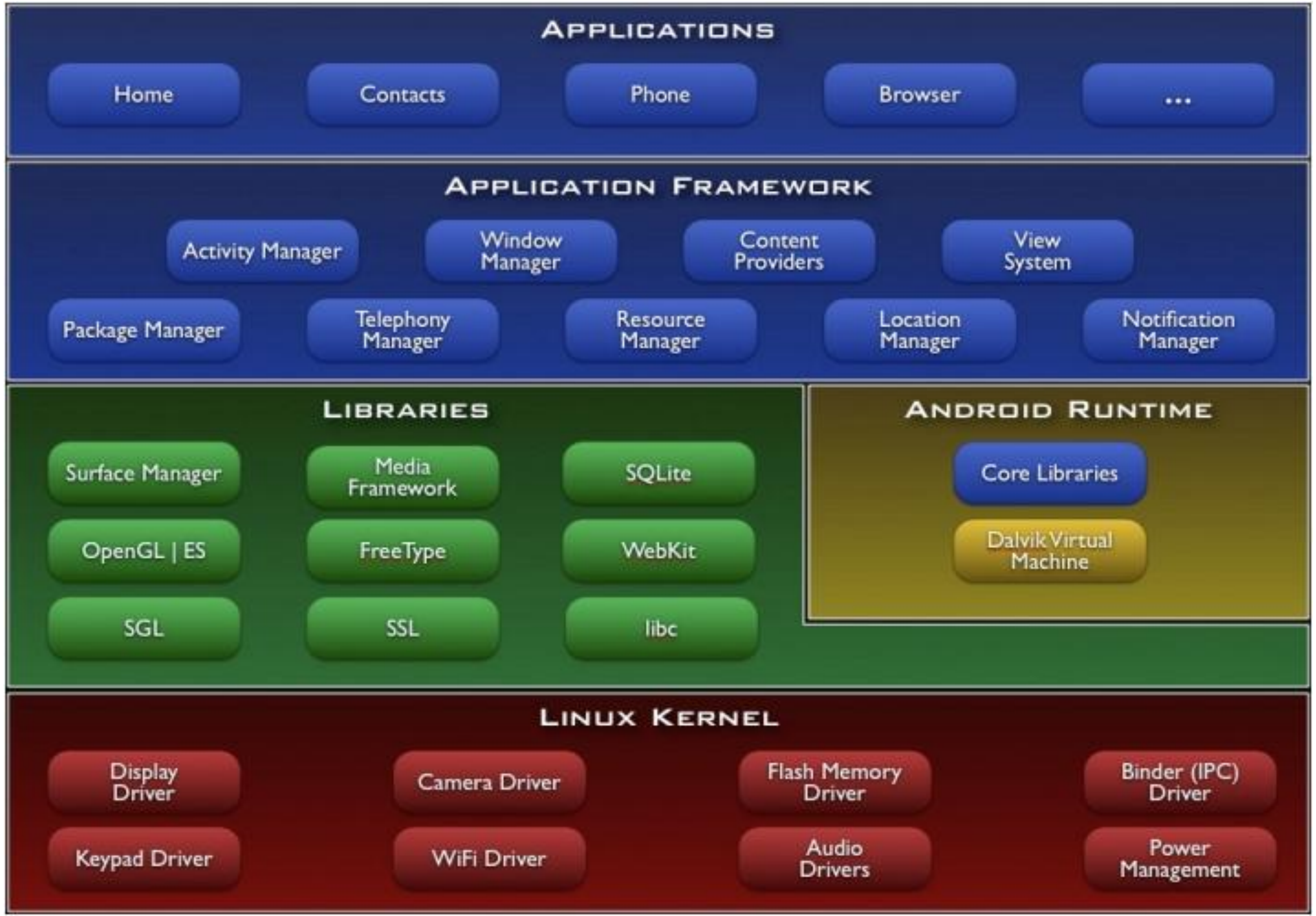
Why Android?

- Robust – based on Linux 2.6
- Flexibility
- Security
- Open source – developer community support
- Cost

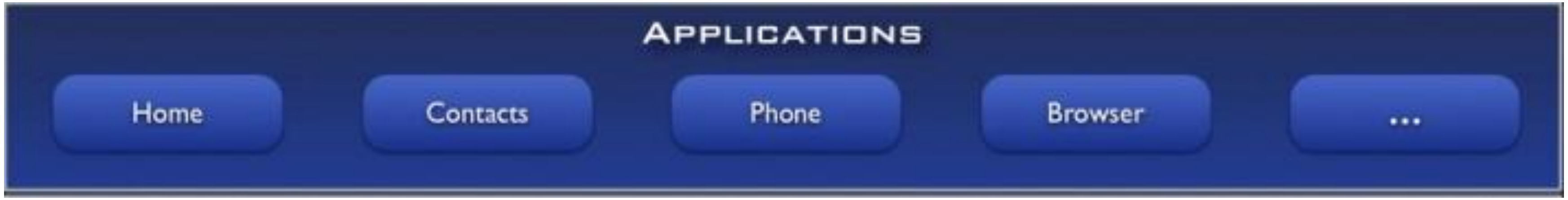
APPFORUM2013

Android Architecture

Android Architecture

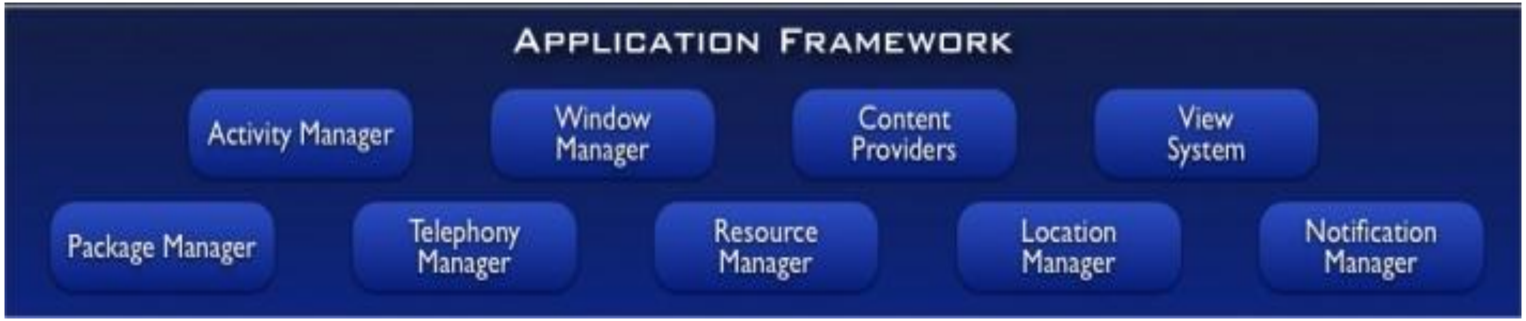


Applications



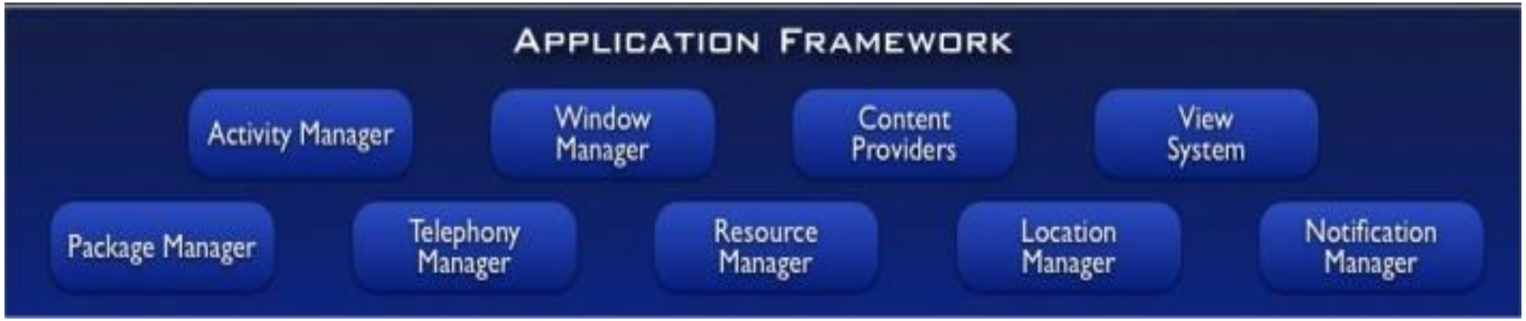
- Android ships with a set of core applications (Contacts, Calendar, Maps, Browser, etc.)
- All apps are written in Java

Application Framework



- Activity Manager
- Package Manager
- Telephony Manager
- Window Manager
- Resource Manager

Application Framework



- Content Providers
- Location Manager
- Notification Manager
- View System

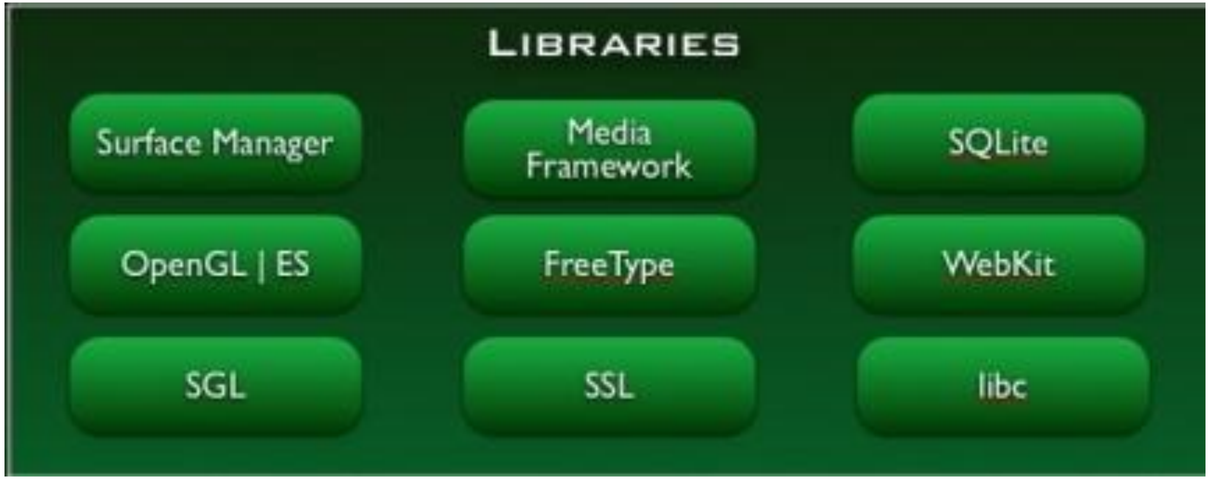
Application Security Box

Upon Android application installation:

- A unique user ID is created for each Android app
- Each app is started in its own process
- Each app runs in that DVM
- The file permissions are set for the owner only to access

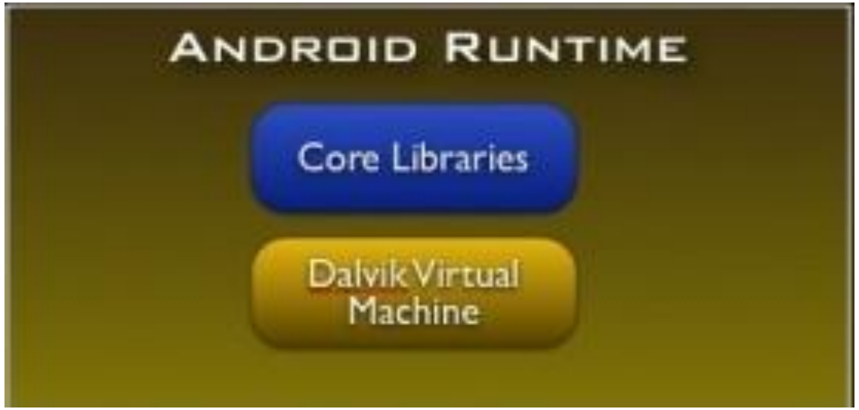
Native Libraries

- LIBC
- OpenGL
- SQLite
- Freetype
- Surface Manager
- LibWebCore
- SSL
- SGL
- Media Libraries



Android Runtime

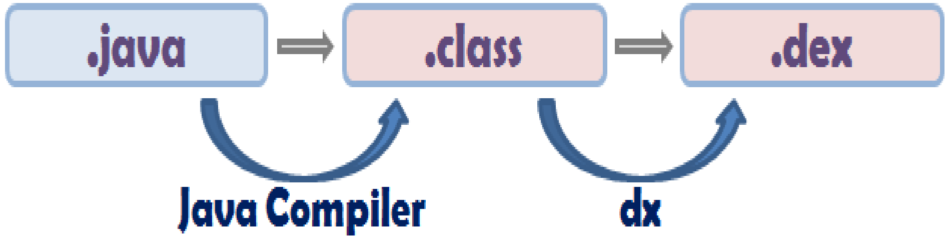
- Core libraries
- Dalvik Virtual Machine – provides environment on which each Android application runs



Android Runtime

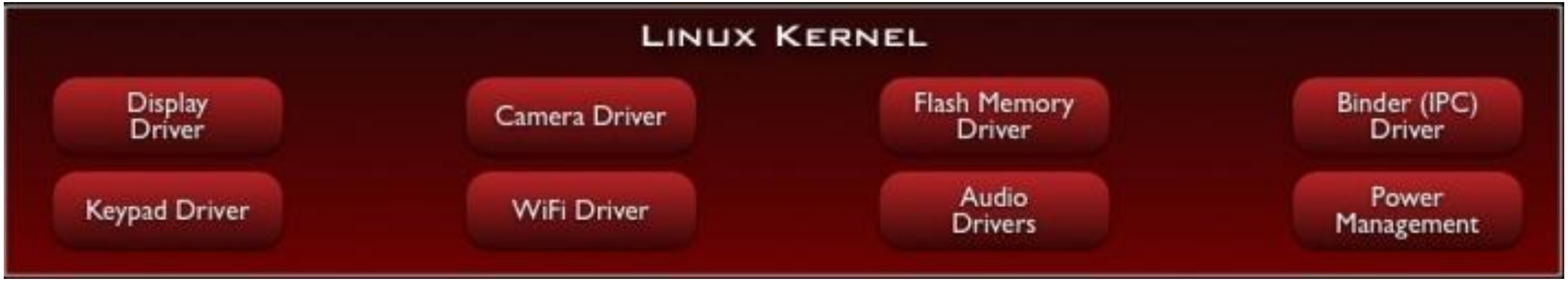
Dalvik Virtual Machine

- Dalvik Executable (.dex) format
  - Optimized for minimal memory footprint
  - Compilation



- Relies on the Linux kernel for:
  - Threading
  - Low-level memory management

Linux Kernel



- Composed of drivers
- Relies on Linux version 2.6 for core system services
  - Security
  - Memory Management
  - Process Management
  - Network Stack
  - Driver Model

What is an .APK File

A package containing

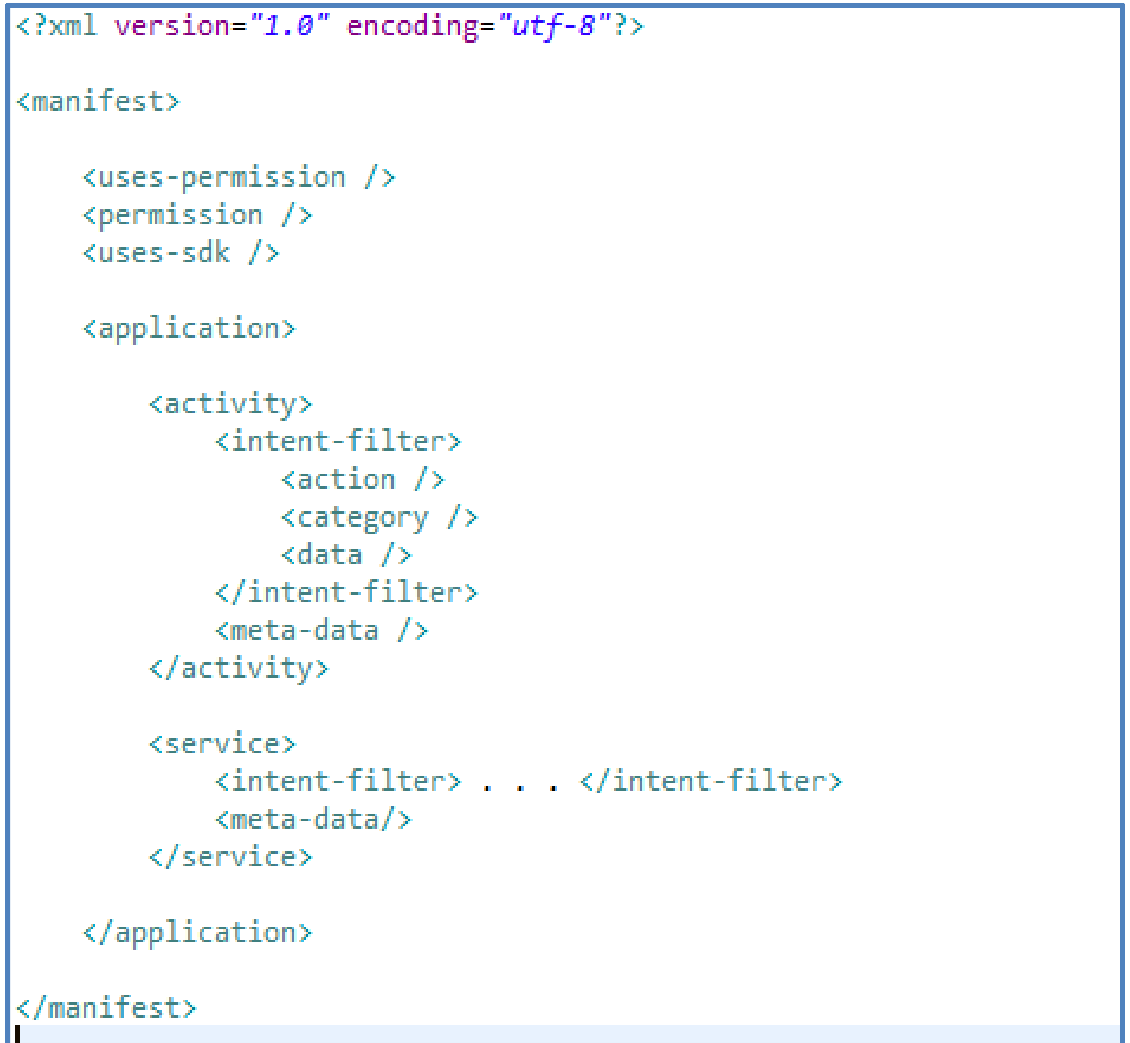
- .DEX files
- Resources
- Assets
- Certificates
- Manifest file

A .ZIP formatted package based on .JAR file format

Manifest File

- Declares the components and settings of an application
- Identifies permissions (ie. network access)
- Defines the package of the app
- Defines the version
- Declares hardware and software features

Manifest File



```xml
<?xml version="1.0" encoding="utf-8"?>

<manifest>

    <uses-permission />
    <permission />
    <uses-sdk />

    <application>

        <activity>
            <intent-filter>
                <action />
                <category />
                <data />
            </intent-filter>
            <meta-data />
        </activity>

        <service>
            <intent-filter> . . . </intent-filter>
            <meta-data/>
        </service>

    </application>

</manifest>
```

Android Applications Components

Android Application Components

- Activity
- Service
- BroadcastReceiver
- ContentProvider

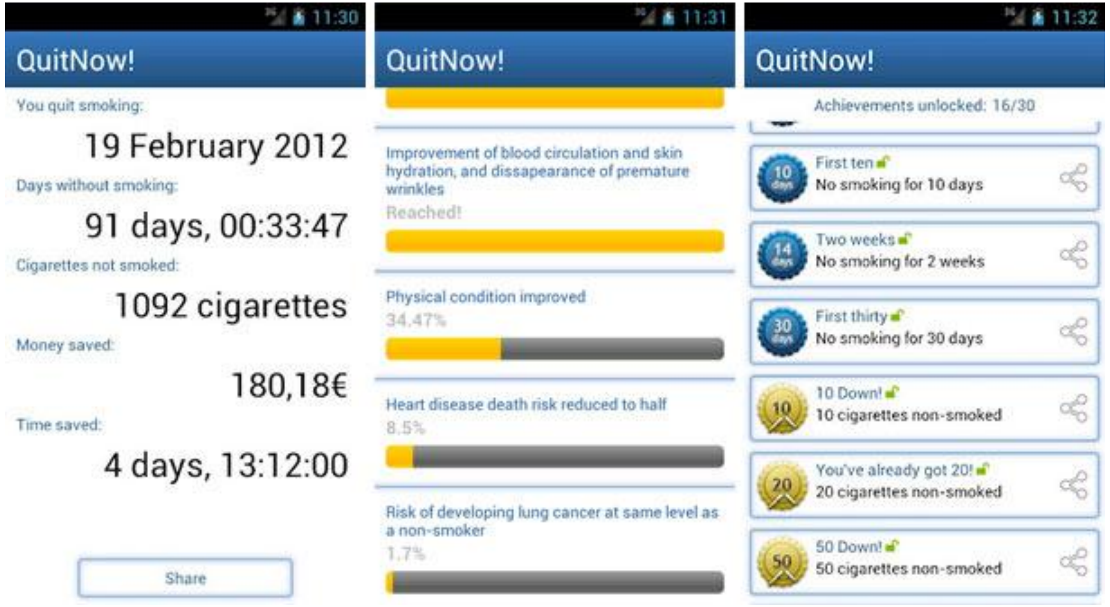
Activating Components

- Intents
- IntentService

Activity

- Activity Overview
- Activity Lifecycle
- Activity Callbacks
- Activity Lifecycle Explored
- Registering Activity
- Building UI
- Activity Sample
- Preference Activity

Activity Overview



11:30

QuitNow!

You quit smoking:

19 February 2012

Days without smoking:

91 days, 00:33:47

Cigarettes not smoked:

1092 cigarettes

Money saved:

180,18€

Time saved:

4 days, 13:12:00

Share

11:31

QuitNow!

Improvement of blood circulation and skin hydration, and dissapearance of premature wrinkles

Reached!

Physical condition improved

34,47%

Heart disease death risk reduced to half

8,5%

Risk of developing lung cancer at same level as a non-smoker

1,7%

11:32

QuitNow!

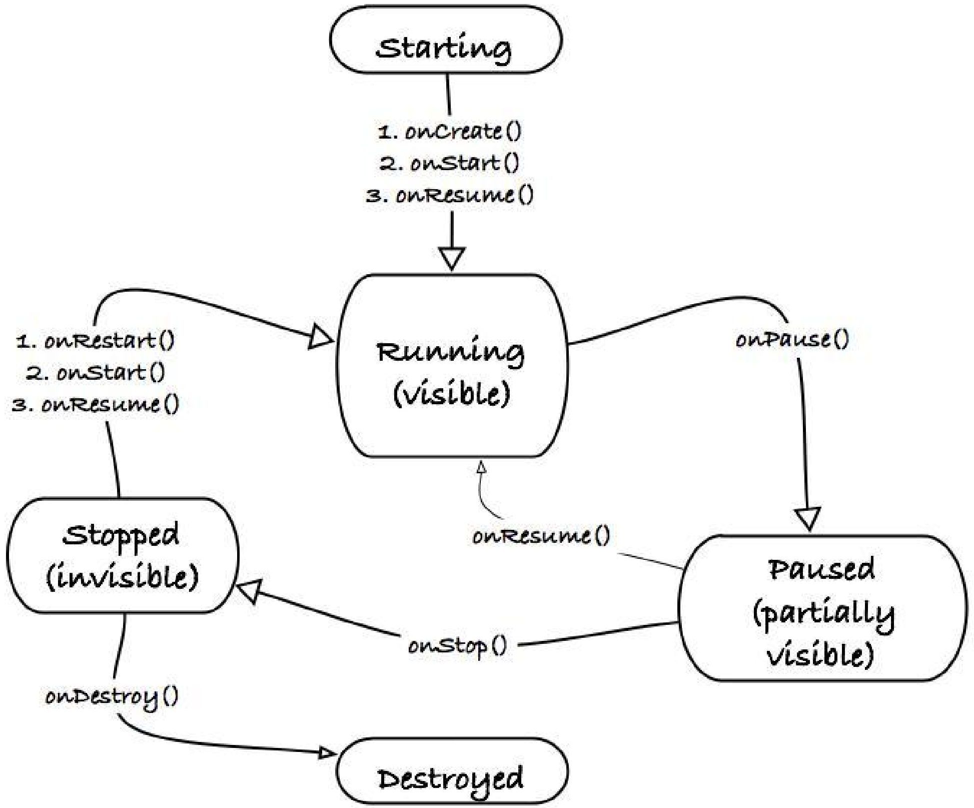
Achievements unlocked: 16/30

10 days	First ten No smoking for 10 days
14 days	Two weeks No smoking for 2 weeks
30 days	First thirty No smoking for 30 days
10	10 Down! 10 cigarettes non-smoked
20	You've already got 20! 20 cigarettes non-smoked
50	50 Down! 50 cigarettes non-smoked

- A screen with a UI
- A typical application may have many activities
- Typically expensive and so are managed by the system
- May be shutdown by the system

Activity Lifecycle

- Activities are managed by the system’s Activity Manager.
- Applications show new screens by pushing a new activity onto the screen. Navigating back (via back button) causes the top screen to be popped off the stack.
- You define what happens in each callback.



Activity Callbacks

onCreate()
Used to setup your activity. Must be implemented. Good place to inflate the UI and setup listeners.

onResume() and onPause()
Use them to turn on and off things that you'd like to have running only while the activity is visible. This is important for things that consume a lot of battery, such as GPS and sensors.

onStart() and onStop()
Use to setup code that starts/stops the activity. Unlike onResume() and onPause(), it includes Paused state as well.

Activity Callbacks

onRestart()
Called when the activity is restarted. It is followed by onStart() and onResume().

onDestroy()
A good place to do any clean-up before the activity is cleaned up from memory.
This is the counter-part to onCreate().

Activity Callbacks

onCreateOptionsMenu()
Called when the menu is loaded (typically from an XML resource).

onOptionsItemSelected()
Called whenever an option menu item is clicked on. Load the menu, typically from an XML resource.

onSaveInstanceState()
Called before placing the activity in a background state

onRestoreInstanceState()
Called after onStart() when the activity is being re-initialized from a previously saved state

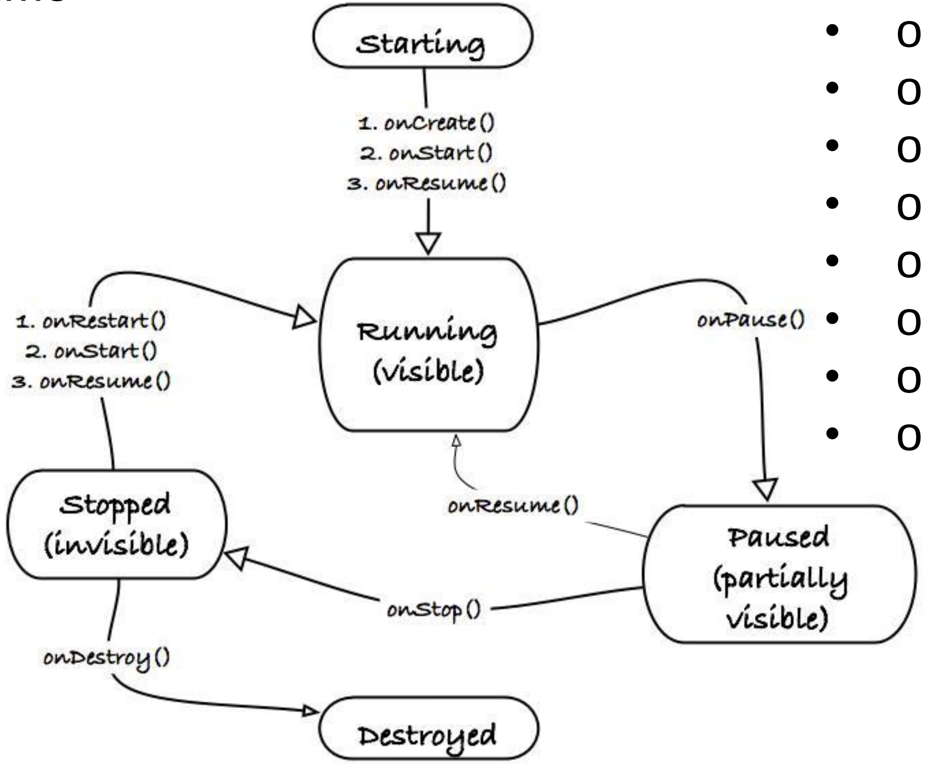
Activity Lifecycle Explored

Starting for the first time

- onCreate
- onStart
- onResume

Restarting next time

- onRestart
- onStart
- onResume



Rotating the screen

- onSaveInstanceState
- onPause
- onStop
- onDestroy
- onCreate
- onStart
- onRestoreInstanceState
- onResume

Registering the Activity

Register the activity in the manifest file:

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

The intent filter specifies that this activity is to be the main entry point into the application as well as that it should be shown in the app launcher on home screen.

Building Android UI

There are 2 approaches to building Android UI:

Declaratively

- Declare UI in XML
- Eclipse provides nice drag-n-drop tools
- Inflate the XML view in Java

Programmatically

- Write Java code for the UI
- Instantiate all widgets programmatically
- Set properties for each

Best approach is to combine both:

1. Declare the look and feel using XML
2. Inflate XML into Java.
3. Program the actions using Java.

Preference Activity

Specialized activity that knows how to display, read and write application’s user preferences.

```
import android.os.Bundle;
import android.preference.PreferenceActivity;

public class PrefsActivity extends PreferenceActivity {
	@Override
	protected void onCreate(Bundle savedInstanceState {
		super.onCreate(savedInstanceState);
				addPreferencesFromResource(R.xml.prefs);
	}
}
```

R.xml.prefs refers to Preference screen resource.

Preference Activity

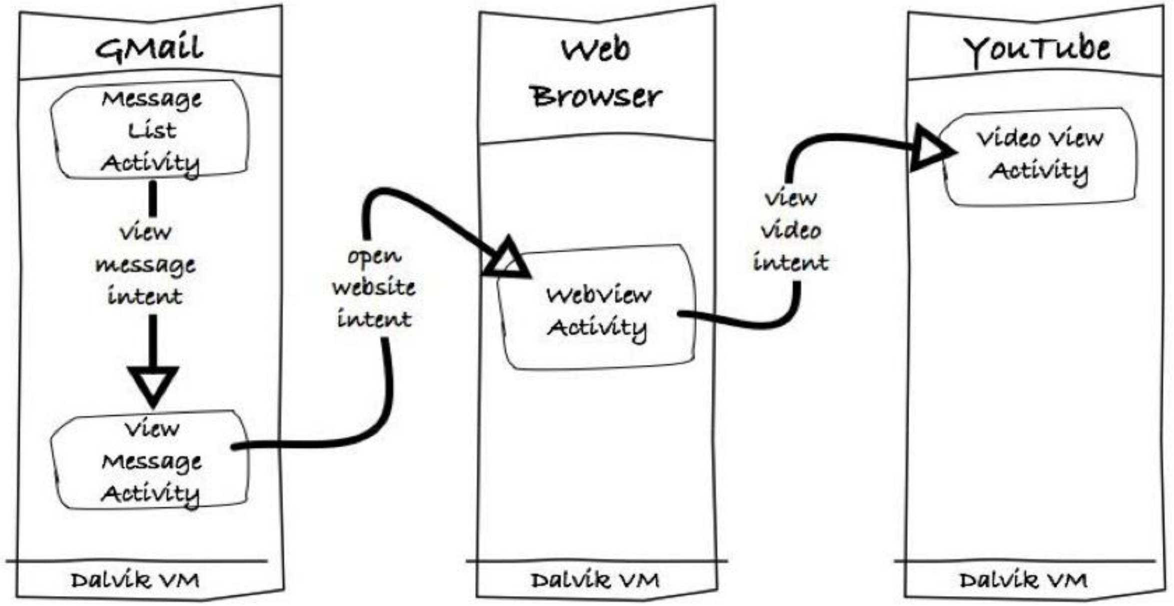
res/xml/prefs.xml

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen
xmlns:android="http://schemas.android.com/apk/res/android">
        <CheckBoxPreference
                android:title="Background data"
                android:key="background_data"
                android:summary="Use background data?”>
        </CheckBoxPreference>
        <ListPreference
                android:key="text_size"
                android:entries="@array/text_size_entries“
                        android:entryValues="@array/text_size_values"
                android:title="Text Size”>
        </ListPreference>
</PreferenceScreen>
```

Intent

- Intent Overview
- Using Intents
- Intent Filters
- Explicit and Implicit Intent Sample

Intent Overview



- Intents are like events or messages.
- Can be used to start activities, start/stop services, or send broadcasts.
- Composed of an action that needs to be performed and the data that it needs to perform the action on.
- Can be implicit or explicit.

Using Intents

startActivity()
Starts an activity specified by the intent. If activity does not exist already, it calls onCreate() to create it. Otherwise, it calls onStart() and onResume().

startService()
Starts a service. Even if the service is not created yet, it called onCreate() on the service first.

stopService()
Stops a service that is already running. If service is not running, it does nothing.

Using Intents

bindService()
Binds to a service. Requires that the service returns a binder via onBind() method.

sendBroadcast()
Sends a broadcast. If there’s a broadcast receiver registered to filter for the same action as this intent is specifying, that receiver’s onReceive() method will be called.

Using Intents Sample

An activity can send an intent to start another activity:

```
Intent I = new Intent (this, SecondActivity.class);
startActivity(i);
```

An intent can contain data to be used by the receiving component:

```
String url = http://www.google.com;
Intent i = new Intent (Intent.ACTION_VIEW);
i.setData(Uri.parse(url));
startActivity(i);
```

Intent Filters

- Intent filter is a way for us to assign certain action to an activity, service, receiver or similar.

- Action is one of system defined actions, or something you come up with.

- Intent filter typically goes into Android Manifest file, within <activity>, <service>, or <receiver> elements.

Android Manifest file

```
...
<intent-filter>
    <action android:name="some.action.goes.here" />
</intent-filter>
...
```

Intent Filter Sample

Requires that there’s an intent filter filtering for this particular intent, for example:

AndroidManifest.xml

```
…
<service android:name=".IntentServiceDemo">
      <intent-filter>
            <action android:name="MSI.intent.action.IntentServiceDemo" />
      </intent-filter>
</service>
…
<receiver android:name=".ReceiverDemo">
      <intent-filter>
            <action android:name="MSI.intent.action.ReceiverDemo" />
      </intent-filter>
</receiver>
…
```

Explicit Intent Sample

`ActivityDemo.java`

```
...
startActivity(new Intent(this, AnotherActivity.class));
...
startService(new Intent(this, ServiceDemo.class));
...
```

this is the context from which this intent is being sent, in this case an Activity or Service

Implicit Intent Sample

`ActivityDemo.java`

```
…
startService(new Intent(“MSI.intent.action.IntentServiceDemo"));
…
sendBroadcast(new Intent(“MSI.intent.action.ReceiverDemo"));
…
```

Requires an intent filter filtering for this particular intent.

Implicit Intent Sample

Sample intent filter:

`AndroidManifest.xml`

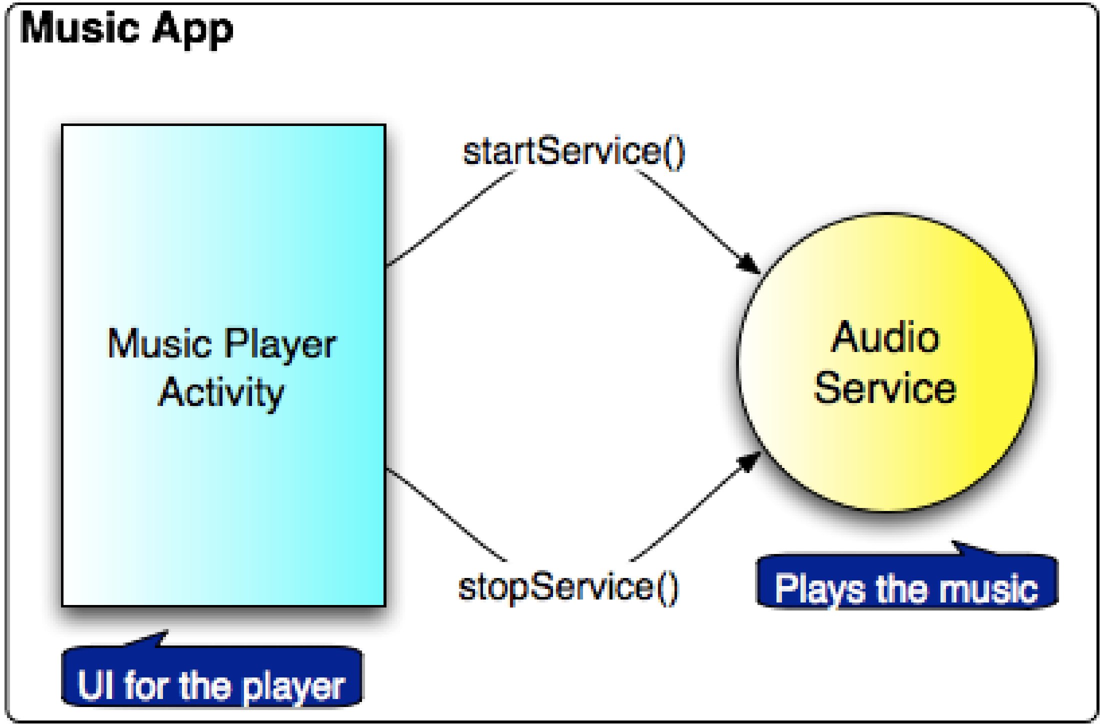
```
…
<service android:name=".IntentServiceDemo">
  <intent-filter>
     <action android:name=”MSI.intent.action.IntentServiceDemo" />
  </intent-filter>
</service>
…
<receiver android:name=".ReceiverDemo">
  <intent-filter>
      <action android:name=”MSI.intent.action.ReceiverDemo” />
  </intent-filter>
</receiver>
…
```

Service

- Service Overview
- Service Lifecycle
- Service Lifecycle Explored
- Service Sample
- Service Callbacks
- Registering Service

Service Overview

- Runs in the background
- Can be started and stopped
- No UI



Service Lifecycle

- Service starts and "runs" until it gets a request to stop.
- Service will run on the main application thread.
- To offload work from main thread, use intent service.
- Intent service uses worker thread, stops when done with work.
- Services can be bound or unbound.

Service Lifecycle Explored

Starting a service first time

- onCreate
- onStartCommand

Restarting the same service

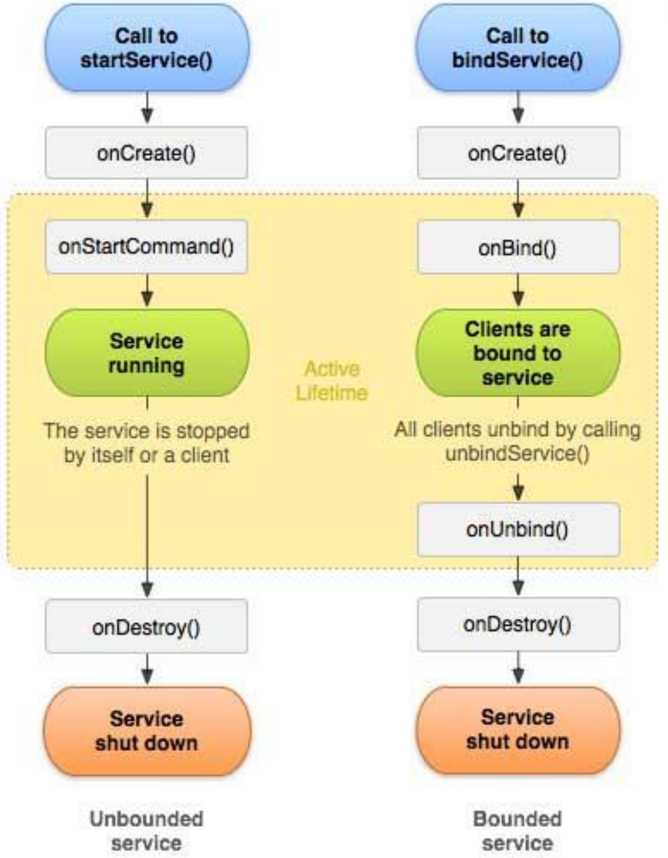
- onStartCommand

Stopping the service

- onDestroy

Starting the intent service

- onCreate
- onHandleIntent
- onDestroy



Service Callbacks

onCreate()
Called when service is first created.

onStartCommand()
Called every time the service is started.

onDestroy()
Called when service is stopped. It is subsequently destroyed.

onBind()
Required, but for unbound services, we just return null. This is called when bindService() is called.

IntentService Callbacks

Constructor
It needs to pass the name of this service to its super.

onCreate()
Called when service is first created.

onHandleIntent()
This is where the work of the service runs.

onDestroy()
Called when service is stopped. It is subsequently destroyed.

Registering a Service

Registering a service that will be called explicitly by its class name

```
<application>
        ...
        <service android:name=".ServiceDemo"></service>
        ...
</application>
```

Registering a service that will be called via action

```
...
<service android:name=".IntentServiceDemo">
        <intent-filter>
                <action android:name=“MSI.intent.action.IntentServiceDemo" />
        </intent-filter>
</service>
...
```

Broadcast Receiver

- Broadcast Receiver Overview
- Broadcast Receiver Sample
- Broadcast Receiver Callbacks
- Registering Broadcast Receiver

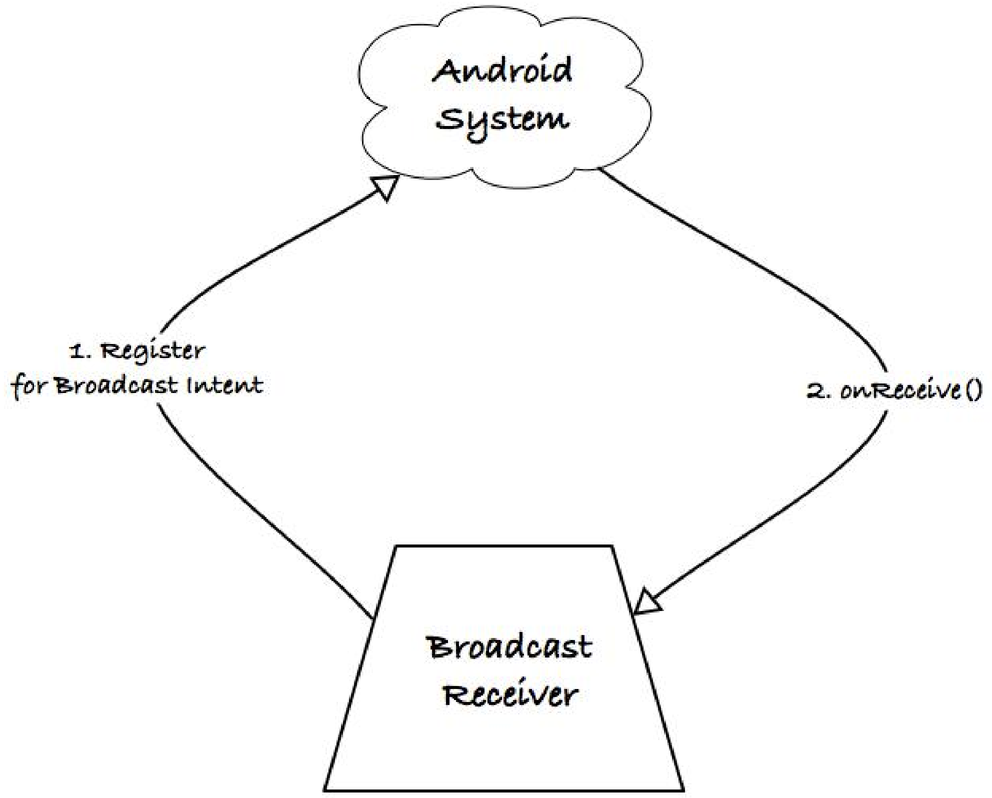
Broadcast Receiver Overview

- An Intent-based publish-subscribe mechanism
- Respond to broadcast messages from other applications or from the system.
- Great for listening for system events such as SMS messages

Broadcast Receiver Lifecycle

Sending a broadcast to a receiver

- onReceive



Broadcast Receiver Callbacks

onReceive()

This is the only method you typically care about for a broadcast receiver. It is called when this receiver is invoked.

Registering a Broadcast Receiver

Registering in Android Manifest file

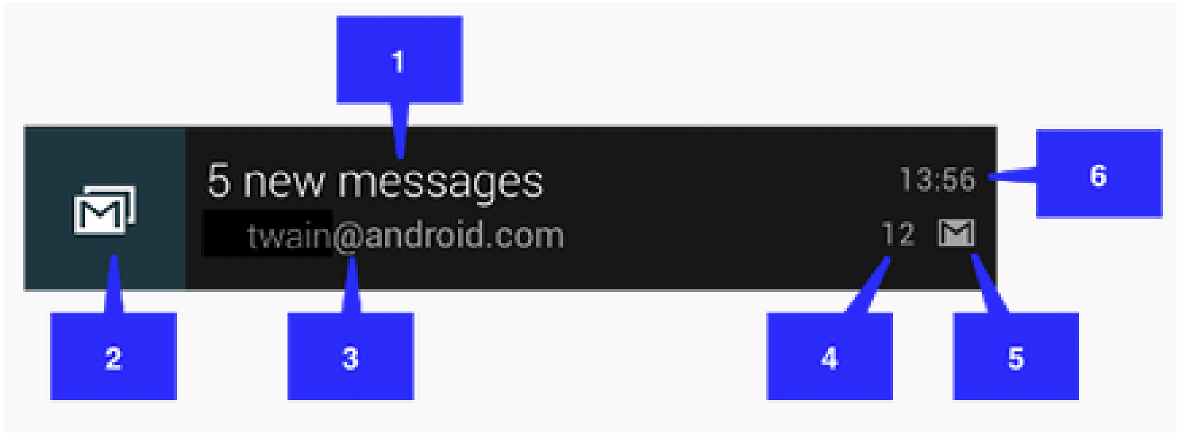
```
<application>
  <receiver android:name=".ReceiverDemo">
       <intent-filter>
              <action android:name=“android.intent.action.BOOT_COMPLETED"/>
       </intent-filter>
  </receiver>
</application>
```

Notification

- Notification Overview
- Notification Usage Sample

Notification Overview

- Alerts the user in the status bar
- Contains the following elements:
    1. Content title
    2. Large icon
    3. Content text
    4. Content info
    5. Small icon
    6. Time the notification was issued


5 new messages
[redacted]@android.com
13:56
12

Notification Overview

- Create notifications with the NotificationManager class
- Implementation includes intent to be started - typically some Activity gets started on clicking the notification
- Don’t forget to cancel the notification

Content Provider

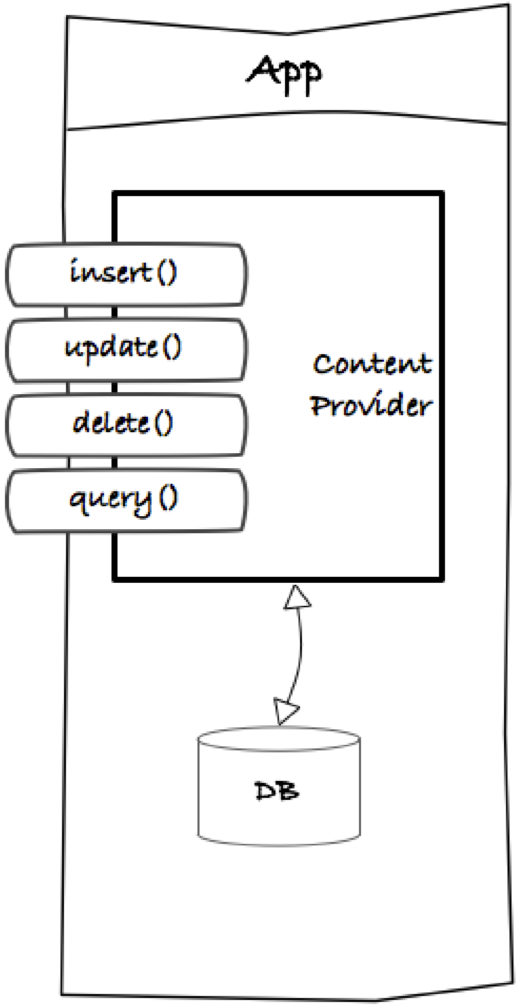
- Content Provider Overview
- Content Provider Lifecycle
- Content Provider Lifecycle Explored
- Content Provider Sample
- Content Provider Callbacks
- Registering Content Provider

Content Provider Overview

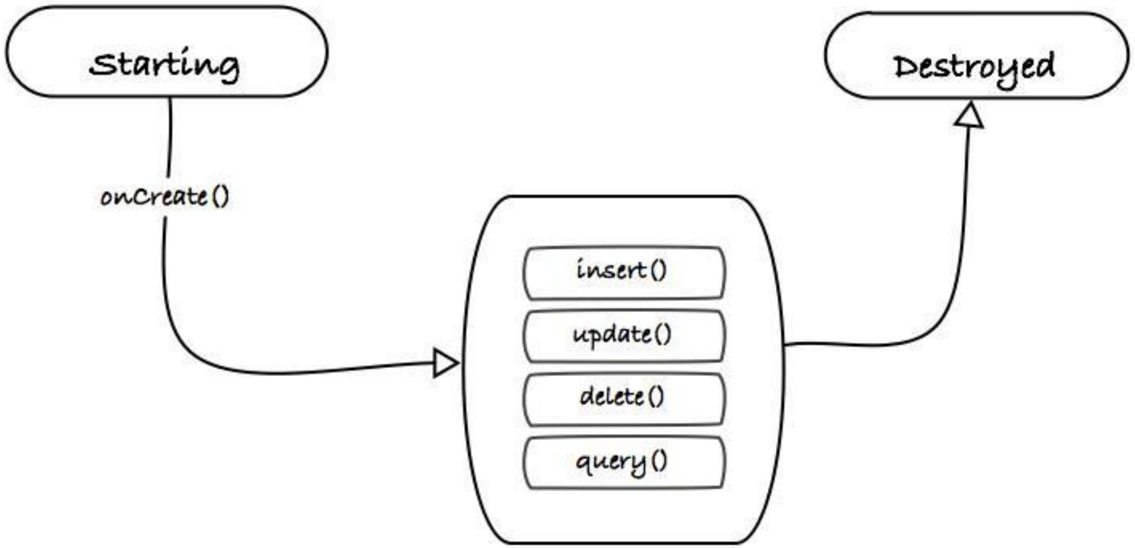
Content Providers share content with applications across application boundaries.

Examples of built-in Content Providers are:

- Contacts
- MediaStore
- Settings and more



Content Provider Lifecycle



- Content provider is initiated first time it is used via a call to onCreate().
- There is no callback for cleaning up after the provider.
- When modifying the data (insert/update/delete), open/close database atomically.
- When reading the data, leave database open or else the data will get garbage collected.

Content Provider Callbacks

onCreate()
Used to initialize this content provider. This method runs on UI thread, so should be quick.

insert()
Inserts the values into the provider returning URI that points to the newly inserted record.

update()
Updates records(s) specified by either the URI or selection/selectionArgs combo. Returns number of records affected.

Content Provider Callbacks

delete()

Deletes records(s) specified by either the URI or selection/selectionArgs combo. Returns number of records affected.

query()

Queries the provider for the record(s) specified by either URI or selection / selectionArgs / grouping / having combo.

getType()

Returns a MIME type that describes the type of data returned by the content URI argument.

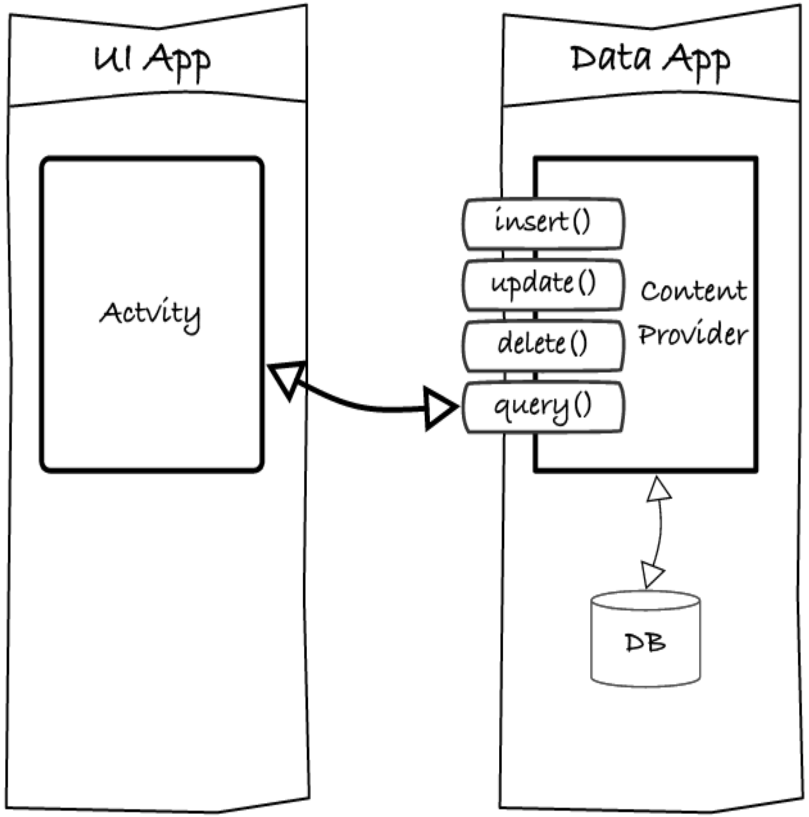
Registering a Content Provider

Registering in the manifest file:

```
<application>
...
  <provider
    android:name=".ProviderDemo“
    android:authorities="com.MSI.android.lifecycle.providerdemo"
  />
...
</application>
```

The authority of this provider must match the URI authority that this provider is responding to.

Typical Use of Content Providers



Summary / Recap

- Android applications are developed in Java. Decide on the API level that your app will be supporting.
- Each Android application runs in its own Application Security Sandbox
- Android system can kill your activity anytime if it decides its not required. Hence save your screen data constantly.
- Your application settings can be stored/retrieved using Preference Activity.
- You could register intent filters to start your activity or service from any other app.
- Services are the code that does not have UI.
- Use IntentService to perform any background activity.
- Broadcast Receiver listens to the system event to occur.
- Use a Content Provider if shared data access is required.

THANK YOU !