CS 475/575

Broadcast Receivers
And Services

Broadcast Receivers

- One of the four primary application components:
  - activities
  - content providers
  - broadcast receivers
  - services

Broadcast Receivers

- Component that responds to system-wide announcements
- Android system sends multiple kinds of broadcasts
  - screen turned off, battery low, picture captured, SMS received, SMS sent
- Can also send custom user defined broadcast
- Really just intents that mean “something has happened”

SideBar – Intents Again

- Requesting please do something for me, (explicit or implicit)

```
// create intent to take picture with camera
Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
startActivityForResult(intent, TAKE_PICTURE);
```

- Announcing something has happened
  - Custom

```
//explicit intent
Intent broadcastIntent = new Intent();
broadcastIntent.setAction(ResponseReceiver.ACTION_RESP);
broadcastIntent.addCategory(Intent.CATEGORY_DEFAULT);
broadcastIntent.putExtra(ResponseReceiver.MSG, "Just a dynamic message");

sendBroadcast(broadcastIntent);
```

  - System

```
android.bluetooth.a2dp.profile.action.CONNECTION_STATE_CHANGE
android.bluetooth.a2dp.profile.action.PLAYING_STATE_CHANGED
android.bluetooth.adapter.action.CONNECTION_STATE_CHANGED
android.bluetooth.adapter.action.DISCOVERY_FINISHED
android.bluetooth.adapter.action.DISCOVERY_STARTED
```

You cannot send a system broadcast but you can register to receive them

Broadcast Receivers

- Applications tell other applications what's happening with actions, intents and sendBroadcast()
- Receivers register to get these notifications
- Receivers should not display UI
  - may create status bar notifications
  - or start servers
- Just a gateway to other components, does very minimal work (10 seconds to ANR)

Broadcast Receivers - Java

- Classes that extend BroadcastReceiver that listen for a particular message

```
public class ManifestDeclaredReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        //respond to the message
    }
}
```

- Created 2 different ways
  - Manifest (Static)
  - Java (Dynamic)

Broadcast Receiver Manifest Registered

```
<receiver android:name="MyReceiver" >
    <intent-filter>
        <action android:name="com.tetonsoftware.action.MYACTION" />
    </intent-filter>
</receiver>
```

- When program installed, OS reads manifest, sees there is a broadcast receiver, notes it and the action (intent) it is looking for
- If intent with this action sent, broadcast receiver invoked by OS. Even if original application closed!
- How to stop it? Can do it programmatically or uninstall

Broadcast Receiver - Dynamically Registered

Only works when app open

Define a broadcast receiver anywhere in Activity/Fragment like this:

```
mReceiver = new BroadcastReceiver() {
 @Override
 public void onReceive(Context context, Intent intent) {
     Log.d(TAG," onRecieve"); //do something with intent
   }
 }
```

Define IntentFilter in onCreate()

```
mIntentFilter=new IntentFilter("action_name");
```

Now register the BroadcastReciever in onResume() and Unregister it in onPause

```
@Override
protected void onResume() {
     super.onResume();
     registerReceiver(mReciever, mIntentFilter);
}

@Override
protected void onPause() {
    super.onPause();
    unregisterReceiver(mReciever);
 }
```

See http://stackoverflow.com/questions/4805269/programmatically-register-a-broadcast-receiver

Sending a Custom Broadcast
(you cannot send a system broadcast)

- Create an intent, set the broadcast action, set additional info and then send it.

```
//explicit intent
Intent broadcastIntent = new Intent();
broadcastIntent.setAction(ResponseReceiver.ACTION_RESP);
broadcastIntent.addCategory(Intent.CATEGORY_DEFAULT);
broadcastIntent.putExtra(ResponseReceiver.MSG, "Just a dynamic message");

sendBroadcast(broadcastIntent);
```

- Broadcast with no additional data

```
Intent intent = new Intent();
intent.setAction(ManifestDeclaredReceiver.ACTION_STRING);
sendBroadcast(intent);
```

BroadcastReceivers (Android Broadcast)

- What broadcasts are available?
- Check the Intent class
- http://developer.android.com/reference/android/content/Intent.html
  - search for "Broadcast Action"
- Also look in android-sdk\platforms\<number>\data\broadcast_actions.txt
- But be aware that you cannot manifest register for many system intents if you target API 26 and Above!

  (see https://developer.android.com/guide/components/broadcasts)

Broadcasts

String	ACTION_CAMERA_BUTTON	Broadcast Action: The "Camera Button" was pressed.
String	ACTION_CHOOSER	Activity Action: Display an activity chooser, allowing the user to pick what they want to before proceeding.
String	ACTION_CLOSE_SYSTEM_DIALOGS	Broadcast Action: This is broadcast when a user action should request a temporary system dialog to dismiss.
String	ACTION_CONFIGURATION_CHANGED	Broadcast Action: The current device `Configuration` (orientation, locale, etc) has changed.
String	ACTION_CREATE_SHORTCUT	Activity Action: Creates a shortcut.
String	ACTION_DATE_CHANGED	Broadcast Action: The date has changed.
String	ACTION_DEFAULT	A synonym for `ACTION_VIEW`, the "standard" action that is performed on a piece of data.
String	ACTION_DELETE	Activity Action: Delete the given data from its container.
String	ACTION_DEVICE_STORAGE_LOW	Broadcast Action: A sticky broadcast that indicates low memory condition on the device This is a protected intent that can only be sent by the system.

Further

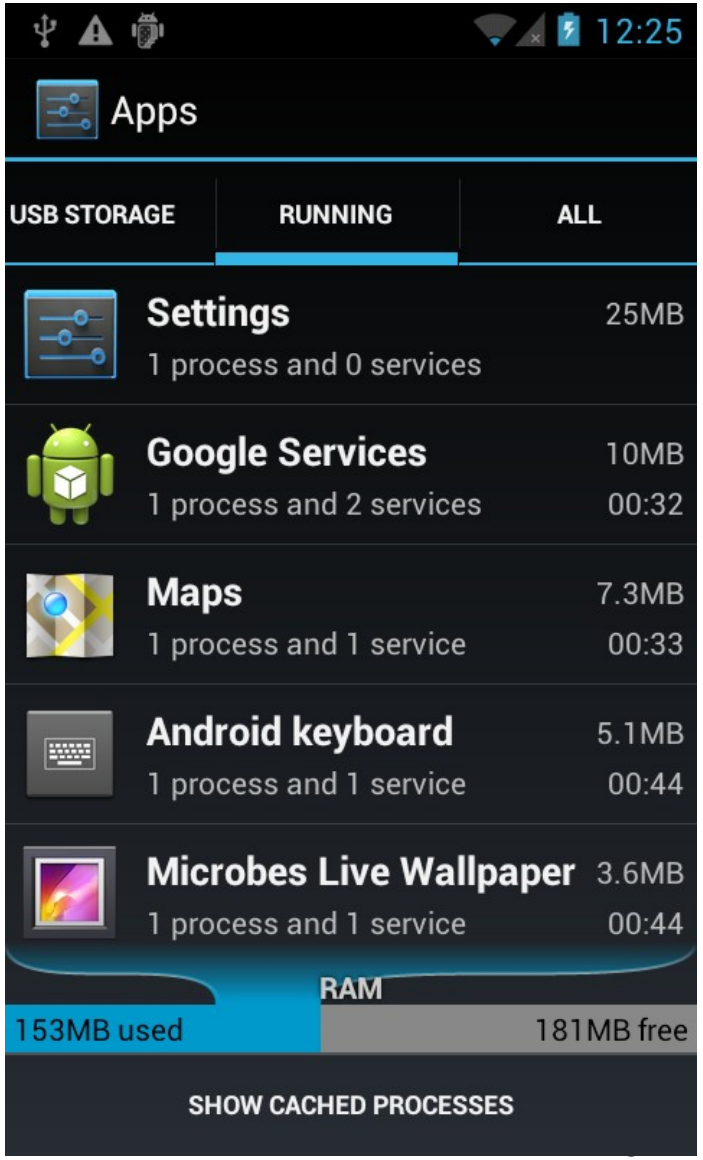
- http://developer.android.com/guide/components/services.html
- http://developer.android.com/reference/android/content/BroadcastReceiver.html
- http://www.vogella.com/tutorials/AndroidBroadcastReceiver/article.html
- http://www.vogella.com/tutorials/AndroidServices/article.html

Services

- One of the four primary application components:
  - activities
  - content providers
  - broadcast receivers
  - services

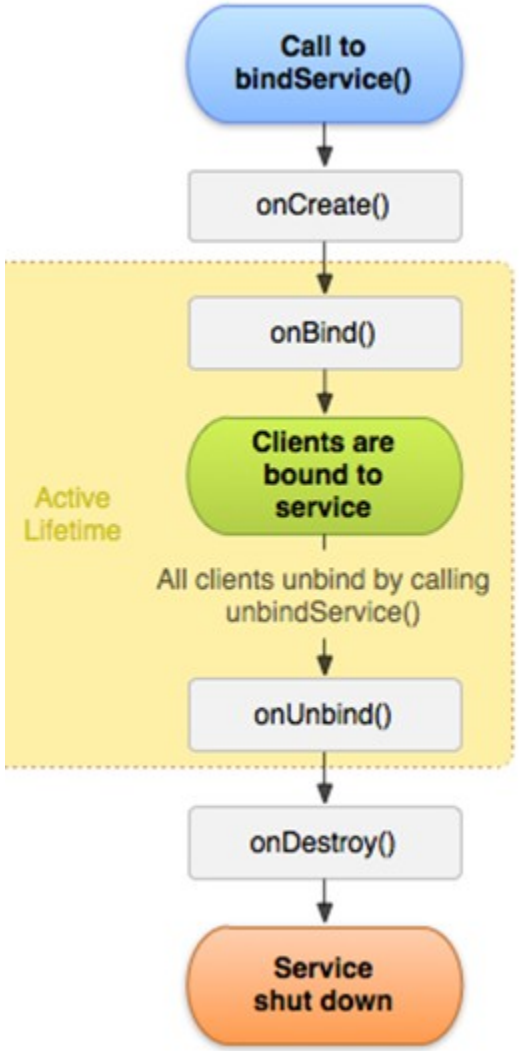
Services

- Application component that performs operations in background with no UI
- application starts service and service continues to run even if original Activity ended or user moves to another app
- 2 kinds; bound and unbound



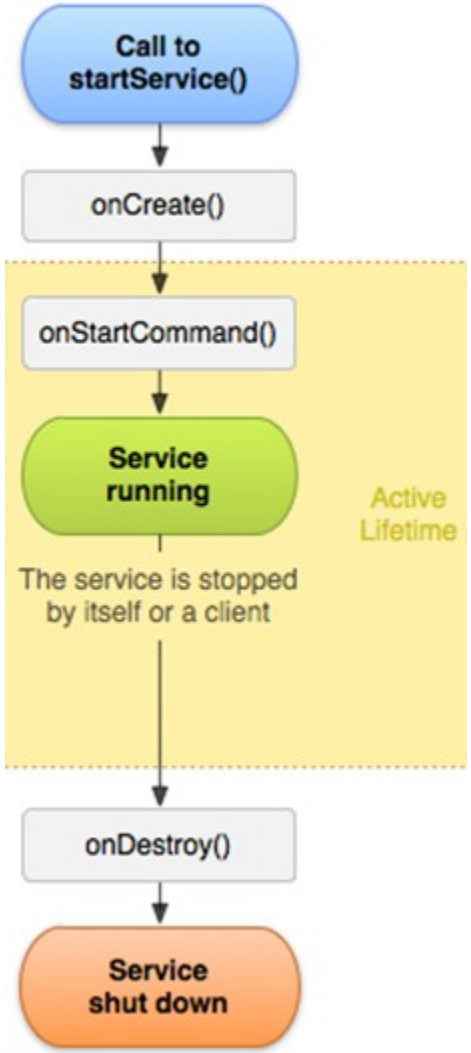
Service Types- Bound

- Bound
  - Client calls bindService()
  - Runs as long as one or more applications bound to it
  - Destroyed when calling application(s) unbinds
  - Calling application can interact with the service
  - Use case – when you need 2 way communications



Service Types-Unbound

- Unbound
  - Client calls startService()
  - Runs in main thread of hosting process
  - Override onStartCommand(..) for your work
  - Limited communication (1 way, 1 time)
    - Calling app -> Service via intent.putExtra()
  - Service closes itself when finished stopSelf() or stopService()
  - Use case example
    - Play music while browsing
    - Download a file



Service - Concurrency

- Service runs in the calling processes main thread
- Just like Activities, don’t bog it down
  - Launch a thread to do work
  - Or derive service from IntentService and override onHandleIntent(..) to do work in a separate thread.

Unbound Service - Java

- Extend the Service class and override onStartCommand()

```
public class MyService extends Service {

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        //do work here, pop UI, start thread etc
        stopSelf();
        return START_NOT_STICKY;
    }
```

  - Return what system should do if system kills service
    - START_NOT_STICKY don't restart
    - START_STICKY recreate, but don’t redeliver intent
    - START_REDELIVER_INTENT recreate and redeliver last intent

Register a Service

- Register in manifest
  - Include the service tag within the application node

```
<service
    android:enabled = "true"
    android:name="MyService"
    android:permission = "com.paad.MY_SERVICE_PERMISSION">
</service>
```

  - The permission tag ensures that any third party apps must have a uses-permission in their manifest in order to use the service.

Starting Service

- By intent from another component
- Will call onCreate and then onStartService

```
//intent to start service, then start it
Intent myIntent = new Intent(context, MyService.class);
context.startService(myIntent);
```

Service Lifecycle

- If component starts service with startService method (leads to call to onStartCommand) service runs until it calls stopSelf or another activity calls stopService
- if component calls bindService (onStartCommand not called) service runs as long as at least one component bound to it

Creating a Service (Started)

- create subclass of Android Service class or one of its existing subclasses
- override callback methods that handle important aspects of service lifecycle
- most important of these are:
  - onStartCommand
  - startService
  - onBind
  - onCreate
  - onDestroy
  - stopSelf
  - stopService