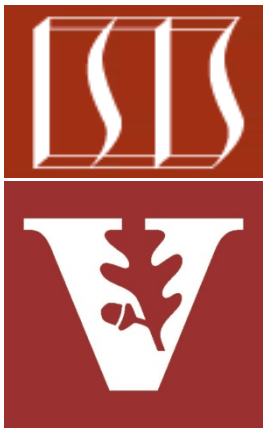
Android Services & Local IPC: Introduction

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt



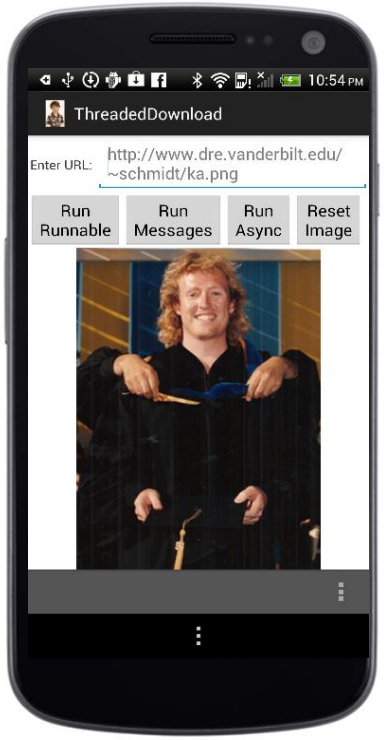
Professor of Computer Science

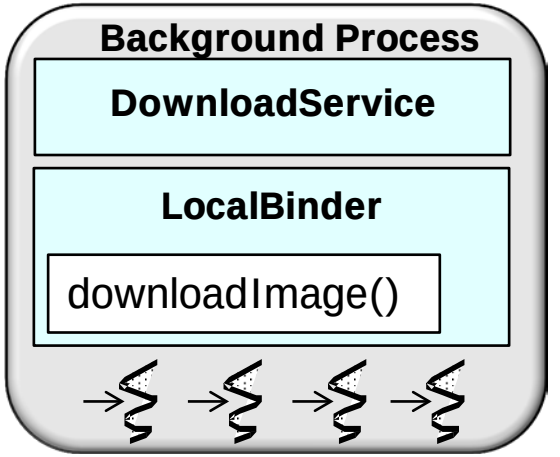
Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Introduction

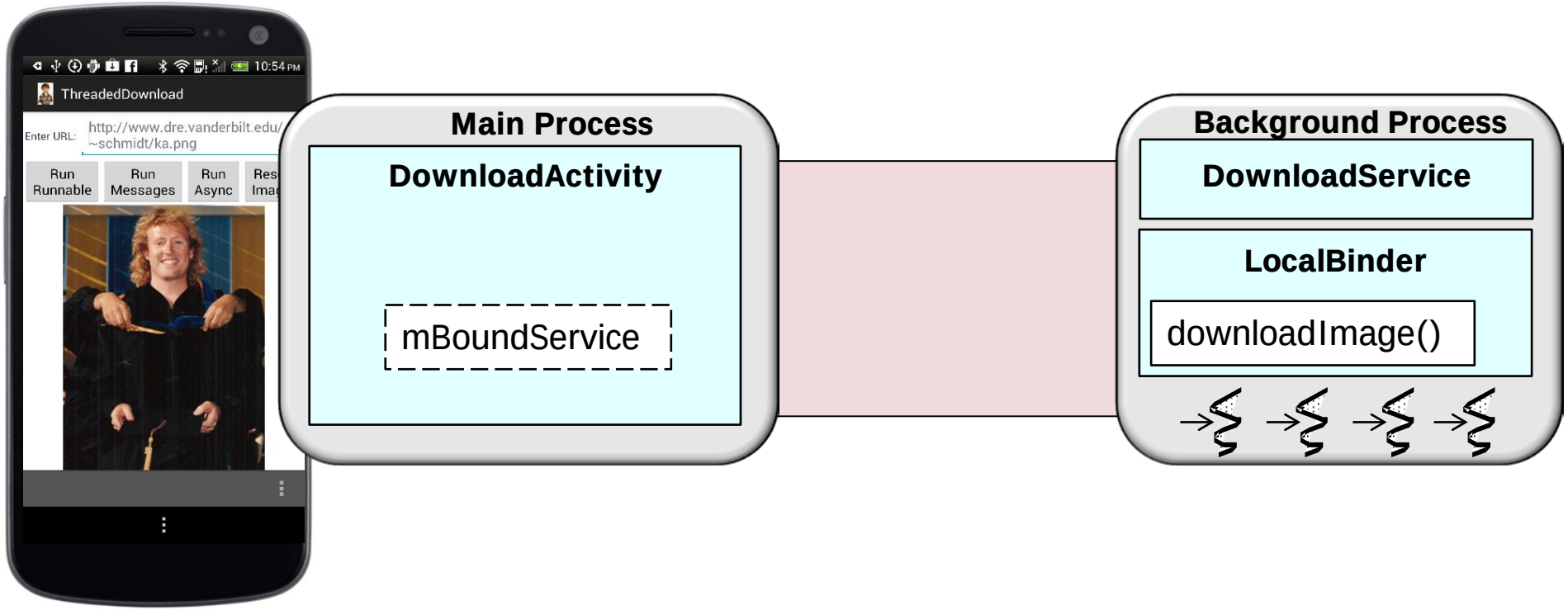
- Services don't have a visual user interface & often run in the background in a separate background thread or process





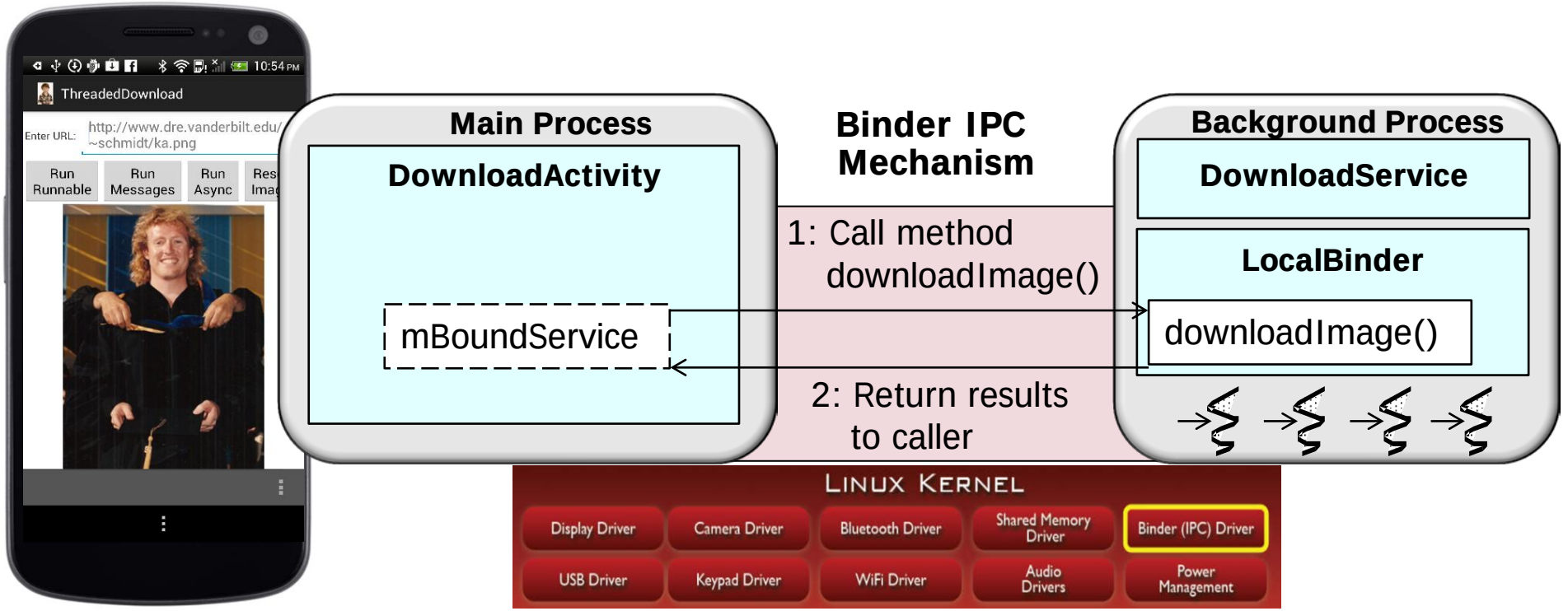
Introduction

- Services don't have a visual user interface & often run in the background in a separate background thread or process
  - Activities use Services to perform long-running operations or access remote resources on behalf of users



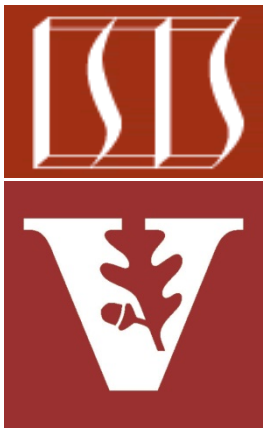
Introduction

- Services don't have a visual user interface & often run in the background in a separate background thread or process
- Activities & Services interact via IPC mechanisms that are optimized for inter-process communication within a mobile device
  - e.g., the Android Interface Definition Language (AIDL) & Binder framework



Android Services & Local IPC: Overview of Services

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt



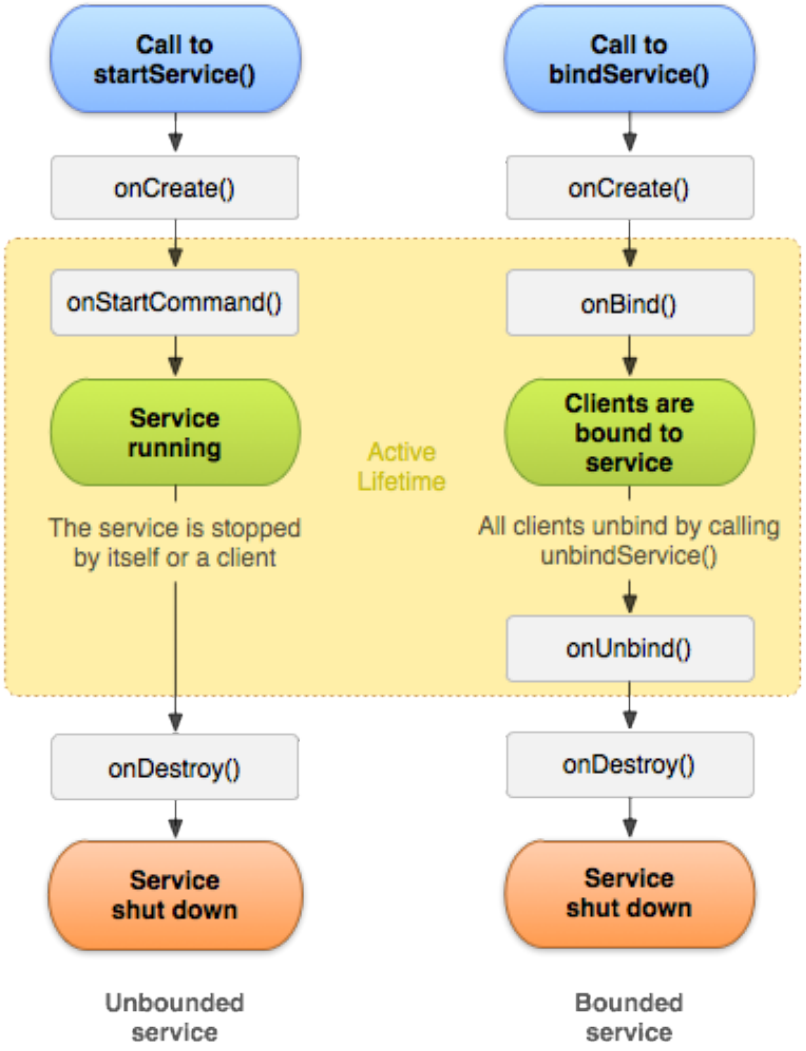
Professor of Computer Science

Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

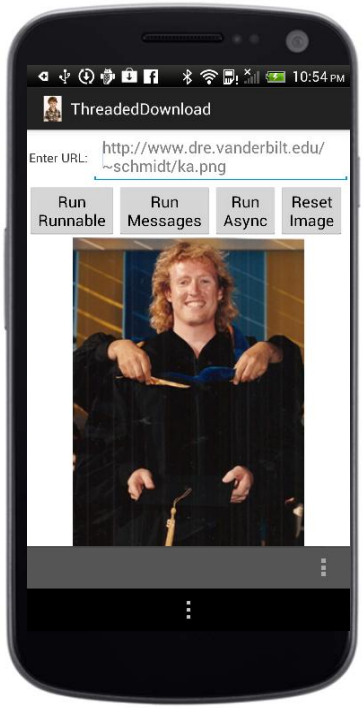
Learning Objectives in this Part of the Module

- Understand what a Service is & what different types of Services Android supports



Overview of a Service

- A Service is an Android component that can perform long-running operations in the background
  - e.g., a service might handle e-commerce transactions, play music, download a file, interact with a content provider, run tasks periodically, etc.



Download Service

developer.android.com/guide/components/services.html has more info

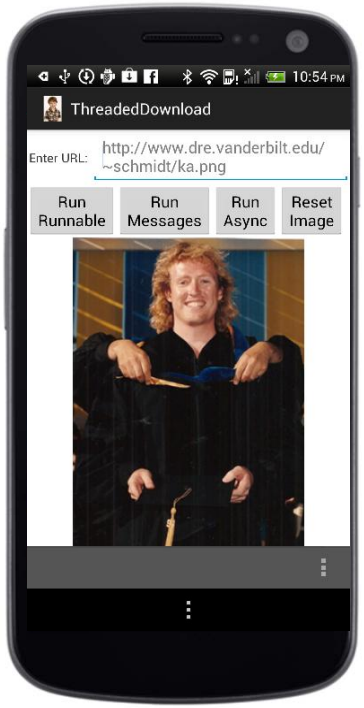
Overview of a Service

- A Service is an Android component that can perform long-running operations in the background
- Another Android component can start a Service
  - It will continue to run in the background even if the user switches to another app/activity

A Service does not provide direct access to the user interface

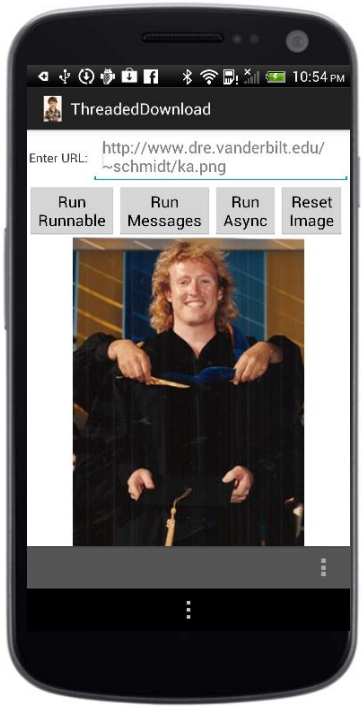
Download Activity

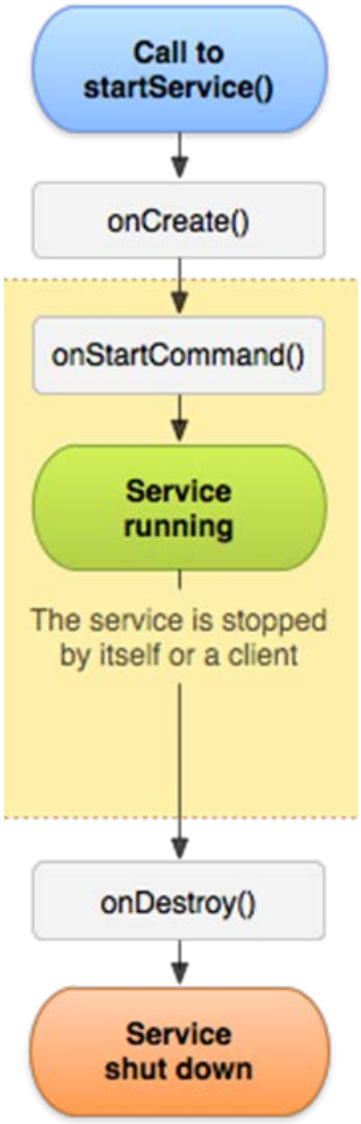
Download Service

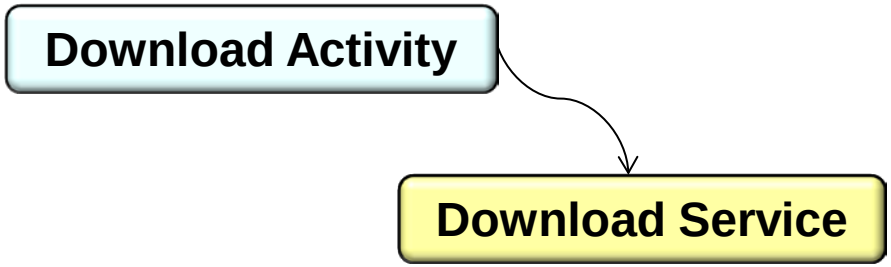


Overview of a Service

- A Service is an Android component that can perform long-running operations in the background
- Another Android component can start a service
- There are two types of Services
  - *Started Service* – Often performs a single operation & might not return a result to the caller directly

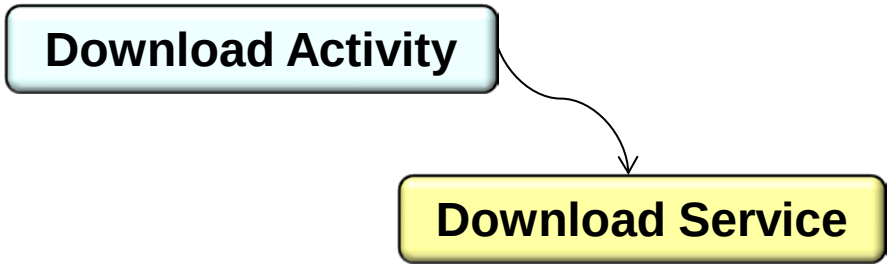


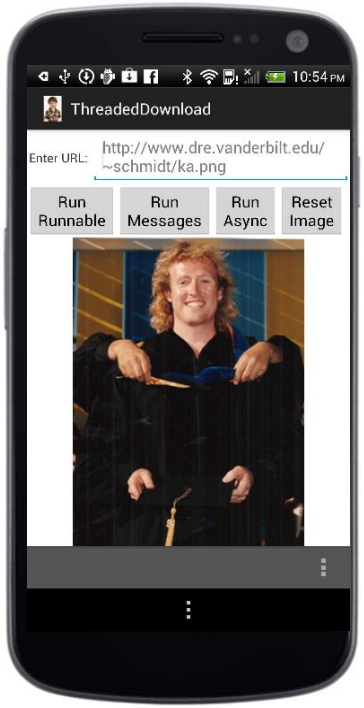


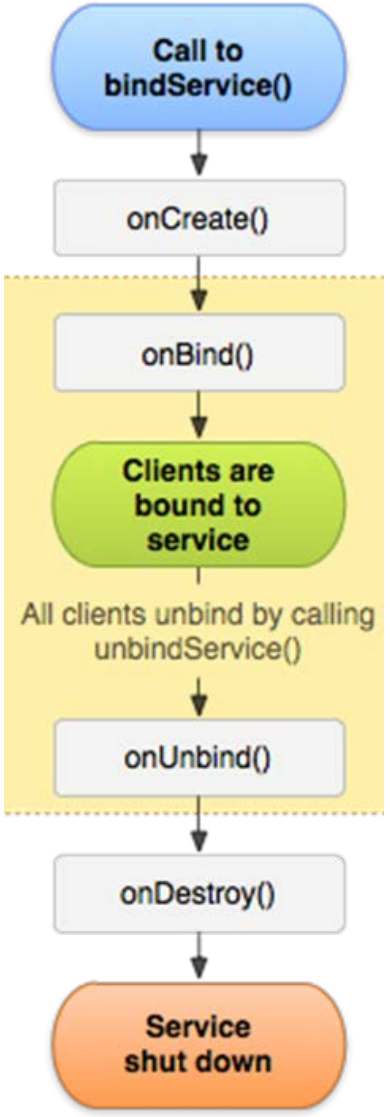


Overview of a Service

- A Service is an Android component that can perform long-running operations in the background
- Another Android component can start a service
- There are two types of Services
  - *Started Service* – Often performs a single operation & might not return a result to the caller directly
  - *Bound Service* – Provides a client-server interface that allows for a conversation with the Service







Overview of Started Services

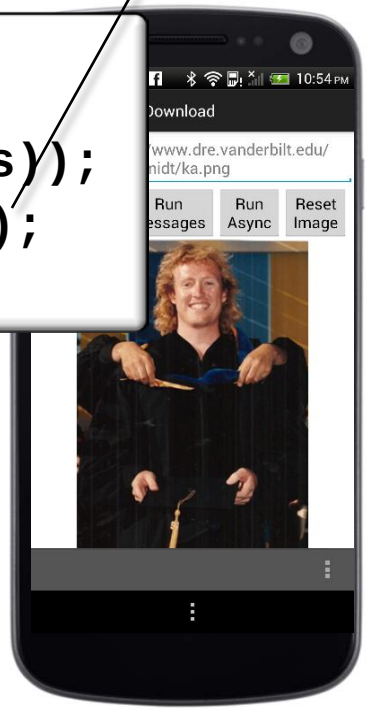
- A started service is one that another component starts by calling startService()

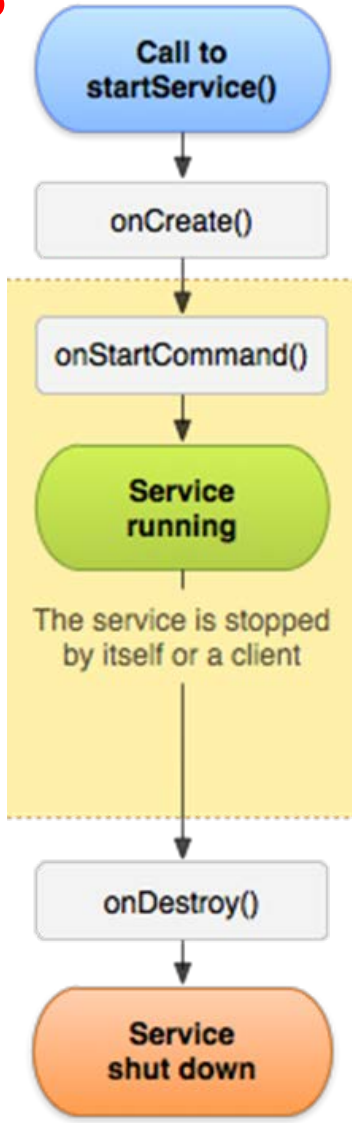
Parameters can be passed as "extras" to the Intent used to start the Service

```
Intent intent = new Intent
  (this,
   ThreadedDownloadService.class));
intent.putExtra("URL", imageUrl);
startService(intent);
```

Download Activity

Download Service

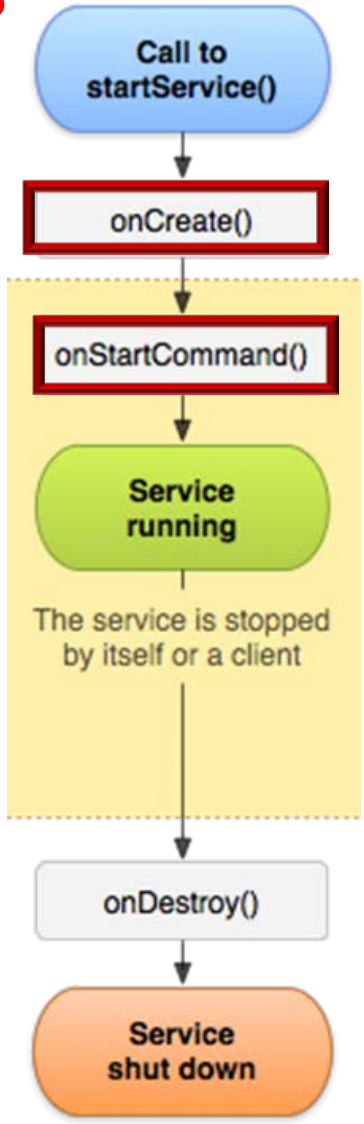




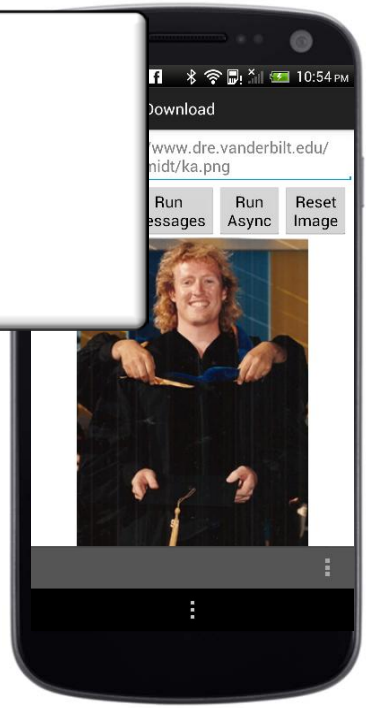
developer.android.com/guide/components/services.html#CreatingStartedService

Overview of Started Services

- A started service is one that another component starts by calling startService()
  - This results in a call to the Service's onCreate() & onStartCommand() hook methods




```
public class DownloadService
             extends Service {
  public int onStartCommand
    (Intent intent, int flags,
     int startId) { ... }
```



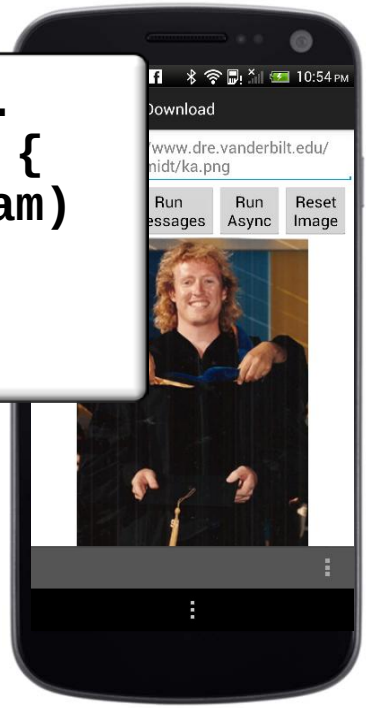
Download Activity

Download Service

Overview of Started Services

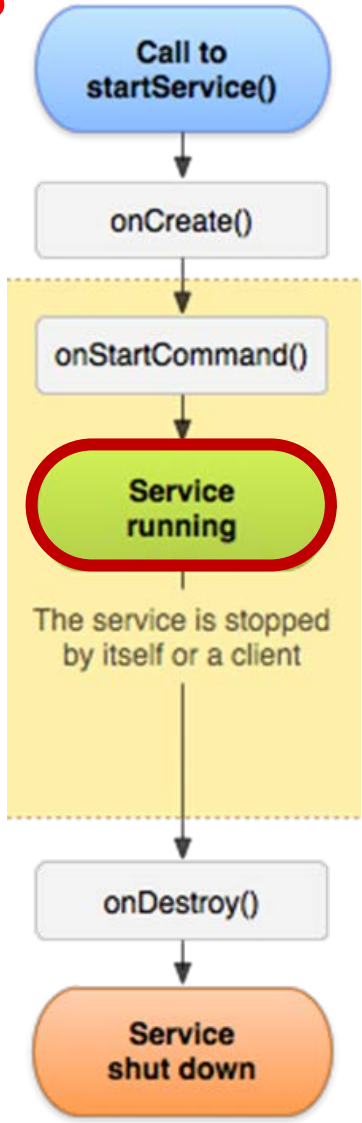
- A started service is one that another component starts by calling startService()
- A started service often performs a single operation & might not even return a result to the caller
  - e.g., it might download or upload a file over TCP

```
public class DownloadService ...
  String downloadFile (Uri uri) {
    InputStream in = (InputStream)
      new URL(uri.toString()).
        getContent();
    ...
```



Download Activity

Download Service



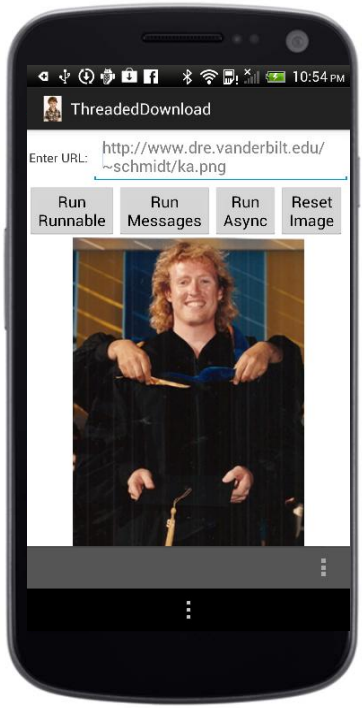
Overview of Started Services

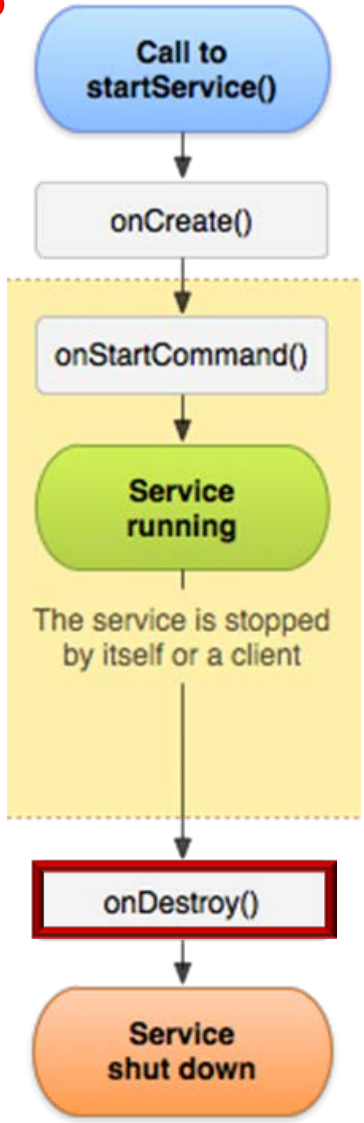
- A started service is one that another component starts by calling startService()
- A started service often performs a single operation & might not even return a result to the caller
- When the operation is done, the service can be stopped

A service can stop itself when its job is done by calling stopSelf(), or another component can stop it by calling stopService()

Download Activity

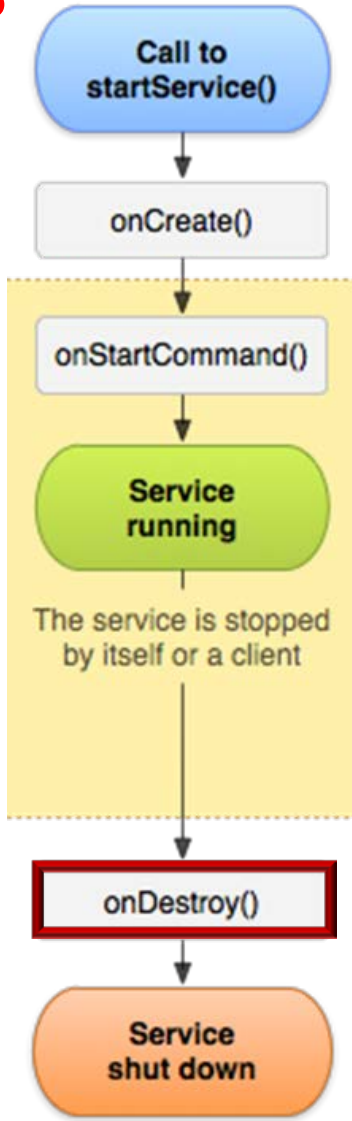
Download Service





Overview of Started Services

- A started service is one that another component starts by calling startService()
- A started service often performs a single operation & might not even return a result to the caller
- When the operation is done, the service can be stopped
- Examples of Android Started Services
  - *SMS & MMS Services*
    - Manage messaging operations, such as sending data, text, & pdu messages
  - *AlertService*
    - Handle calendar event reminders

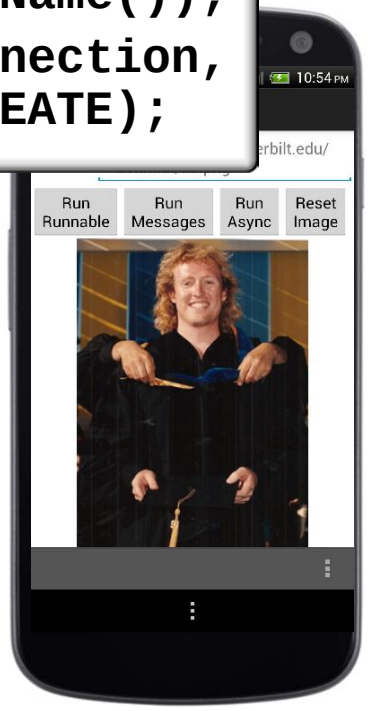


See packages/apps in Android source code for many services

Overview of Bound Services

- A bound service is one that allows app components to bind to it by calling bindService() to create a long-standing connection

```
Intent intent = new
  Intent(IDownloadSync.class.getName());
bindService(intent, this.syncConnection,
            Context.BIND_AUTO_CREATE);
```



Download Activity

Download Service

Overview of Bound Services

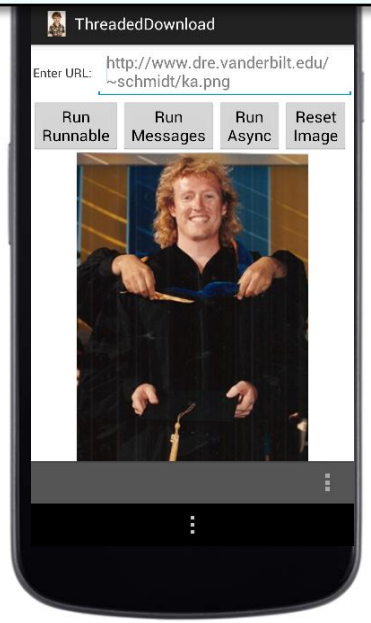
- A bound service allows app components to bind to it by calling bindService() to create a long-standing connection
  - This results in a call to the Service's onCreate() & onBind() hook methods

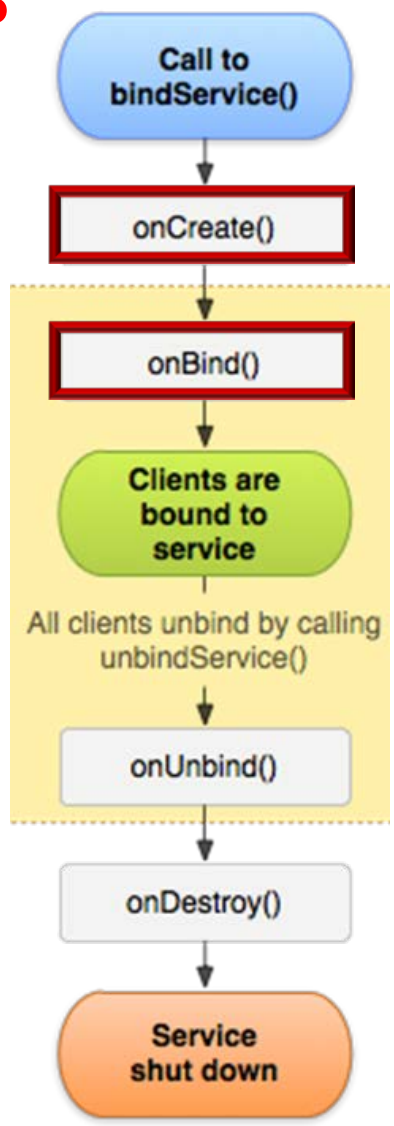
Returns an IBinder that defines the API for communicating with the Service

```
public IBinder onBind
  (Intent intent) {
  return this.binder;
}
```

Download Activity

Download Service



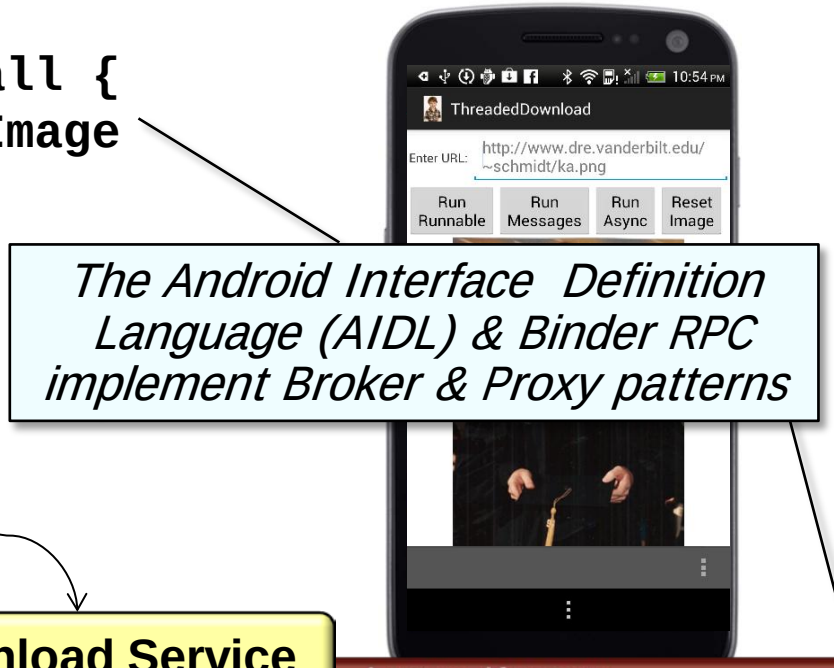


An interesting callback-driven protocol is used to establish a connection

Overview of Bound Services

- A bound service allows app components to bind to it by calling bindService() to create a long-standing connection
- A bound service offers a client-server interface that allows components to interact with the service, send requests, get results across processes via IPC

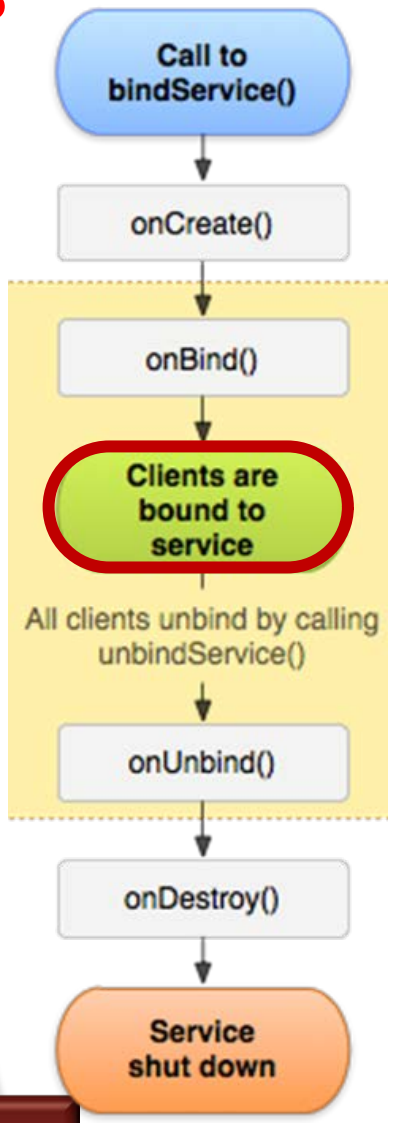
```
interface DownloadCall {
    String downloadImage
      (in Uri uri);
}
```



The Android Interface Definition Language (AIDL) & Binder RPC implement Broker & Proxy patterns

Download Activity

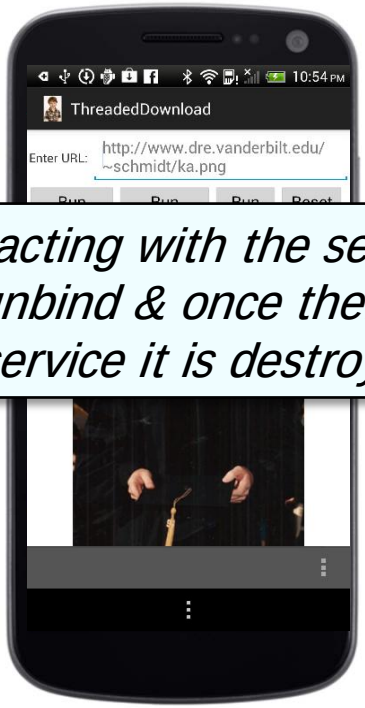
Download Service



Overview of Bound Services

- A bound service allows app components to bind to it by calling bindService() to create a long-standing connection
- A bound service offers a client-server interface that allows components to interact with the service, send requests, get results across processes via IPC
- A bound service runs only as long as another application component is bound to it

When a client is done interacting with the service, it calls unbindService() to unbind & once there are no clients bound to the service it is destroyed



Download Activity

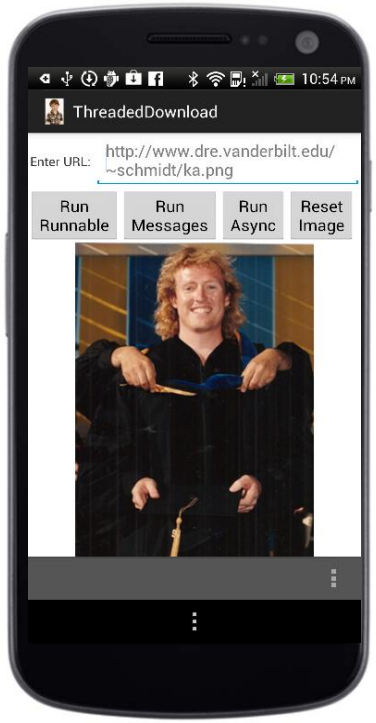
Download Service

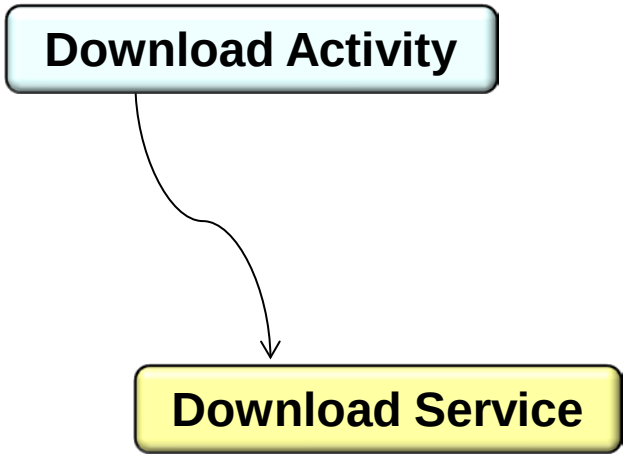
Overview of Bound Services

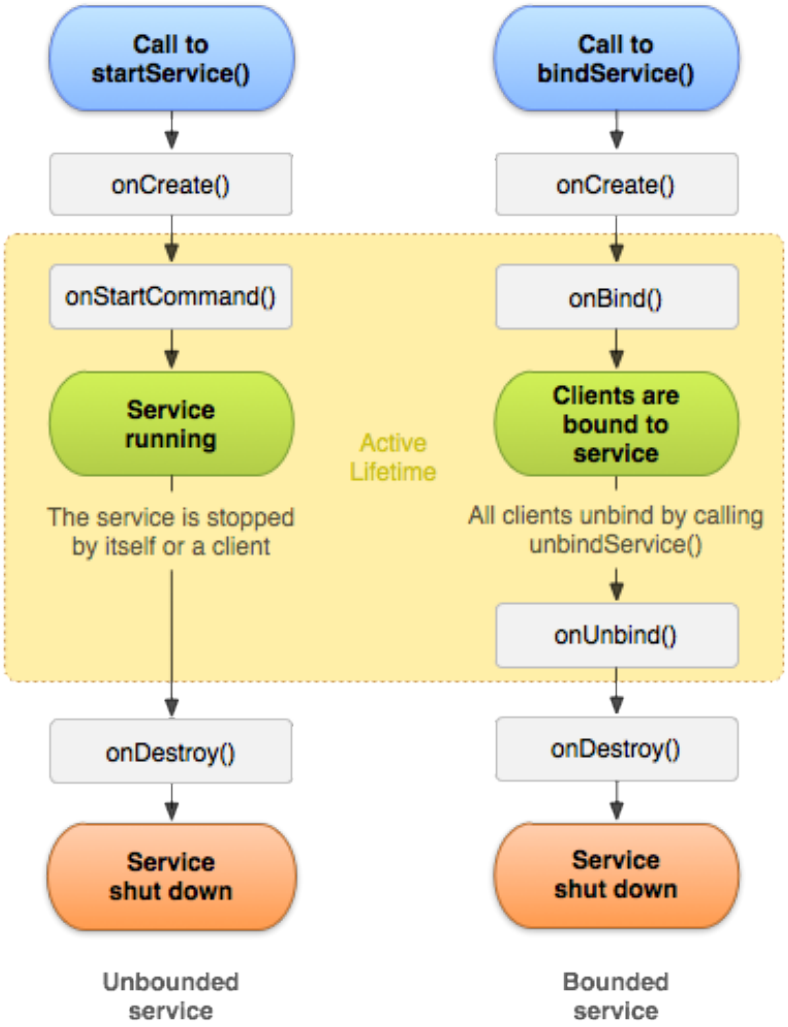
- A bound service allows app components to bind to it by calling bindService() to create a long-standing connection
- A bound service offers a client-server interface that allows components to interact with the service, send requests, get results across processes via IPC
- A bound service runs only as long as another application component is bound to it
- Examples of Android Started Services
  - *BluetoothHeadsetService*
    - Provides Bluetooth Headset & Handsfree profile, as a service in the Phone application
  - *MediaPlaybackService*
    - Provides "background" audio playback capabilities
  - *Exchange Email Services*
    - Manage email operations, such as sending messages

Summary

- Apps can use Services to implement long-running operations in the background

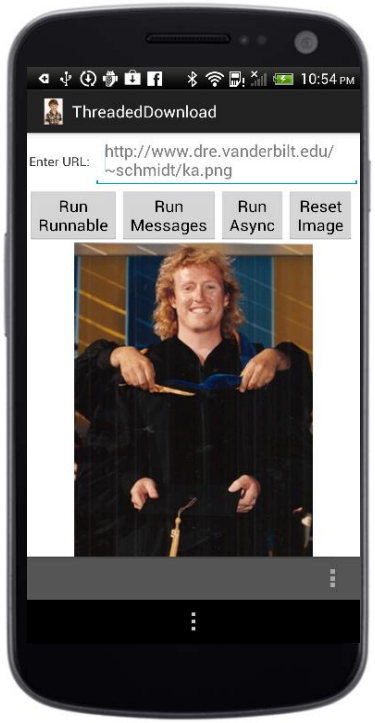


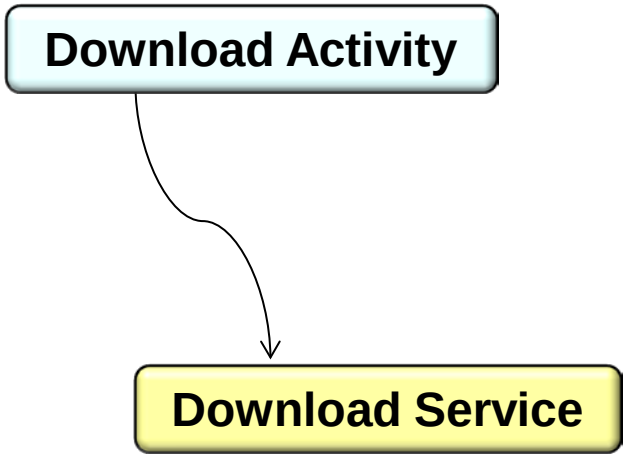


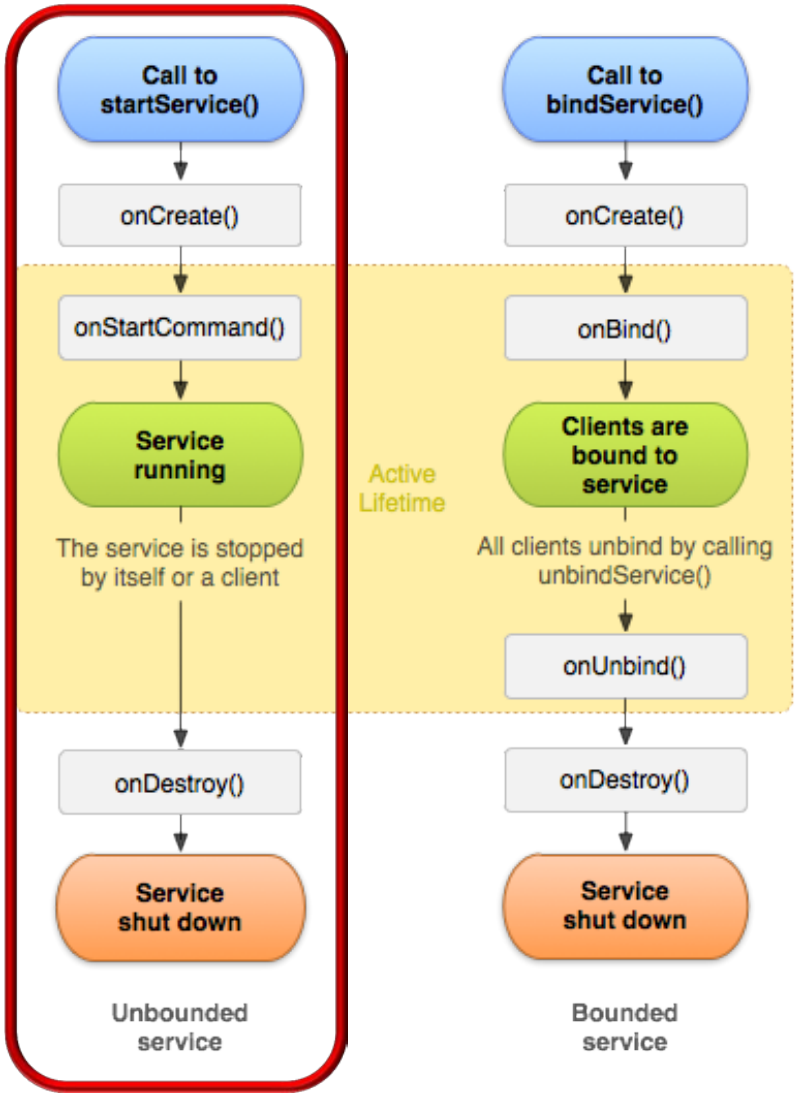


Summary

- Apps can use Services to implement long-running operations in the background
- Started Services are simple to program

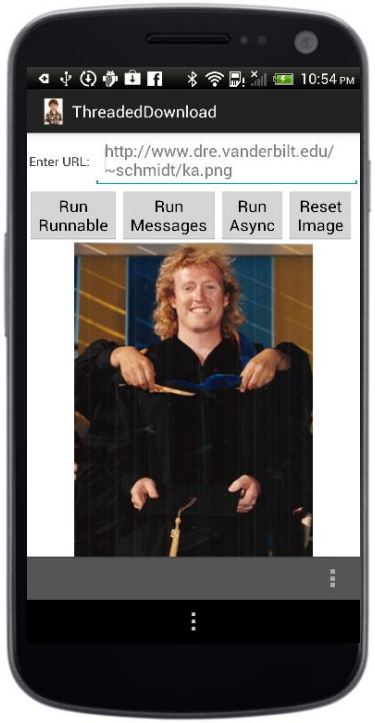


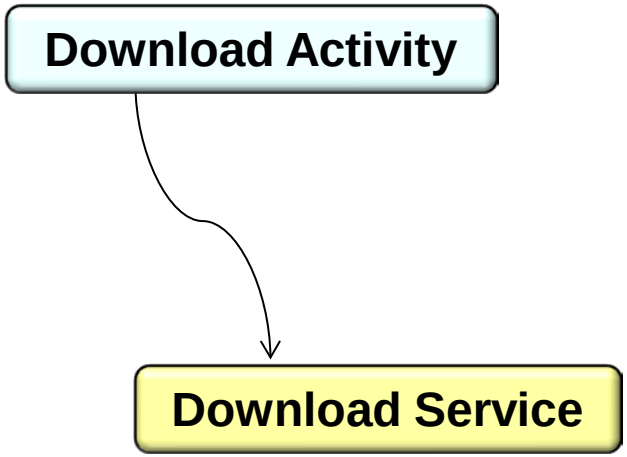


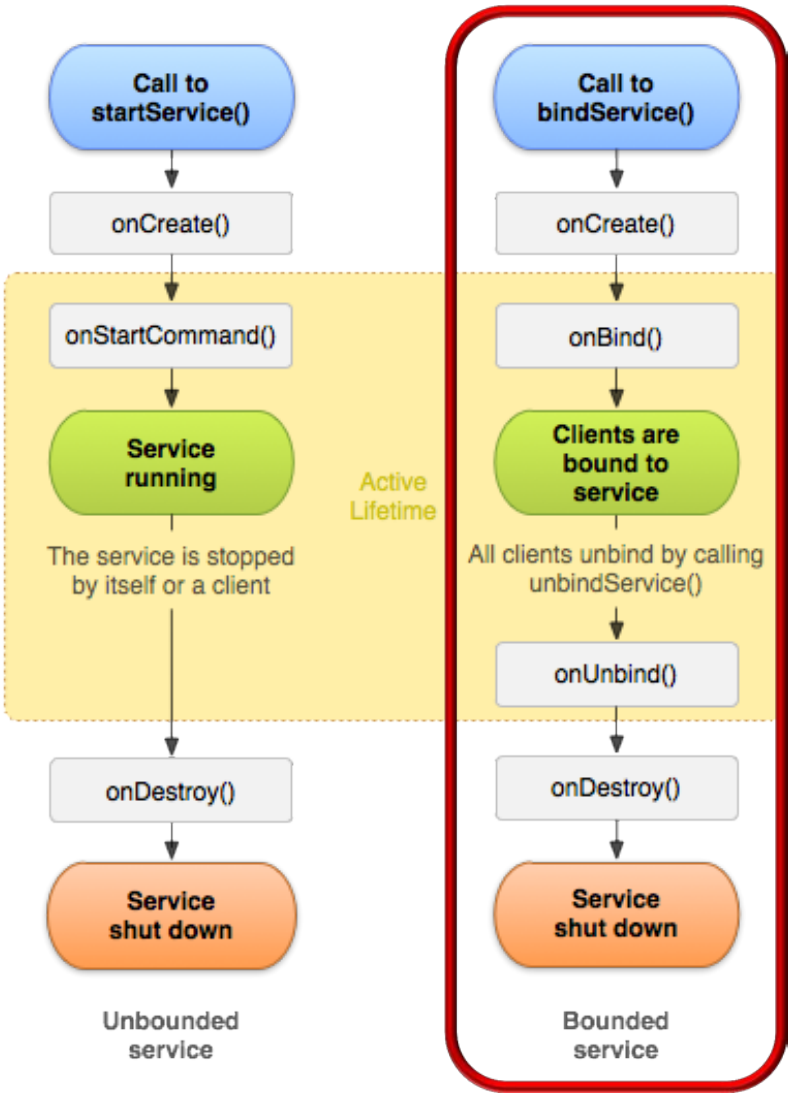


Summary

- Apps can use Services to implement long-running operations in the background
- Started Services are simple to program
- Bound Services provide more powerful communication models

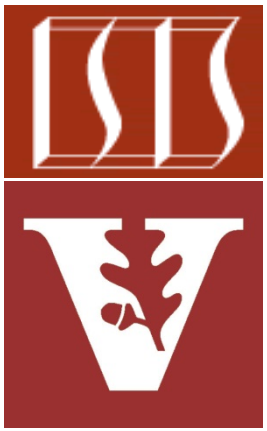






Android Services & Local IPC: Programming Started Services

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt



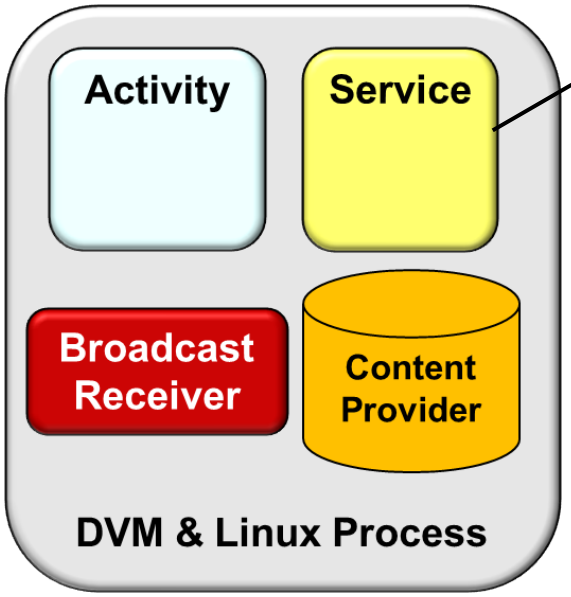
Professor of Computer Science

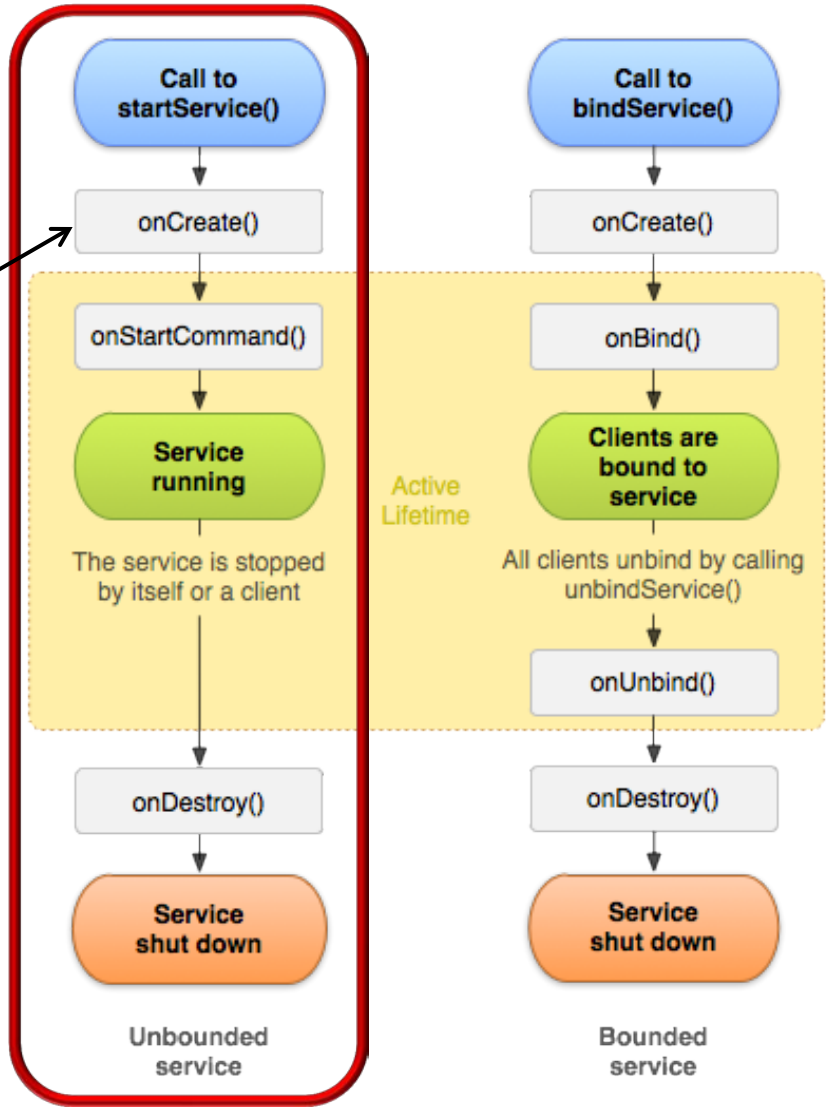
Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Module

- Understand how to subclass Service & implement the hook methods it defines to manage its various lifecycle states





We'll emphasize commonalities & variabilities in our discussion

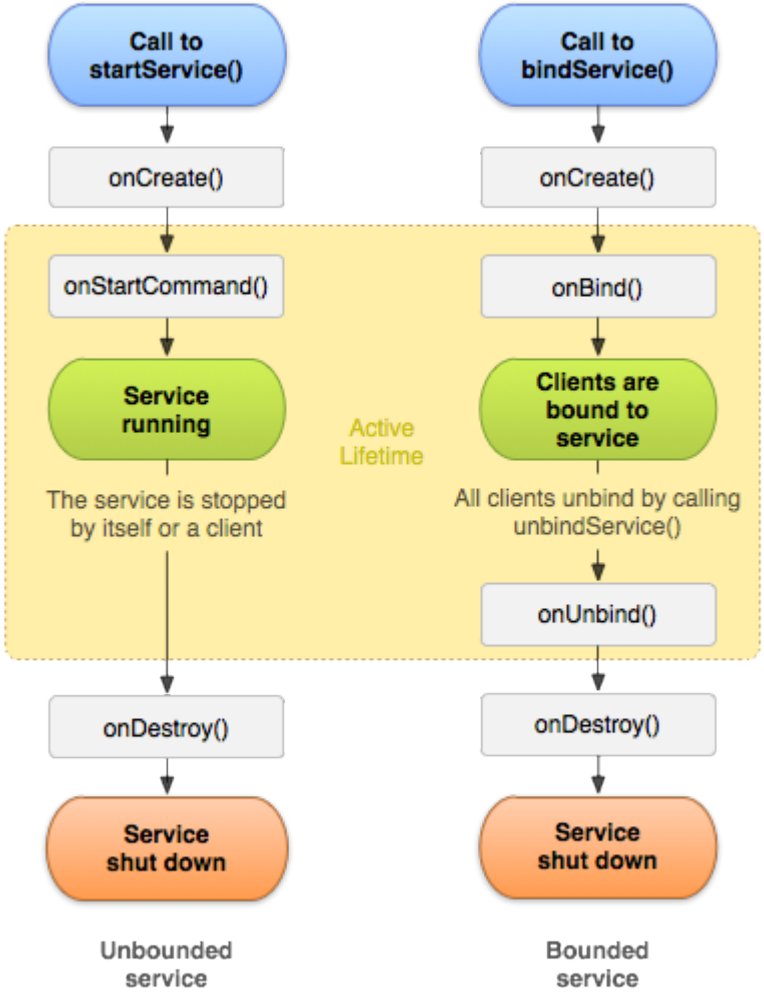
Implementing a Started Service

- Implementing a Service is similar to implementing an Activity
  - e.g., inherit from Android Service class, override lifecycle methods, include Service in the config file AndroidManifest.xml, etc.

```
public class Service extends
             ... {
   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();
   ...
}
```

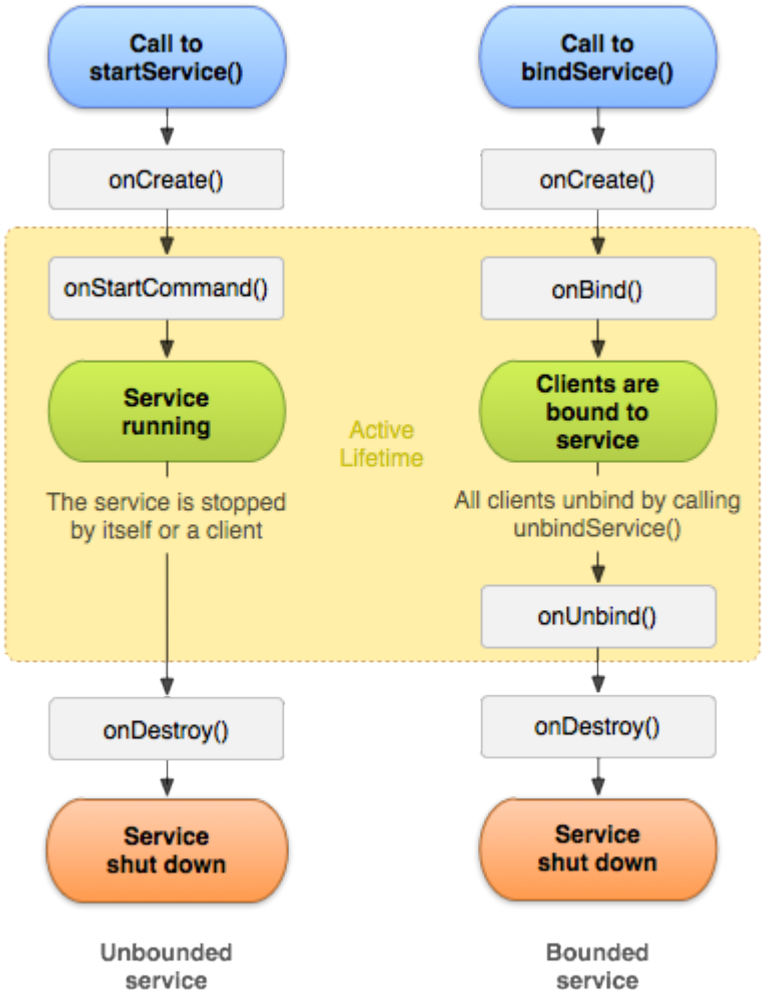
Implementing a Started Service

- Implementing a Service is similar to implementing an Activity
- Android communicates state changes to a Service by calling its lifecycle hook methods



Implementing a Started Service

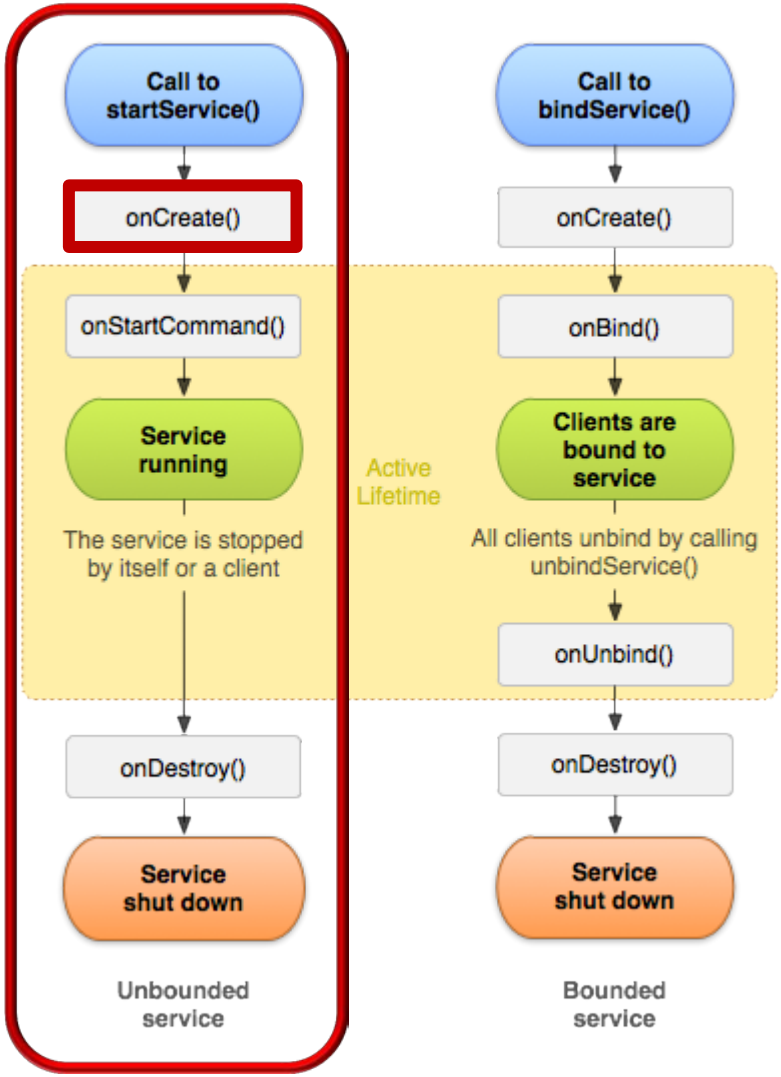
- Implementing a Service is similar to implementing an Activity
- Android communicates state changes to a Service by calling its lifecycle hook methods



- **Commonality**: Provides common interface for performing long-running operations that don't interact directly with the user interface
- **Variability**: Subclasses can override lifecycle hook methods to perform necessary initialization for *Started* & *Bound* Services

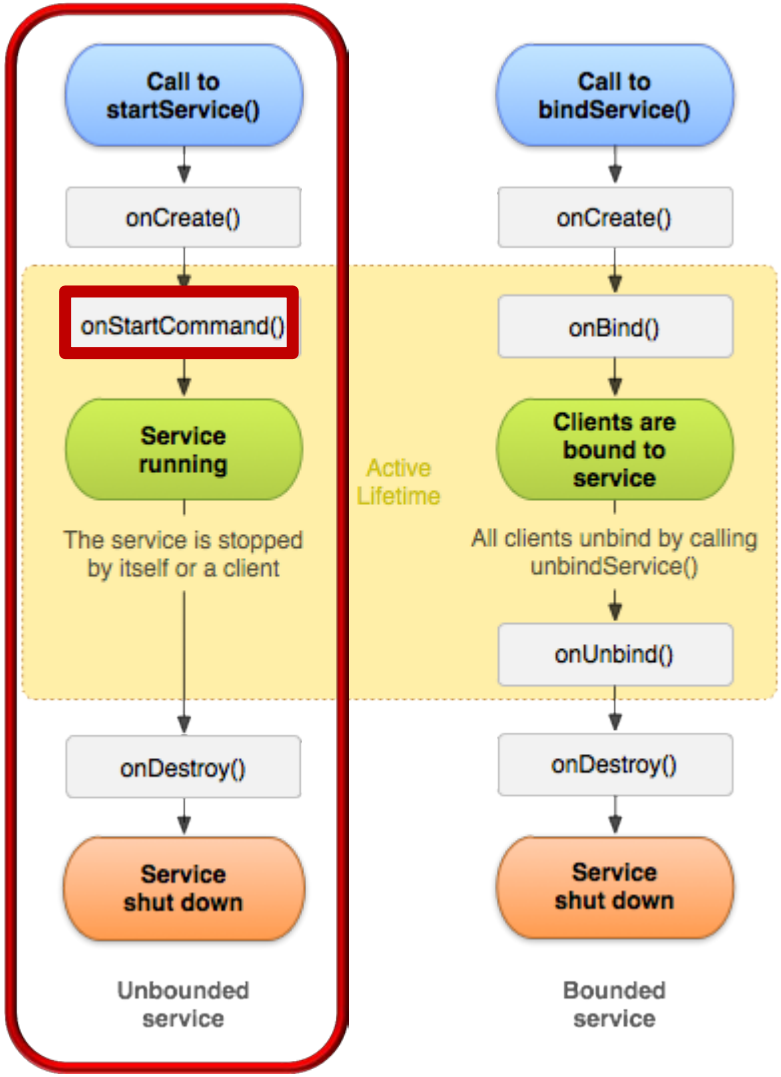
Started Service Lifecycle Hook Methods

- Services lifecycle methods include
  - **onCreate()** – called when Service process is created, by any means



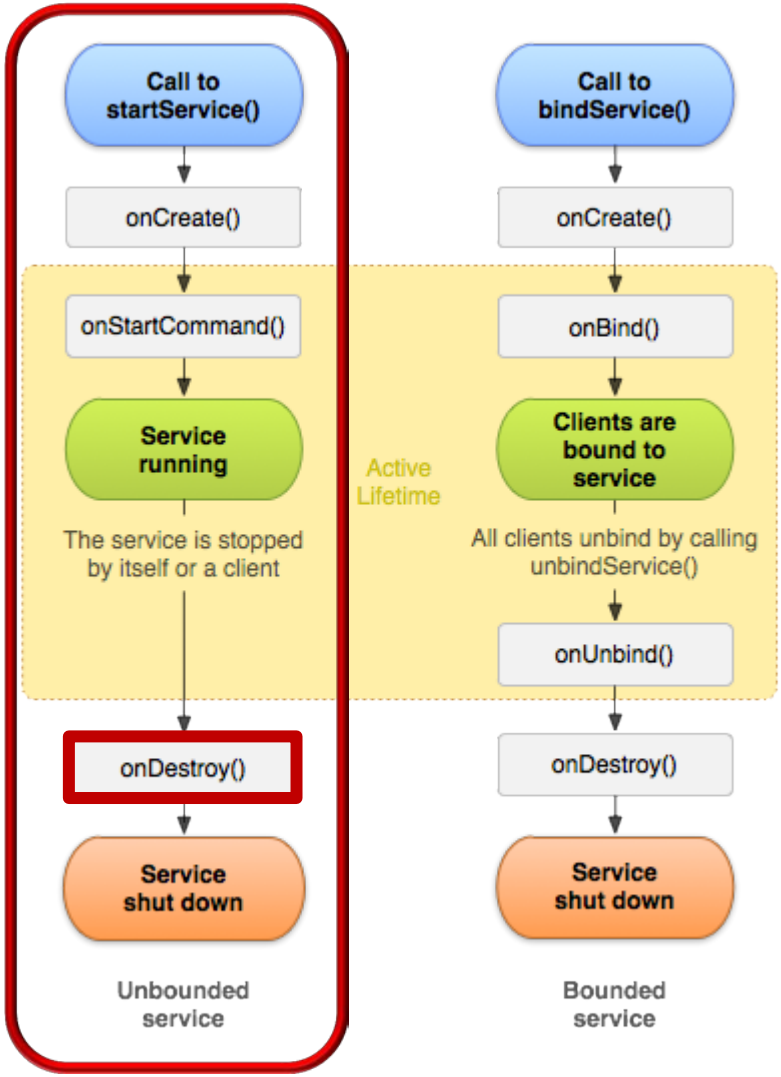
Started Service Lifecycle Hook Methods

- Services lifecycle methods include
  - **onCreate()** – called when Service process is created, by any means
  - **onStartCommand()** – called each time Service is sent a command via startService()



Started Service Lifecycle Hook Methods

- Services lifecycle methods include
  - **onCreate()** – called when Service process is created, by any means
  - **onStartCommand()** – called each time Service is sent a command via startService()
  - **onDestroy()** – called as Service is being shut down to cleanup resources



Programming Started Services

- A Started Service is activated via Context.startService()
  - The Intent identifies the service to communicate with & supplies parameters (via Intent extras) to tell the service what to do

```
Intent intent = new Intent
  (this,
   ThreadedDownloadService.class));
intent.putExtra("URL", imageUrl);
startService(intent);
```

Programming Started Services

- A Started Service is activated via Context.startService()
- startService() does not block
  - If the service is not already running it will be started & will receive the Intent via onStartCommand()
  - Started Services usually perform a single operation & terminate themselves

```
public class DownloadService
             extends Service {
  public int onStartCommand
    (Intent intent, int flags,
     int startId) { ... }
```

Programming Started Services

- A Started Service is activated via Context.startService()
- startService() does not block
- Started Services don't return results to callers, but do return values to Android via onStartCommand():

- *START_STICKY – Don't redeliver Intent to onStartCommand()*
- *START_REDELIVER_INTENT – Restart Service via onStartCommand(), supplying the same Intent as was delivered this time*
- *START_NOT_STICKY – Service should remain stopped until explicitly started by application code*

```
public class DownloadService
             extends Service {
  public int onStartCommand
    (Intent intent, int flags,
     int startId) { return ...; }
```

android-developers.blogspot.com.au/2010/02/service-api-changes-starting-with.html

Programming Started Services

- A Started Service is activated via Context.startService()
- startService() does not block
- Started Services don't return results to callers, but do return values to Android via onStartCommand():
- You need to add a Service to your AndroidManifest. xml file
  - Add a <service> element as a child of the <application> element & provide android:name to reference your Service class

MMS Services

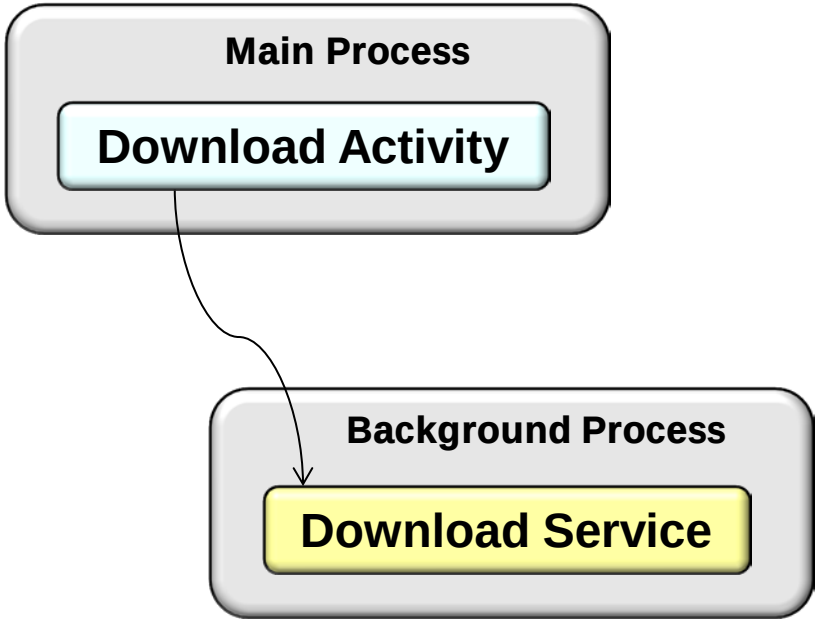
```
<service android:name=
   ".transaction.
     TransactionService"
    android:exported="true"/>

<service android:name=
   ".transaction.
     SmsReceiverService"
     android:exported="true"/>
```

Music Service

```
<service android:name=
   "com.android.music.
    MediaPlaybackService"
     android:exported="false"/>
```

Programming Started Services

- A Started Service is activated via Context.startService()
- startService() does not block
- Started Services don't return results to callers, but do return values to Android via onStartCommand():
- You need to add a Service to your AndroidManifest. xml file
  - Add a <service> element as a child of the <application> element & provide android:name to reference your Service class
  - Use android:process= ":myProcess" to run the service in its own process

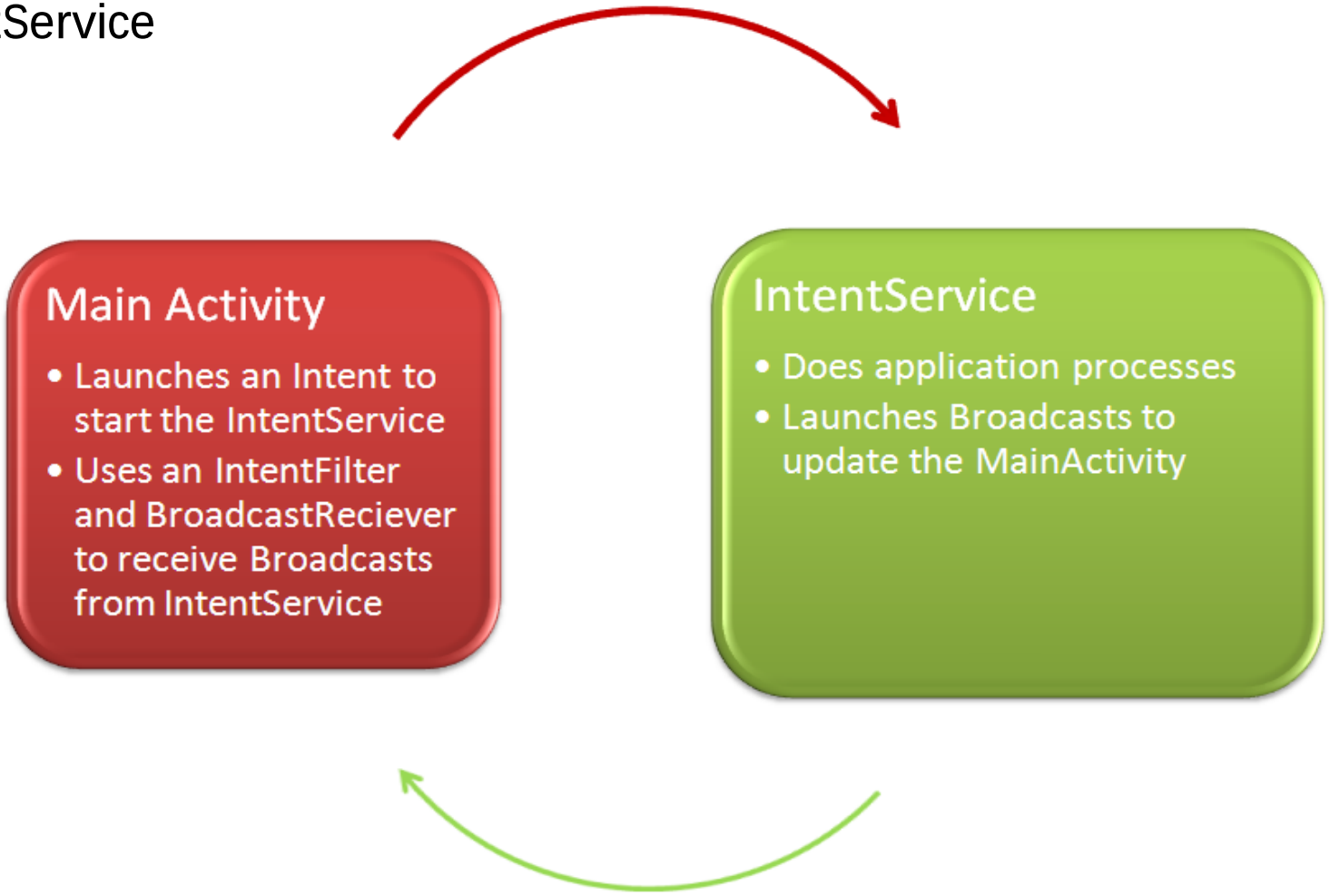


Music Service

```
<service android:name=
   "com.android.music.
    MediaPlaybackService"
     android:process=":myProcess"/>
```

Overview of IntentService

- The most common Service subclass is IntentService



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

Overview of IntentService

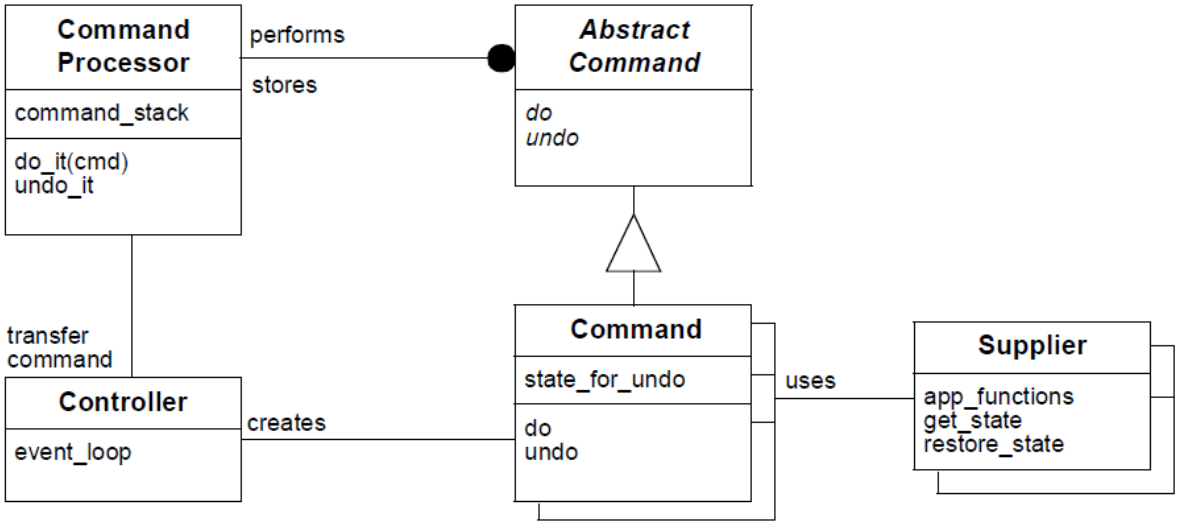
- The most common Service subclass is IntentService
- IntentService is a base class for Services that handle asynchronous requests (expressed as Intents) on demand

```
public class HelloIntentService
       extends IntentService {
  protected void onHandleIntent
    (Intent intent) {
    ...
  }
}
```

- *The IntentService calls this method from the default worker thread with the intent that started the service*
- *When this method returns, IntentService stops the service, as appropriate*

Overview of IntentService

- The most common Service subclass is IntentService
- IntentService is a base class for Services that handle asynchronous requests (expressed as Intents) on demand



- IntentService is commonly used to implement the Command Processor pattern & implements the Activator pattern
  - See www.dre.vanderbilt.edu/~schmidt/CommandProcessor.pdf, www.voelter.de/data/pub/CommandRevisited.pdf, & www.dre.vanderbilt.edu/~schmidt/PDF/ActivatorReloaded.pdf for info on these patterns

Programming an IntentService

- Clients send requests through startService(Intent) calls
  - The service is started as needed, handles each Intent in turn using a worker thread, & stops itself when it runs out of work
  - This "work queue processor" model (aka Command Processor pattern) is commonly used to offload tasks from an application's main thread
  - The IntentService class exists to simplify this pattern & take care of the mechanics
- To program an IntentService, extend the IntentService class & implement the hook method onHandleIntent(Intent)
  - The IntentService will receive the Intents, launch a worker thread, & stop the service as appropriate
- All requests are handled on a single worker thread
  - they may take as long as necessary (& will not block the application's main loop), but only one request will be processed at a time

www.vogella.com/articles/AndroidServices/article.html

IntentService Example

```
public class ThreadedDownloadService extends IntentService {
```

Inherit from IntentService class

```
  public void onHandleIntent(Intent intent) {
    String downloadType = intent.getCharSequenceExtra
                            ("DOWNLOAD_TYPE").toString();
      if (downloadType.equals("messenger"))
        threadMessengerDownload(intent);
      else if (downloadType.equals("pending_intent"))
        threadPendingIntentDownload( intent );
      else if (downloadType.equals("asynctask")
        asyncTaskDownload(intent);
      ...
    }
```

Lifecycle hook method downloads image via various concurrency & IPC mechanisms

Instruct Android to run ThreadedDownloadService in its own process

```
<service android:name="ThreadedDownloadService“
         android:process=":my_process"/>
```

Note the “inversion of control” in the Android Service framework

Service vs. IntentService

- The Service class uses the application's main thread, while IntentService creates a worker thread & uses that thread to run the service
- IntentService creates a queue that passes one intent to onHandleIntent() at a time
  - Implementing a multi-threaded service should therefore often be made by extending Service class directly
- The Service class needs a manual stop using stopSelf()
  - Meanwhile, IntentService automatically stops itself when there is no intent in queue
- IntentService implements onBind() that returns null, which means the IntentService can not be bound by default
- IntentService implements onStartCommand() that places the Intent on its work queue & calls onHandleIntent()

Service vs. Thread vs. AsyncTask

- A Service is not a separate process
  - The Service object itself does not imply it is running in its own process
  - Unless otherwise specified, it runs in the same process as the application it is part of
  - It keeps running until stopped by itself, stopped by the user or killed by the system if it needs memory
- A Service is not a thread
  - It is not a means itself to do work off of the main thread (to avoid Application Not Responding errors)
- Threads or AsyncTask perform their work in a background thread, so they don't block the main thread
- Since a Service performs its work in the main thread it might block that thread until it finishes when performing an intensive task
  - such as calling a web service
- For intensive tasks a service should run it's work in a background thread