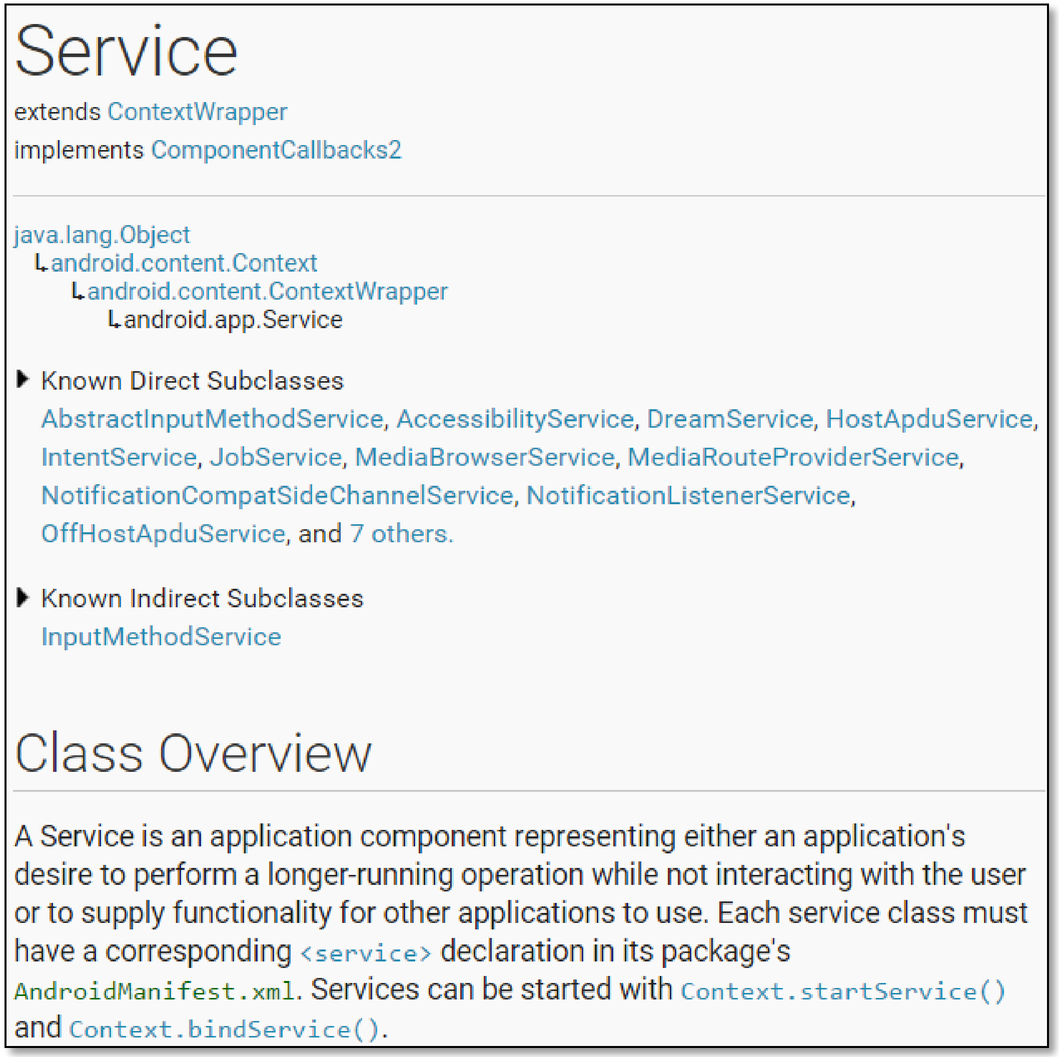
Steps for Implementing a Service (Part 1)

Steps for Implementing a Service



Service

extends ContextWrapper
implements ComponentCallbacks2

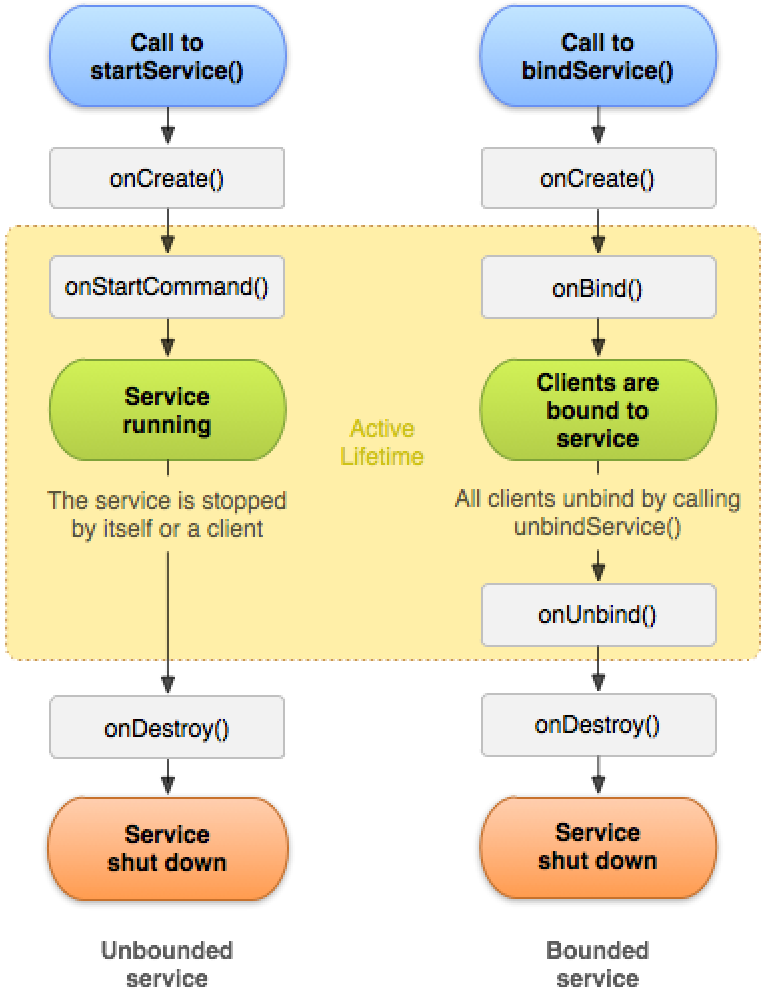
java.lang.Object
  ↳android.content.Context
    ↳android.content.ContextWrapper
      ↳android.app.Service

▸ Known Direct Subclasses
AbstractInputMethodService, AccessibilityService, DreamService, HostApduService, IntentService, JobService, MediaBrowserService, MediaRouteProviderService, NotificationCompatSideChannelService, NotificationListenerService, OffHostApduService, and 7 others.

▸ Known Indirect Subclasses
InputMethodService

Class Overview

A Service is an application component representing either an application's desire to perform a longer-running operation while not interacting with the user or to supply functionality for other applications to use. Each service class must have a corresponding `<service>` declaration in its package's `AndroidManifest.xml`. Services can be started with `Context.startService()` and `Context.bindService()`.



See developer.android.com/reference/android/app/Service.html

Steps for Implementing a Service

- Implementing a Service is similar to implementing an Activity

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

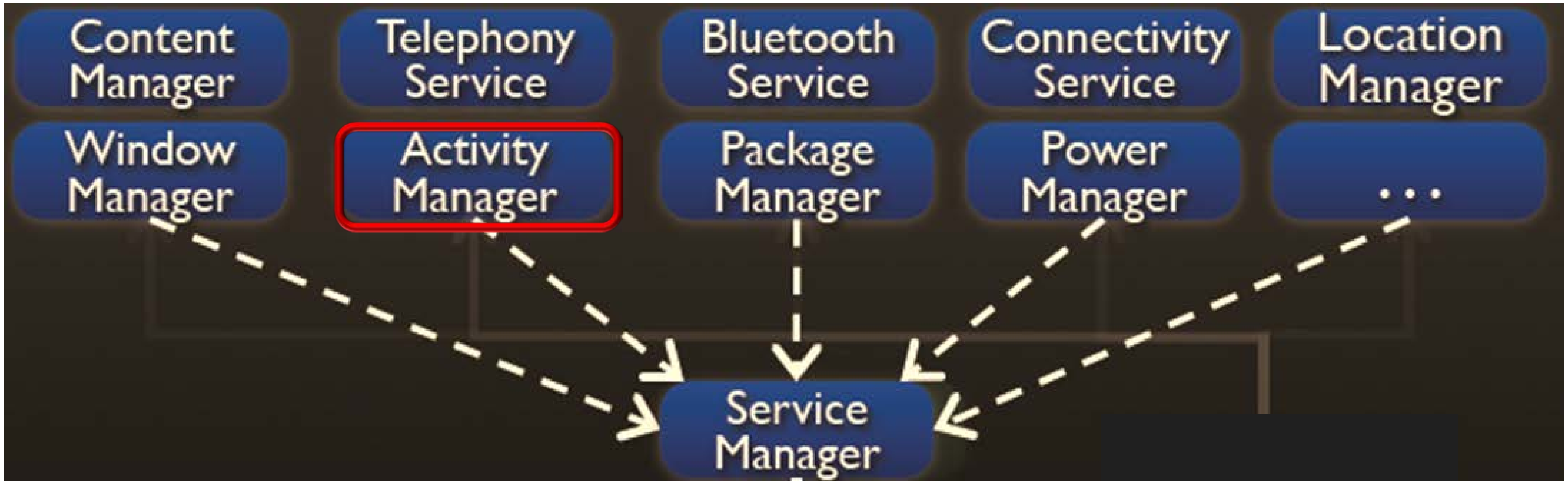
See Week 2 lectures at class.coursera.org/android-001/lecture

Steps for Implementing a Service

- Implementing a Service is similar to implementing an Activity

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

Services & Activities are both programmed via canonical framework techniques



See frameworks/base/services/java/com/android/server/am

Steps for Implementing a Service

- Implementing a Service is similar to implementing an Activity, e.g.
  - Extend the Android Service class

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

See frameworks/base/core/java/android/app/Service.java

Steps for Implementing a Service

- Implementing a Service is similar to implementing an Activity, e.g.
  - Extend the Android Service class

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

Service can be extended either directly or indirectly

See developer.android.com/reference/android/app/IntentService.html

Steps for Implementing a Service

- Implementing a Service is similar to implementing an Activity, e.g.
  - Extend the Android Service class
    - Defines Service-specific lifecycle hook methods

```
public class DownloadService
       extends Service {
  public void onCreate() { ... }
  public int onStartCommand
     (Intent intent,
      int flags, int startId) {
    ...
  }
  public abstract IBinder
    onBind(Intent intent) { ... }
  public boolean
    onUnbind(Intent intent) { ... }
  protected void onDestroy() {
    ...
  }
  ...
}
```

Android's Service framework dispatches these hook methods via "inversion of control"

See en.wikipedia.org/wiki/Inversion_of_control

Steps for Implementing a Service

- Implementing a Service is similar to implementing an Activity, e.g.
  - Extend the Android Service class
    - Defines Service-specific lifecycle hook methods
    - Selectively override lifecycle hook methods

Android's Service framework defines reusable structure & functionality that's specific to different type of Services

```
public class DownloadService
       extends Service {
  public void onCreate() { ... }
  public int onStartCommand
      (Intent intent,
       int flags, int startId) {
    ...
  }
  public abstract IBinder
    onBind(Intent intent) { ... }
  public boolean
    onUnbind(Intent intent) { ... }
  protected void onDestroy() {
    ...
  }
  ...
}
```

Steps for Implementing a Service

- Implementing a Service is similar to implementing an Activity, e.g.
  - Extend the Android Service class
    - Defines Service-specific lifecycle hook methods
    - Selectively override lifecycle hook methods
    - Define other methods & nested classes needed to implement the Service

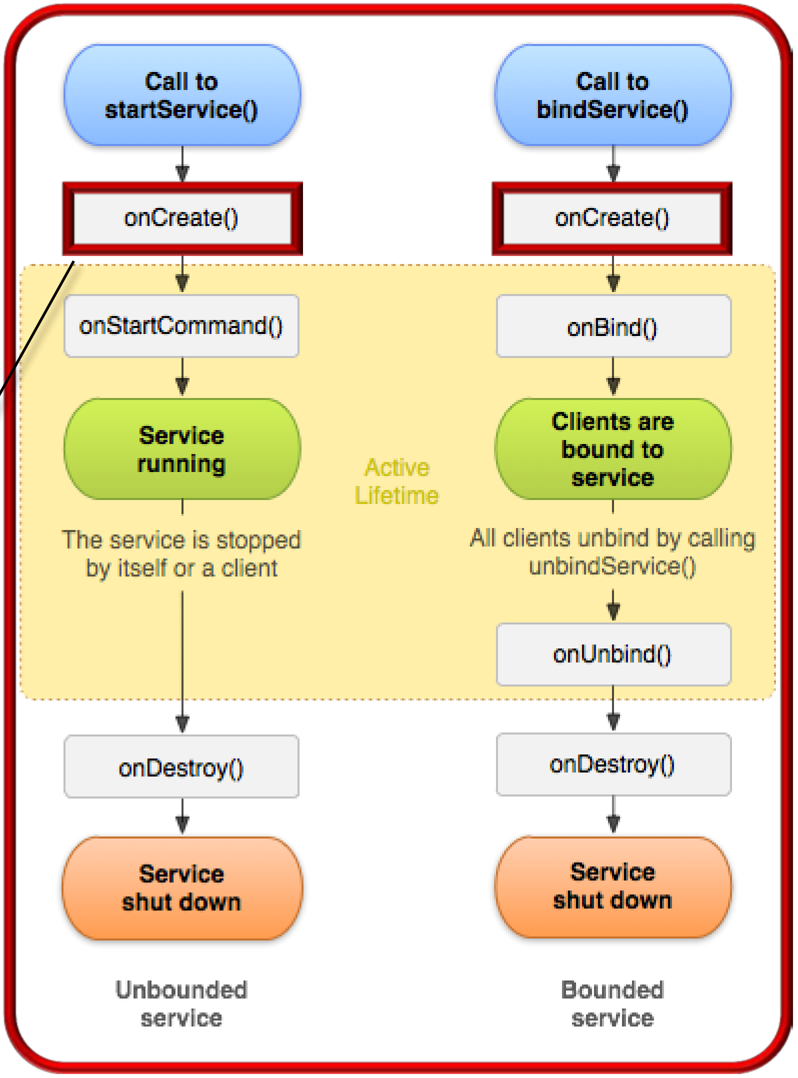
```
public class DownloadService
       extends Service {
  ...
  public static Intent
    makeIntent() {...}

  private final class
    ServiceHandler extends Handler
  {...}
  ...
}
```

These methods & classes implement a Service's application logic, as well as concurrency & communication behaviors

Steps for Implementing a Service

- Implementing a Service is similar to implementing an Activity
- Android communicates state changes to a Service by calling its lifecycle hook methods
  - **onCreate()** – called when Service is first launched, by any means

This method is typically used to initialize the Service

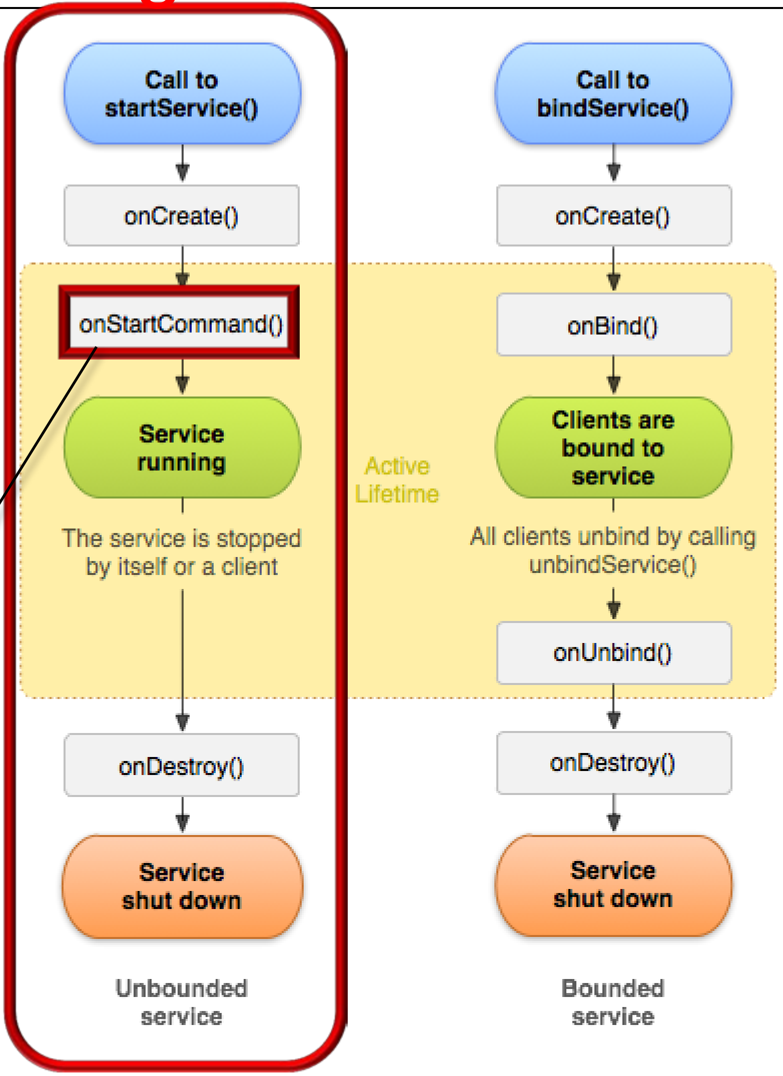


Steps for Implementing a Service (Part 2)

Steps for Implementing a Service

- Implementing a Service is similar to implementing an Activity
- Android communicates state changes to a Service by calling its lifecycle hook methods
  - **onCreate()** – called when Service is first launched, by any means
  - **onStartCommand()** – called each time a Started Service is sent an Intent via startService()

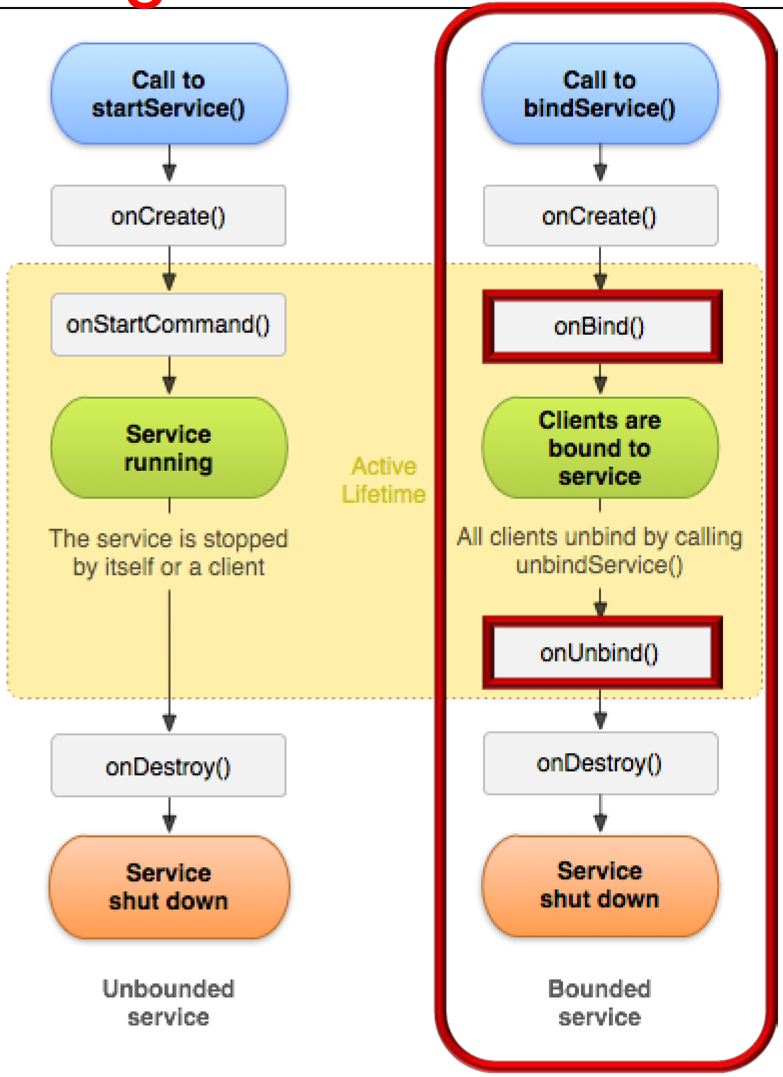
This method receives the Intent passed by the client's call to startService()



onStartCommand() is typically used in conjunction with the concurrency model a Service applies to perform its processing

Steps for Implementing a Service

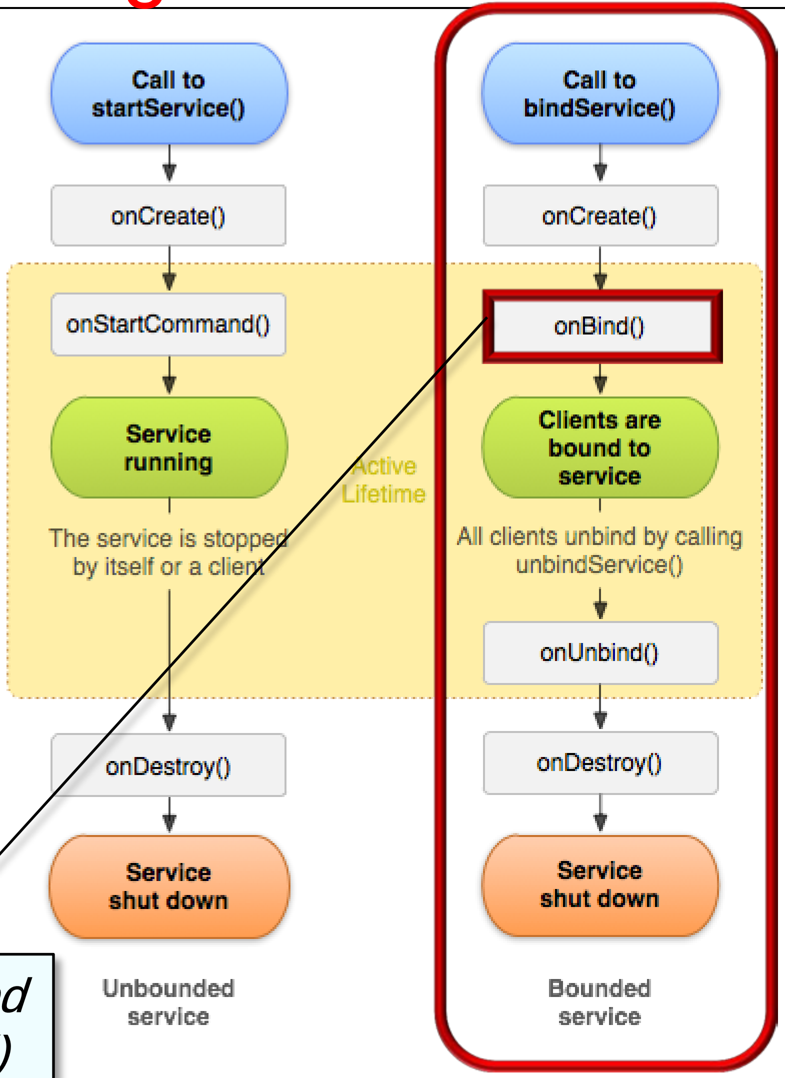
- Implementing a Service is similar to implementing an Activity
- Android communicates state changes to a Service by calling its lifecycle hook methods
  - **onCreate()** – called when Service is first launched, by any means
  - **onStartCommand()** – called each time a Started Service is sent an Intent via startService()
  - **onBind() / onUnbind** – called when client binds/unbinds to Bound Service via bindService()/unBindService()



Steps for Implementing a Service

- Implementing a Service is similar to implementing an Activity
- Android communicates state changes to a Service by calling its lifecycle hook methods
  - **onCreate()** – called when Service is first launched, by any means
  - **onStartCommand()** – called each time a Started Service is sent an Intent via startService()
  - **onBind() / onUnbind** – called when client binds/unbinds to Bound Service via bindService()/unBindService()

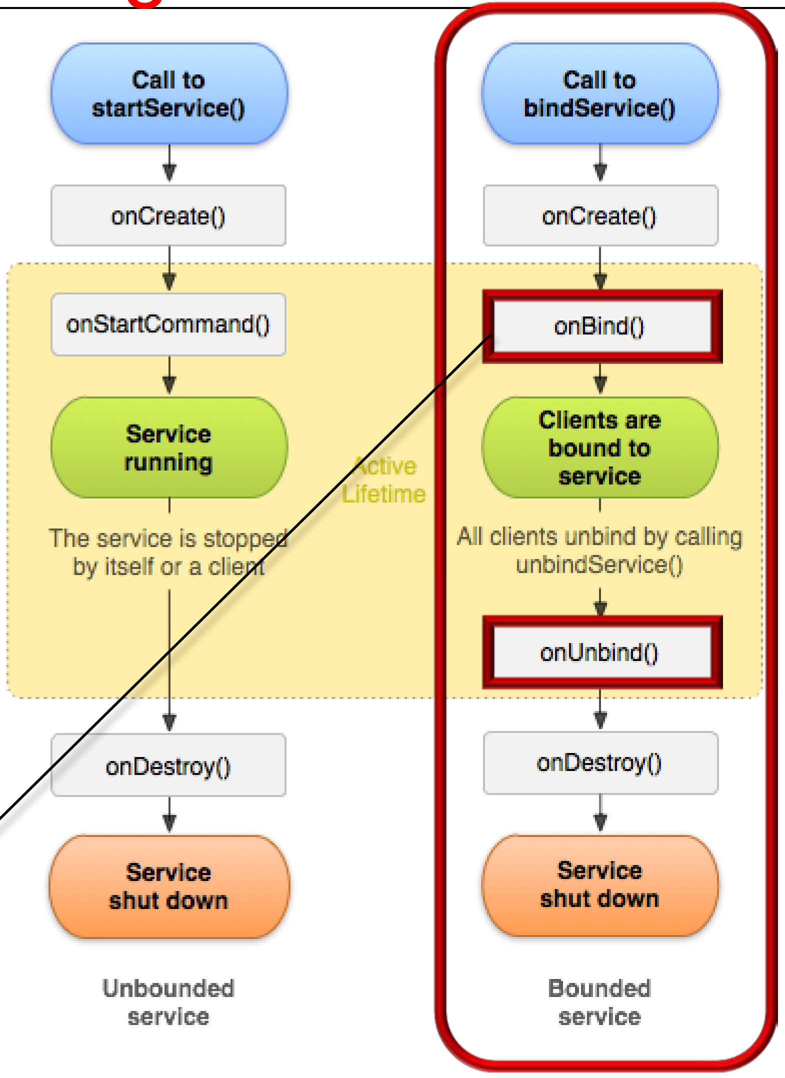
onBind() receives the Intent passed by the client's call to bindService()



Steps for Implementing a Service

- Implementing a Service is similar to implementing an Activity
- Android communicates state changes to a Service by calling its lifecycle hook methods
  - **onCreate()** – called when Service is first launched, by any means
  - **onStartCommand()** – called each time a Started Service is sent an Intent via startService()
  - **onBind() / onUnbind** – called when client binds/unbinds to Bound Service via bindService()/unBindService()

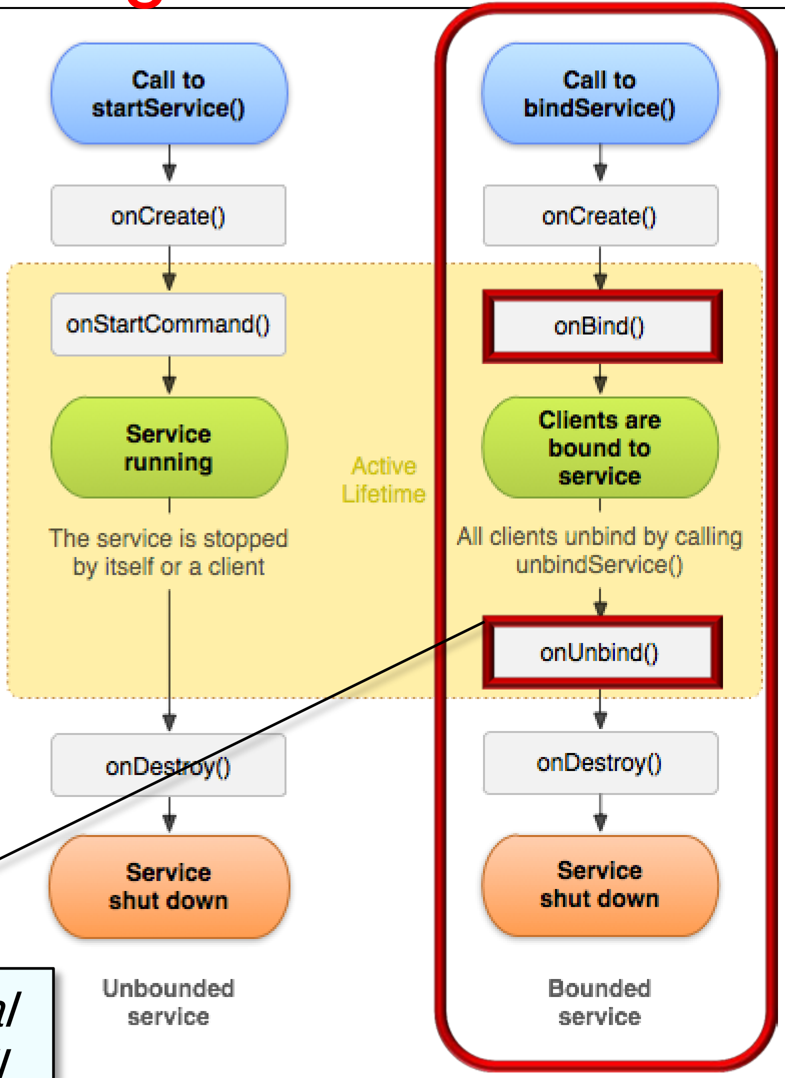
onBind() is a factory method that returns an IPC channel to the client



See upcoming part on "Overview of the AIDL & Binder Framework"

Steps for Implementing a Service

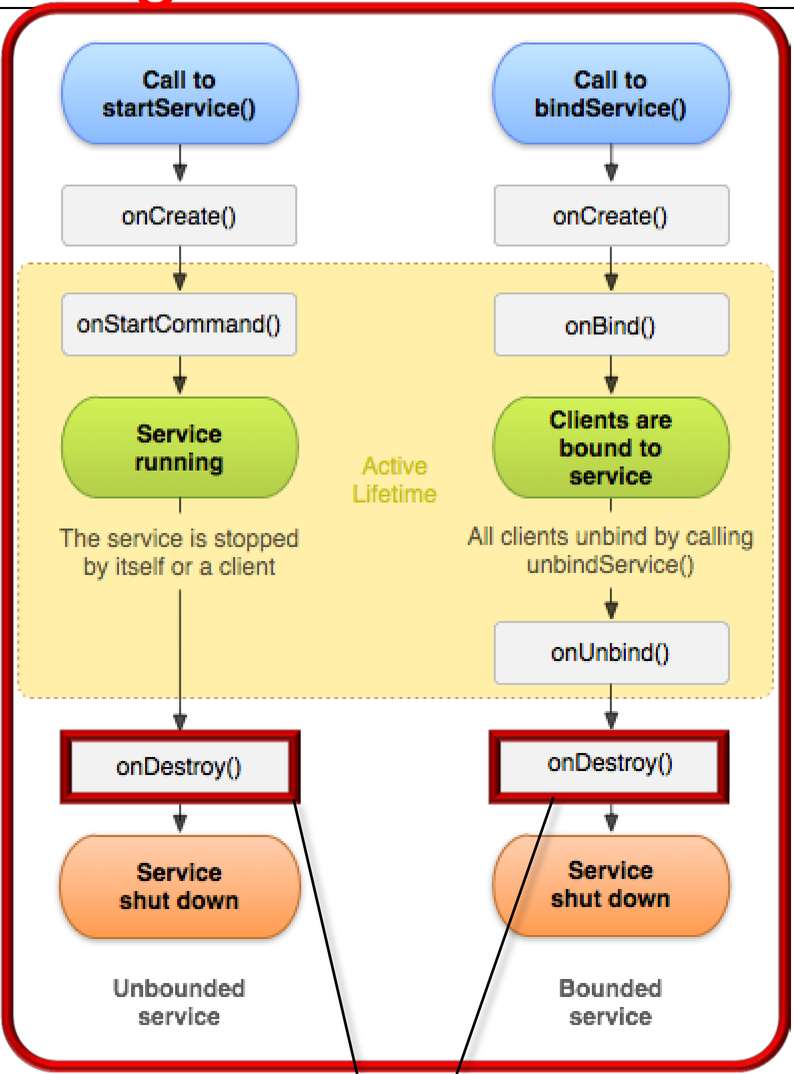
- Implementing a Service is similar to implementing an Activity
- Android communicates state changes to a Service by calling its lifecycle hook methods
  - **onCreate()** – called when Service is first launched, by any means
  - **onStartCommand()** – called each time a Started Service is sent an Intent via startService()
  - **onBind() / onUnbind** – called when client binds/unbinds to Bound Service via bindService()/unBindService()



onUnbind() is a disposal method called when all clients have disconnected

Steps for Implementing a Service

- Implementing a Service is similar to implementing an Activity
- Android communicates state changes to a Service by calling its lifecycle hook methods
  - **onCreate()** – called when Service is first launched, by any means
  - **onStartCommand()** – called each time a Started Service is sent an Intent via startService()
  - **onBind() / onUnbind** – called when client binds/unbinds to Bound Service via bindService()/unBindService()
  - **onDestroy()** – called as Service is being shut down



This disposal method cleans up any resources held by the Service