

Android Concepts and Programming

TUTORIAL 3

Kartik Sankaran

kar.kbc@gmail.com

CS4222 Wireless and Sensor Networks

[2nd Semester 2015-16]

22nd March 2016

Agenda

Project discussion

Android GUI topics:

1. Activity Lifecycle
2. Sending Intents
3. Loopers and Handlers
4. Services
5. Activity-service communication
6. Notifications
7. AdapterViews and Adapters
8. Internal/External storage, resources, assets

A few useful references for the project

GPS for stop locations:

An integrated stop-mode detection algorithm for real world smartphone-based travel survey

(<http://ares.lids.mit.edu/fm/documents/stopmode.pdf>)

Walking detection:

A Comparison of Pedestrian Dead-Reckoning Algorithms using a Low-Cost MEMS IMU

(uploaded with lecture files in the workbin)

Indoor/outdoor detection:

IODetector: a generic service for indoor outdoor detection

Activity Lifecycle

© 2010 Bottomless, Inc. All Rights Reserved
<http://blog.bottomlessinc.com>

Starting

1 onCreate()
2 onStart()
3 onRestoreInstanceState() *
4 onResume()

Only if app was killed or screen was rotated

Running

1 onSaveInstanceState() *
2 onPause()

save() maybe called. Before or after pause(), but surely before stop().

3 onResume()
2 onStart()
1 onRestart()

onResume()

Stopped

Paused

1 onSaveInstanceState() *
2 onStop()

Use 'shared preferences' to save GUI state

onDestroy()
or
Process killed

Process killed

Destroyed

* Optional

Android Activity Life Cycle

The diagrams in the Android website are not very clear

Activity Lifecycle

TYPICAL TRANSITIONS:

1. **ON LAUNCHING APP:** create, start, *maybe restore*, resume
2. **BACK BUTTON PRESSED:** pause, stop, destroy
3. **HOME BUTTON PRESSED:** *save*, pause, stop
4. **AFTER HOME PRESSED, LAUNCHING APP AGAIN:** restart, start, resume
5. **TURN OFF SCREEN:** *save*, pause
6. **TURN ON SCREEN (BEFORE UNLOCKING):** resume
7. **ROTATE SCREEN:**

save, pause, stop, destroy, create, start, *restore*, resume

Note: *save()* maybe called. If it is called, it is before or after *onPause()*, but surely before *onStop()*.

Intents

Intents are used to launch other activities/services,
to perform an action on some data

Examples:

1. Open a browser to display a webpage
2. Call a contact's phone number
3. Send an email/SMS
4. Pick a photo from the Gallery
5. Take a picture using the camera

and lots more...

Action: What needs to be done? (View/Edit/Pick something)

Data (URI): On what does it need to be done?

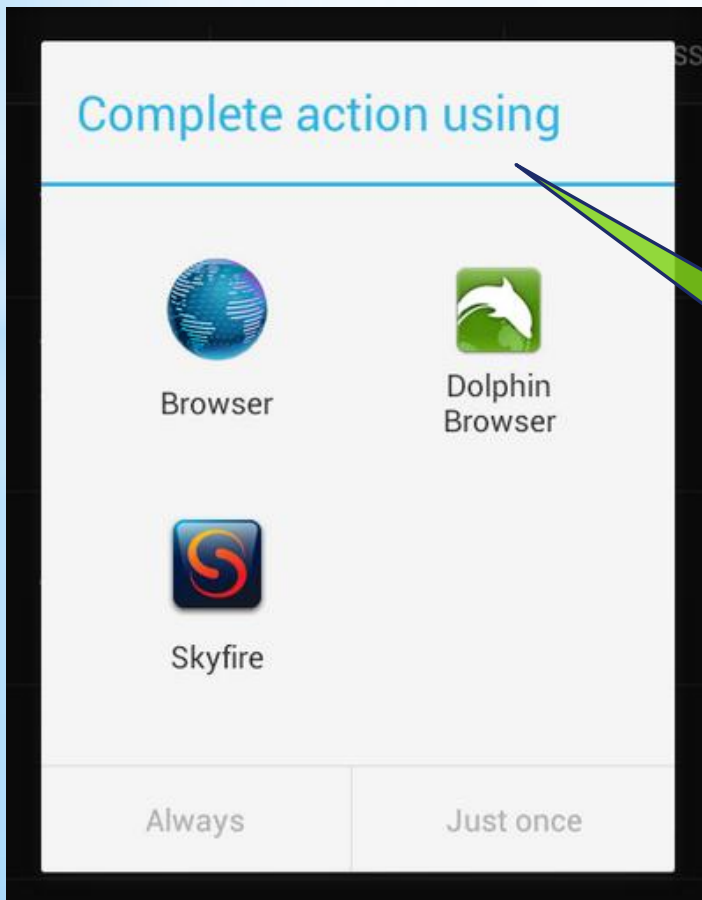
Intents

Code Example (launching a browser to view a webpage):

```
Uri uri = Uri.parse( "http://www.google.com" );  
Intent it = new Intent( Intent.ACTION_VIEW , uri );  
startActivity( it );
```

Action

Data



Intents are usually *generic (implicit)*.
Any app is allowed to handle it.

Same intent can be
handled by multiple
applications

Intents

Intent is a *powerful* feature of Android.

But, was *not well documented*.

What to use as Action and Data??

<http://developer.android.com/guide/components/intents-common.html>

<http://www.openintents.org/en/intentstable>

<http://stackoverflow.com/questions/1705728/where-is-a-list-of-available-intents-in-android>

Loopers and Handlers

What is '*handler to the main thread*'? Why is it needed?

- Only the main UI thread can update the GUI (*Why?*)
- Programmer has to figure out which thread is being used

Looper is a thread that processes tasks one after the other.

- GUI events
- Downloads/Uploads/Updates (*Why not in parallel?*)

Handler is used to post tasks (threads/messages) to the looper thread.

Loopers and Handlers

Creating your own Looper thread:

```
public void run() {  
    try {  
        Looper.prepare();  
        handler = new Handler();  
        Looper.loop();  
    }  
    catch (Throwable t) {  
        ...  
    }  
}
```

Android OS already
does this for the Main
UI thread

The handler is tied to
the thread in which it is
created

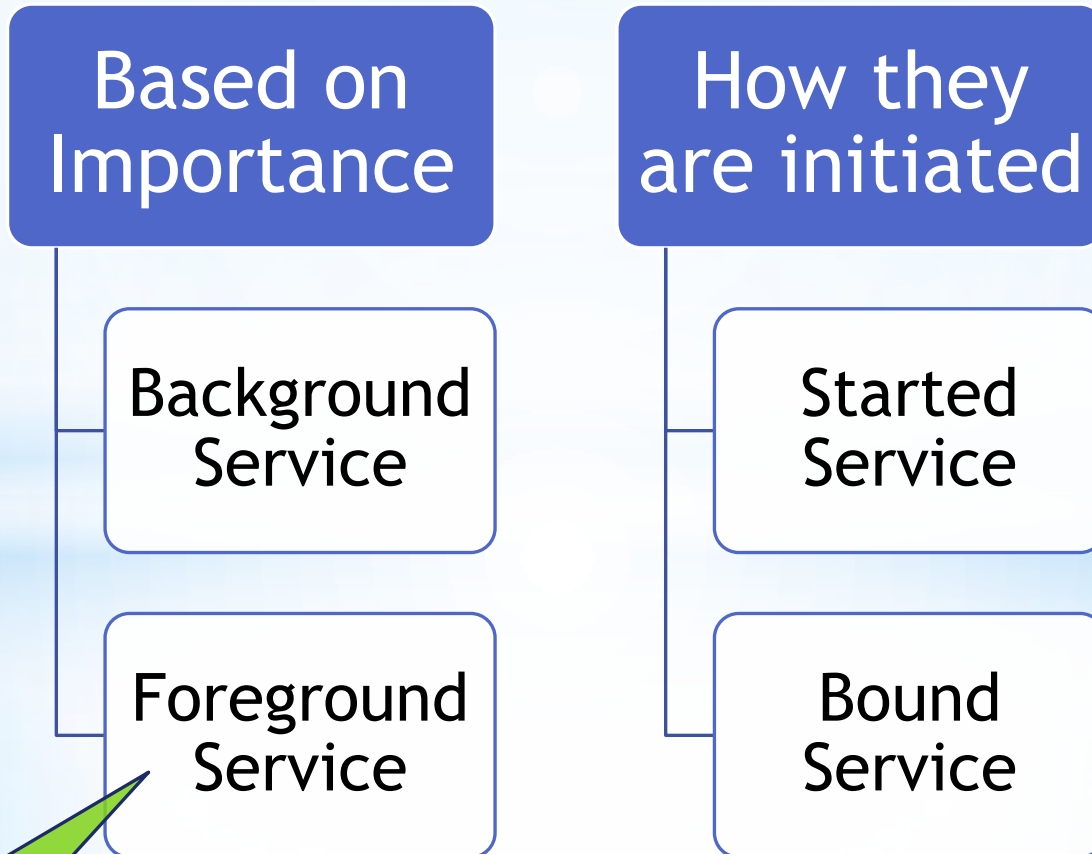
Posting a Task:

```
// From another thread  
handler.post( new Runnable() {  
    public void run() {  
        // Done in Looper thread  
    }  
} );
```

This is how you
post a task to
update the GUI

Services

Services are background tasks with little or no user interaction



Eg: Playing Music

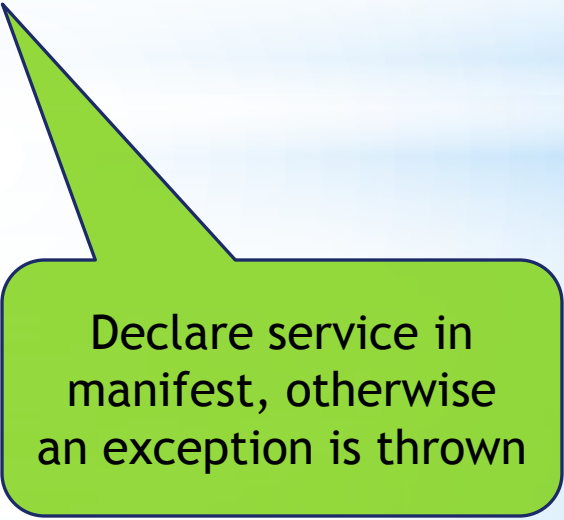
Services

Typical uses:

1. Perform a job and stop (Eg: *Download a file*)
2. Keep running periodically in the background (Eg: *Checking emails*)

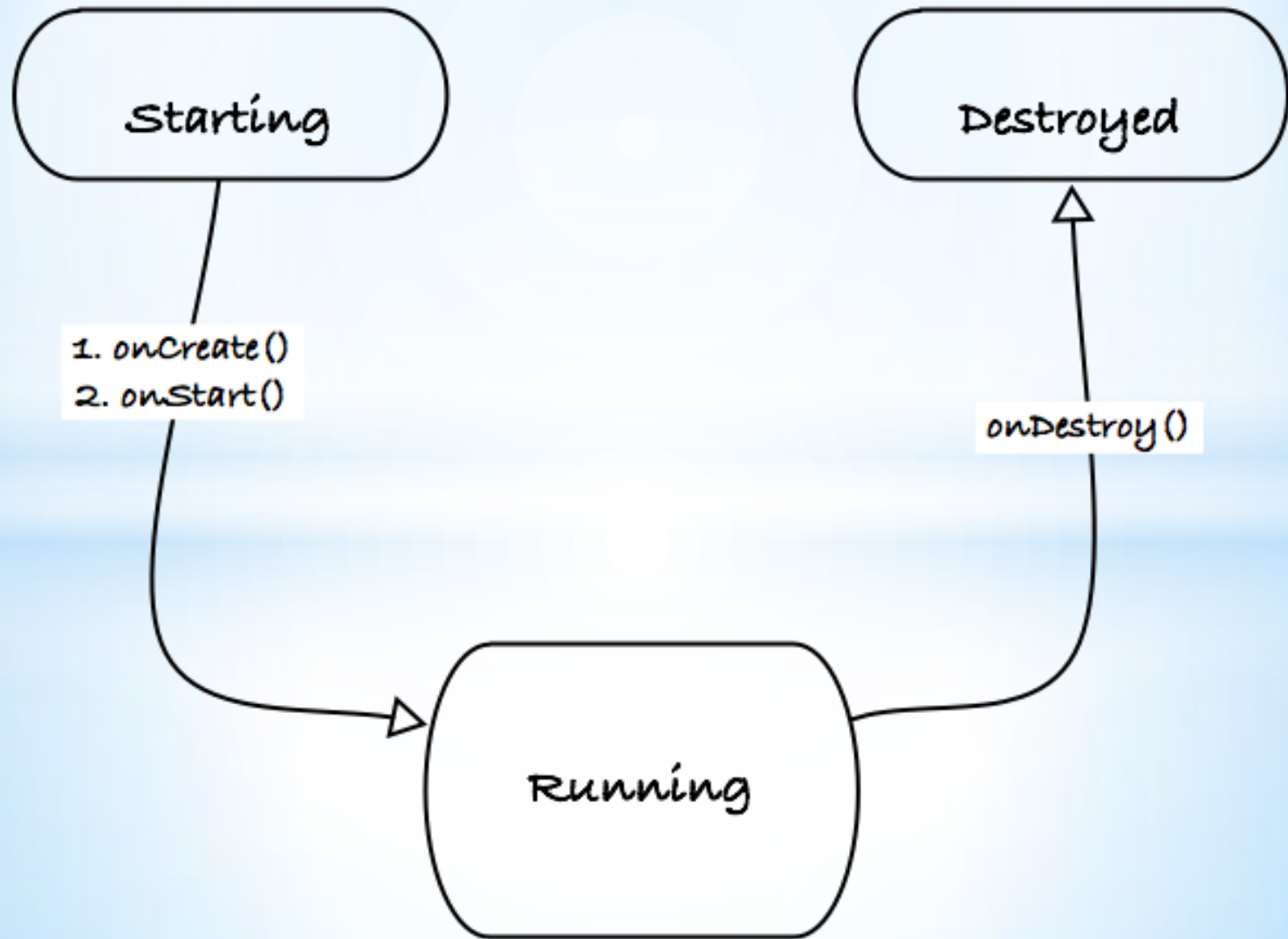
```
<application ... >  
    <service android:name=".MyService" />  
</application>
```

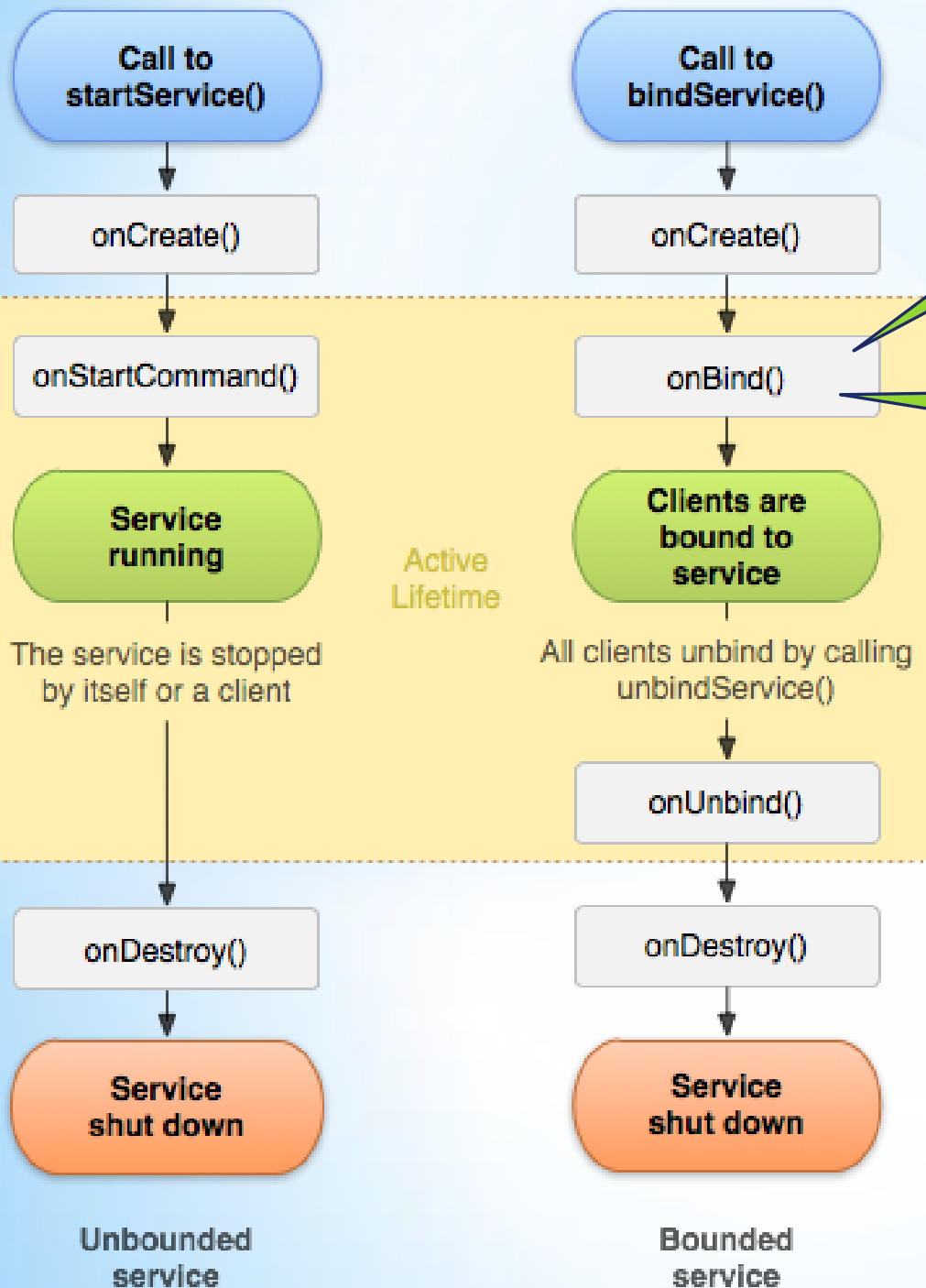
```
public class MyService  
    extends Service {  
    ...  
}
```



Declare service in
manifest, otherwise
an exception is thrown

Services





Activity's `ServiceConnection's onServiceConnected() is called after onBind()`

Android Bug: If `onBind()` fails, Activity is not notified!

Service can be *both* Started + Bound!

Eg: Music Player Service

Bound service:
`onServiceDisconnected()` called only when service crashes

By default, Services run in Main UI thread. So, *create your own thread!*

Activity-Service Communication

Communication (*most common confusion*):

Activity ==> Service (Eg: Send play/stop command to music service)

Use the Java Object returned by *onBind()*

Service ==> Activity (Eg: Service needs to update the GUI)

Can be done using binder object above, or use a messenger as below:

<http://stackoverflow.com/questions/14443247/send-message-not-reaching-the-handler-in-activity>

Discussion: Music Service updating the Music Player's GUI

DON'Ts (common mistakes):

1. Passing the GUI objects to directly the Service
(memory leaks)
2. Using *startService()* when actually you need a bound service

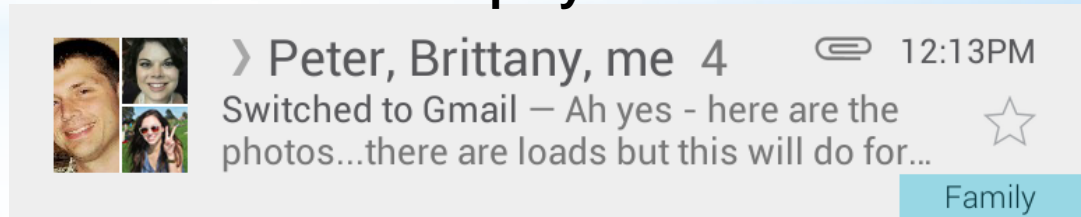
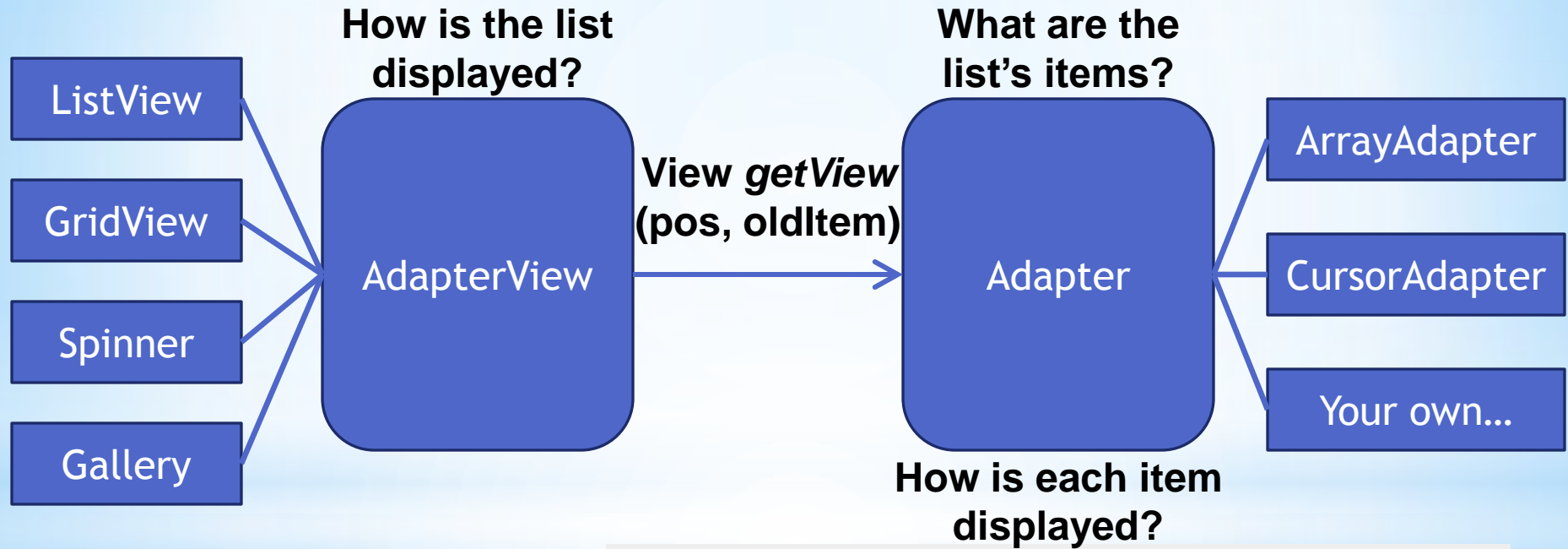
Notifications

Notifications (*second powerful feature of Android*):

- Foreground Services are *always* displayed in Notification bar
- Started Services typically post notifications (Services and Notifications are closely related)
- Pressing a notification launches the App's activity



AdapterViews and Adapters



Scrolling optimizations:

1. Modifying old out-of-screen items (*oldItem*) instead of XML inflation
2. Store reference to item's widgets in 'ViewHolder Tag' instead of using *oldItem.findViewById(R.id.email_title)*

Internal/External storage, Res, Assets, Res/raw/

User's point of view:

16 GB built-in flash (1 GB private, 15 GB public) +
32 GB removable memory card (public as well)

Developer's point of view:

Normal Java File API for public (shared/external) storage.

Special Android API for private app data storage.

Resources are understood by Android, adaptable to different screen sizes. Files in '*res/raw/*' get Resource IDs based on file name.

Assets are resources not understood by Android (Eg: Game data). No IDs. Accessed using special Android API, can be used with '*file:///*' URI.

Res & assets are stored in apk, read-only, NOT part of the filesystem.

<http://stackoverflow.com/questions/5771366/reading-a-simple-text-file>

Other Topics

**I've not covered Loaders/Fragments/ActionBar/Database
(Mark Murphy's book explains DB and AdapterViews well)**

(Old) Tools for creating Android GUIs:

<https://code.google.com/p/evoluspencil/>

<http://www.droiddraw.org/>



Questions?



Thank You!