G53MDP
Mobile Device Programming

Services

Threads and Services

- How long do things take?
  - Expected that Activities regularly transition to the ”background”
    - i.e. stopped
- Threads
  - Interacting with the UI thread
  - Doing work associated with an activity
- Services
  - Application component #2
  - The Service lifecycle

Services

- An Application **Component** that
  - Has no UI
  - Represents a desire to perform a longer-running operation
    - I.e. longer than a single-activity element of the task
    - Threads are associated with the activity that started them
      - i.e. could be orphaned
- Activities are loaded/unloaded as users move around app
  - Services remain for as long as they are needed
- Expose functionality for other apps
  - One service may be used by many applications
  - Avoid duplication of resources

What Services are not

- It’s helpful to think about what a Service is not:
  - Not a separate process
    - Runs in the same process as the application in which it is declared (by default)
  - Not a thread
    - One thread per Application
      - Handles events for all components
    - If you need to do things in the background, start your own thread of execution
      - An IntentService does this automatically
- Services are logically quite simple
  - A way of telling the system about part of your app that is expected to run for a long time
    - i.e. longer than a few seconds
    - managed accordingly
  - But slightly more complicated to implement
    - IPC, *RPC*

Uses of Services

- MP3 Playback
  - Want to play audio while the user is doing other things
- Network Access
  - Long download
  - Sending an email
  - Polling an email server for new mail
- Anything that you don’t want to interrupt the user experience for
  - The user interacts with one application are once on a phone
  - Expected to outlive a single activity lifetime

Uses of Services

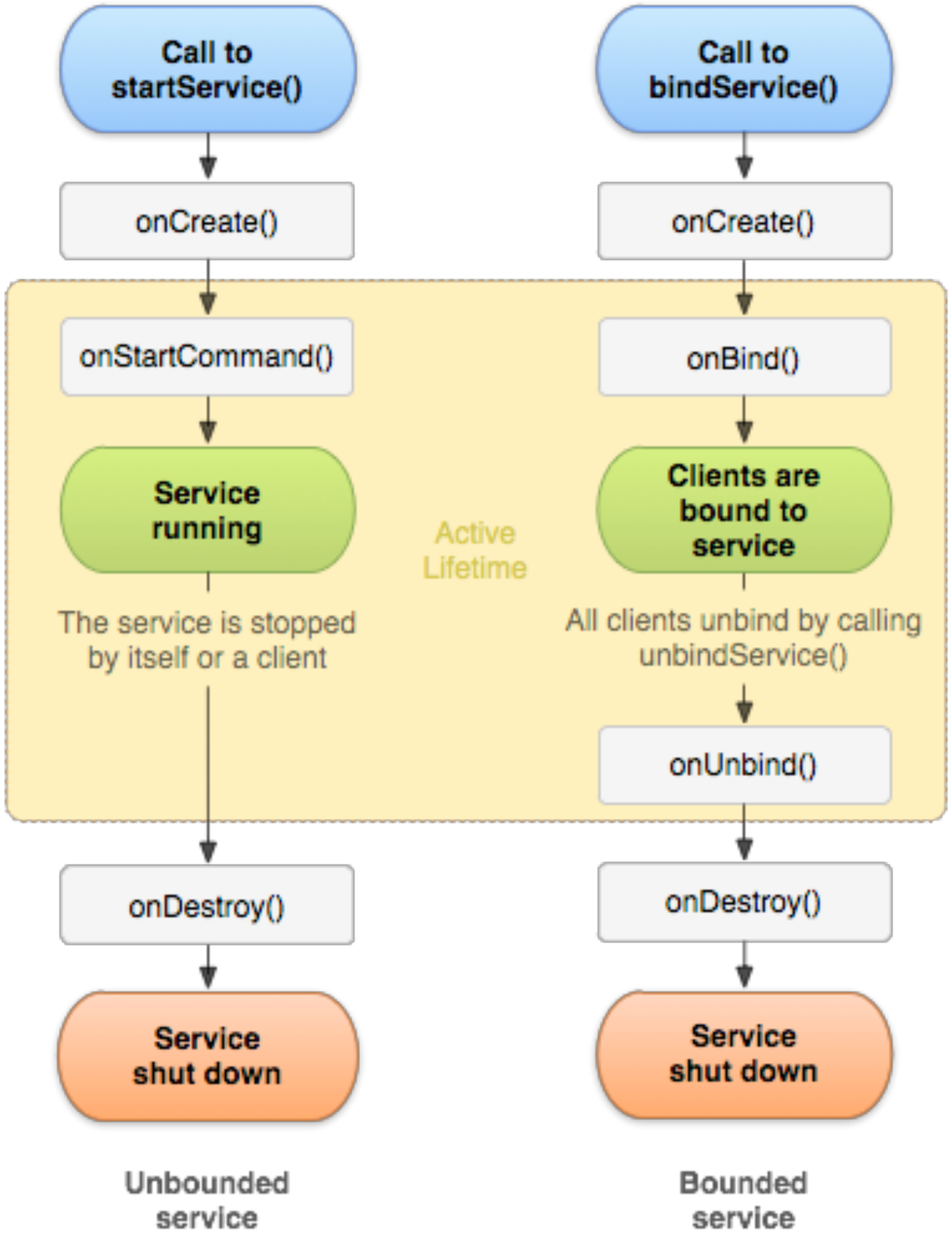
- The email task
  - Checks for new mail occasionally
  - Collects new mail and stores it somewhere
  - Notifies user that there is new mail
  - User switches to the Inbox Activity
  - Inbox Activity then fetches new mails and displays them
- MP3 playback task
  - Play music while the user does something else
  - Have activities that let you change the playing track or the volume

Creating a Service

- Services are designed to support communication with
  - (i.e. doing computation *for*)
  - Local Activities (in the same process)
  - Remote Activities (in a different process)
    - IPC
  - Multiple components
    - *System* services underpin much of Android core OS, but wrapped with various APIs
- Services are components, similar to an Activity
  - Register the service in the manifest
  - Create a subclass of android.app.Service
  - Handle the relevant lifecycle methods
    - Increasingly constrained as to what ”long running” means

Service Lifecycle

- Two ways of spawning a service
  - Started (loosely coupled)
    - Send an **Intent** to explicitly start the service with startService()
      - c.f. Messages, starting Activities
    - Will run / exist in the background indefinitely / kills itself
      - C.f email checking
      - Does not “return” results (?)
    - Explicitly stop the service with stopService()
  - Bound (tightly coupled)
    - Bind to a service using bindService()
    - Will run while any Activities are bound to it
      - Actively using it
    - Provides an interface (programmatic) for Activities to communicate with the Service
- In both cases, if the service is not running it will be created
  - Note both are the **same** service
  - Different responsibilities for the lifecycle
    - **If I start it, I have to stop it. If OS starts it, OS stops it when it decides to**
    - Can do *both*


Call to startService()
onCreate()
onStartCommand()
Service running
The service is stopped by itself or a client
onDestroy()
Service shut down
Unbounded service
Call to bindService()
onCreate()
onBind()
Clients are bound to service
All clients unbind by calling unbindService()
onUnbind()
onDestroy()
Service shut down
Bounded service
Active Lifetime

Service Lifecycle

- By nature, services are singleton objects
  - “There can be only one”
- The Service sub-class object is instantiated if necessary
  - onCreate() is called
  - either onStartCommand or onBind will be called depending on how the service has been “called”
- onCreate / onStart / onBind are called in the context of the main UI thread
  - Must spawn a worker thread to do any significant work
- *Something* calls stopService()
  - Could be the OS again
  - How do we ensure we don’t lose work?
- onDestroy

Implementing Services

- Generic started service
  - Runs persistently
    - i.e. checking for emails
    - (Or stops itself when all work is done)
  - Receives messages asking for more work to be done
    - Delivered via onStartCommand
- IntentService
  - A simple, unbound service
    - Assumes we don’t have multiple requests that need to be handled concurrently
    - Creates a queue of work to be done
  - Handles one intent at a time to onHandleIntent()
    - Intents delivered via onStartCommand added to a queue
  - Stops the service after all start requests have been handled
  - I.e. sending emails
    - “fire and forget”
- RPC interface
  - Binding to services...

Let’s have a look…

Terminating Services

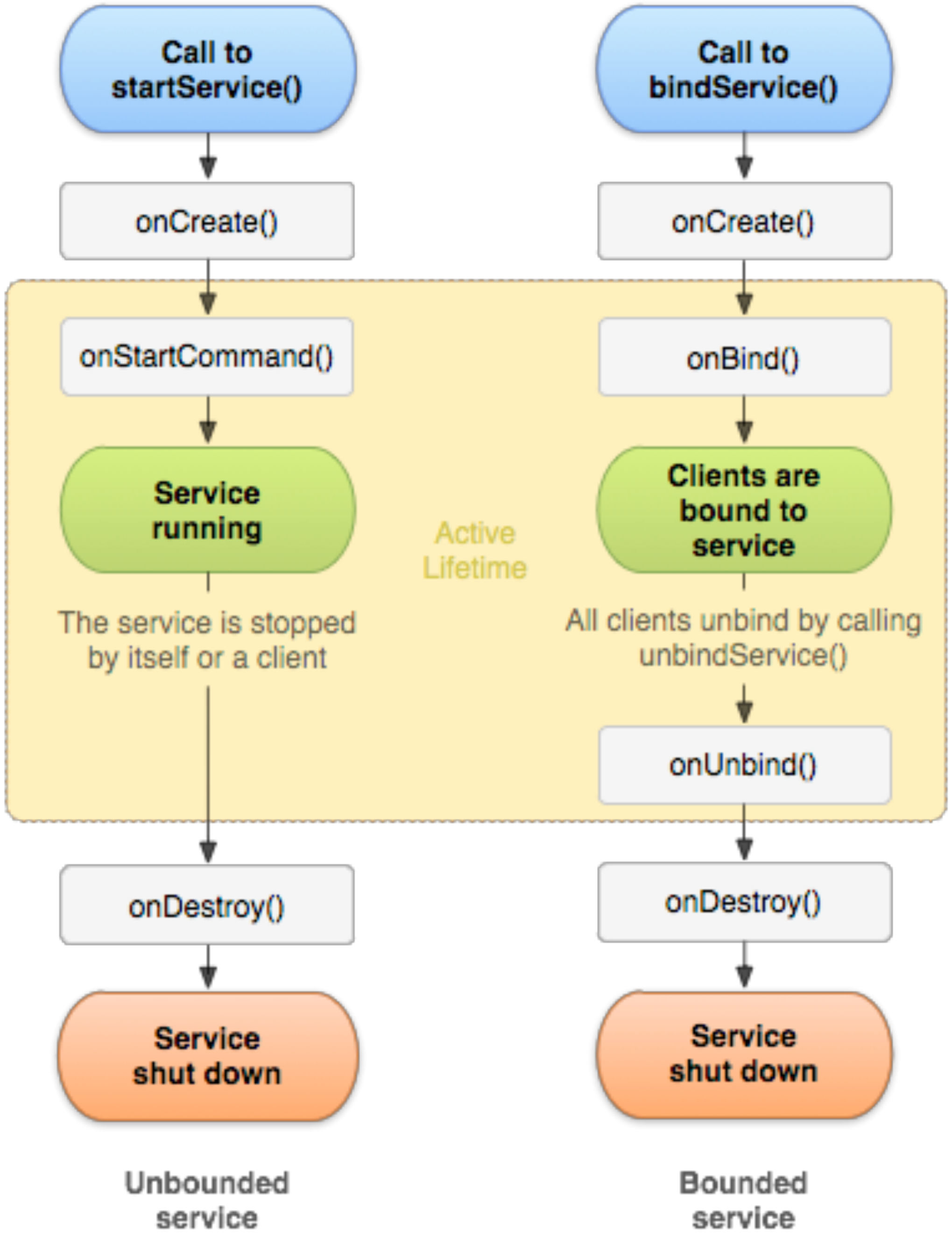
- A Service runs in the background indefinitely
  - Even if the component that started it is destroyed
- Termination of a service
  - Self-termination (calling stopSelf())
  - stopService() via an Intent
  - System termination
    - i.e. memory shortage – Last recently used again
- Avoiding termination
  - Foregrounding a Service
    - This is something the user should really know about
    - Active in the Status Bar / shows a Notification
    - Is treated as important as a foregrounded Activity
    - startForeground(…)
  - Background services are vulnerable
    - Android 8.0
      - Stopped by the system - why?

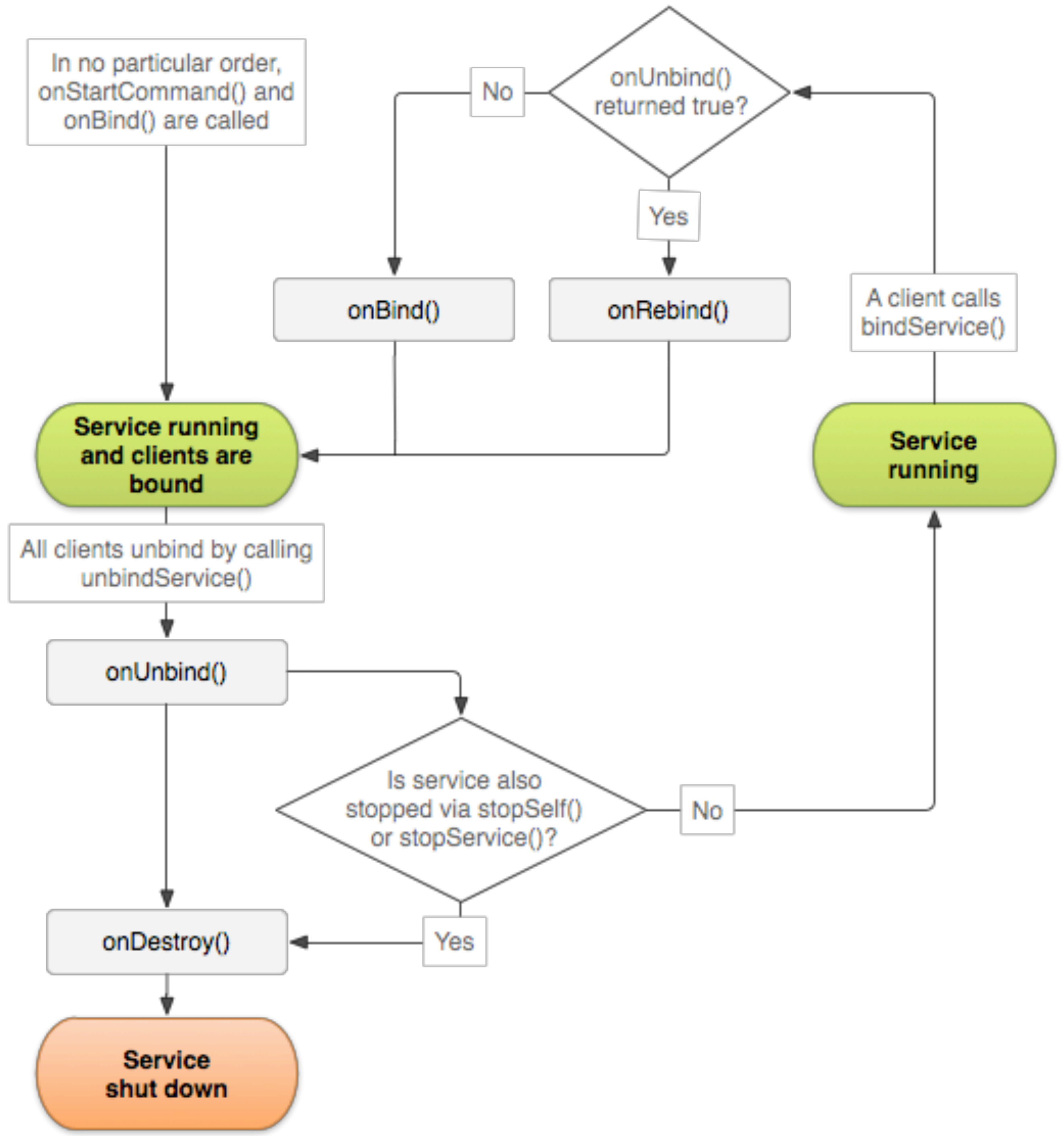
Terminating Services

- A Service runs in the background indefinitely
  - Even if the component that started it is destroyed
  - onStartCommand return value determines how the service should be continued if it is destroyed
- START_NOT_STICKY
  - After onStartCommand returns, do not recreate the service unless there are intents to deliver
- START_STICKY
  - Recreate the service and call onStartCommand again, but do not redeliver the last intent
- START_REDELIVER_INTENT
  - Recreate the service and call onStartCommand again, redeliver the last intent
    - Immediately resume the previous job, i.e. downloading a file

Notifications

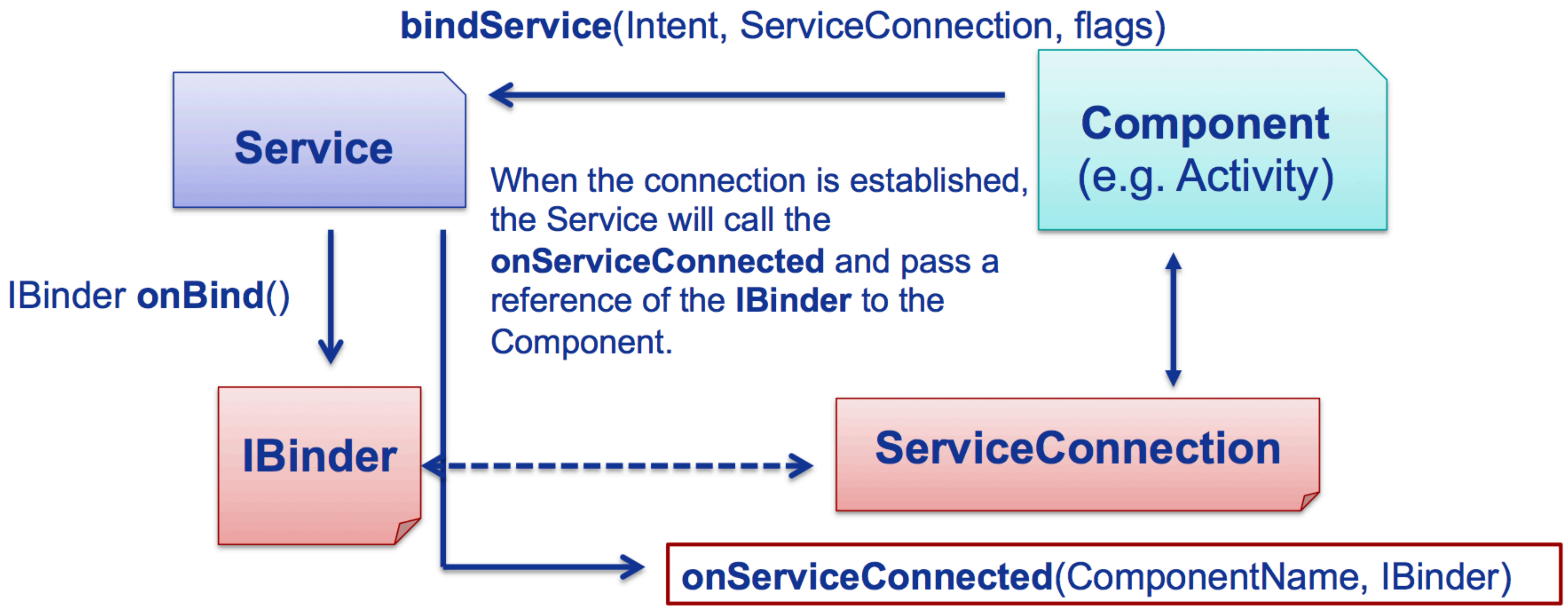
- But how do we notify the user that the Service is operating / has done something?
  - The original Activity may no longer exist
- Status bar notification
  - Maintained by the service
  - Can specify an Intent / Activity to launch if the user clicks on it
    - Return to the Activity that spawned the Service
    - Via a *Pending* Intent


Call to startService()
onCreate()
onStartCommand()
Service running
The service is stopped by itself or a client
onDestroy()
Service shut down
Unbounded service
Call to bindService()
onCreate()
onBind()
Clients are bound to service
All clients unbind by calling unbindService()
onUnbind()
onDestroy()
Service shut down
Bounded service
Active Lifetime


In no particular order, onStartCommand() and onBind() are called
onUnbind() returned true?
No
Yes
onBind()
onRebind()
A client calls bindService()
Service running and clients are bound
Service running
All clients unbind by calling unbindService()
onUnbind()
Is service also stopped via stopSelf() or stopService()?
No
Yes
onDestroy()
Service shut down

Bound Services

- If not explicitly started, will be started by the o/s
  - …when something binds to it
  - Then stopped if everything unbinds from it
  - What is it **is** explicitly started?
- Provide an interface for clients (Activities) to interact with a Service
  - Provide a programmatic interface for clients
  - Fast *and* stable?
- **Extending** the Binder class
  - Return an interface via the onBind method
  - Only for a Service used by the same application
    - Local Services only
      - i.e. the same process
    - Make method calls within the same JVM
- Binder object asynchronously provides a reference to the service that we can call methods on
  - Via *ServiceConnection*
  - Why asynchronous?


bindService(Intent, ServiceConnection, flags)
Service
Component
(e.g. Activity)
IBinder **onBind**()
IBinder
ServiceConnection
onServiceConnected(ComponentName, IBinder)

When the connection is established, the Service will call the **onServiceConnected** and pass a reference of the **IBinder** to the Component.

Let’s have a look…

References

- http://developer.android.com/guide/components/processes-and-threads.html

- http://developer.android.com/guide/components/services.html

- http://grepcode.com/file/repository.grepcode.com/java/ext/com.google.android/android/5.1.1_r1/android/app/IntentService.java#IntentService.ServiceHandler