Android Services & Local IPC: Programming Bound Services with Messengers (Part 1)

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

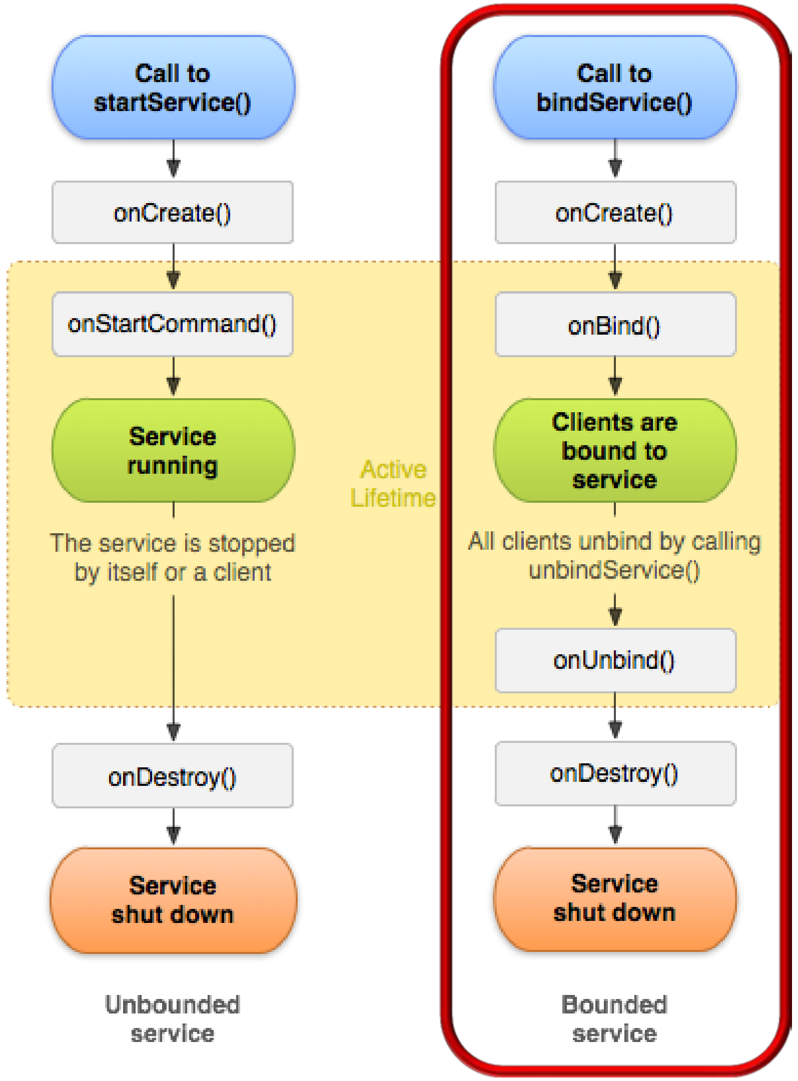
Professor of Computer Science

Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Module

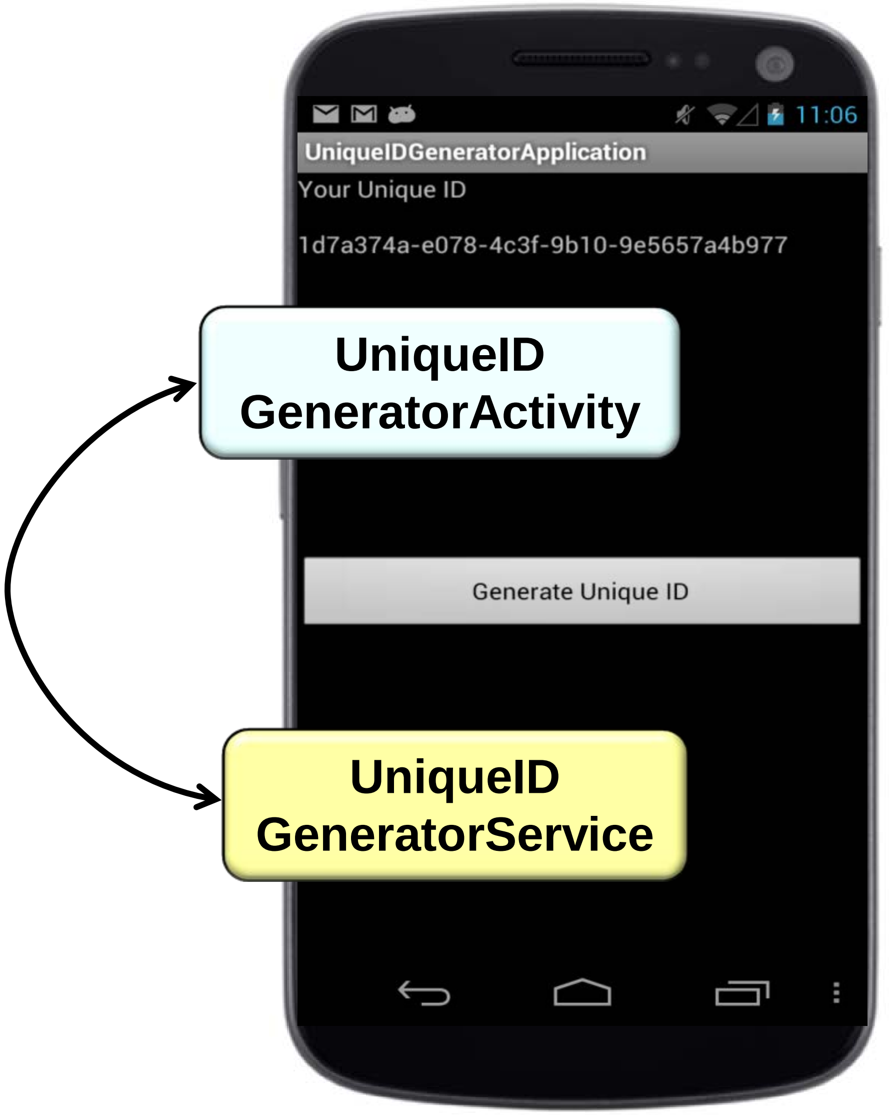
- Recognize how a Bound Service provides a Client-Server interface that allows two-way conversations between one or more Clients & the Service



See earlier part on "Overview of Android Services"

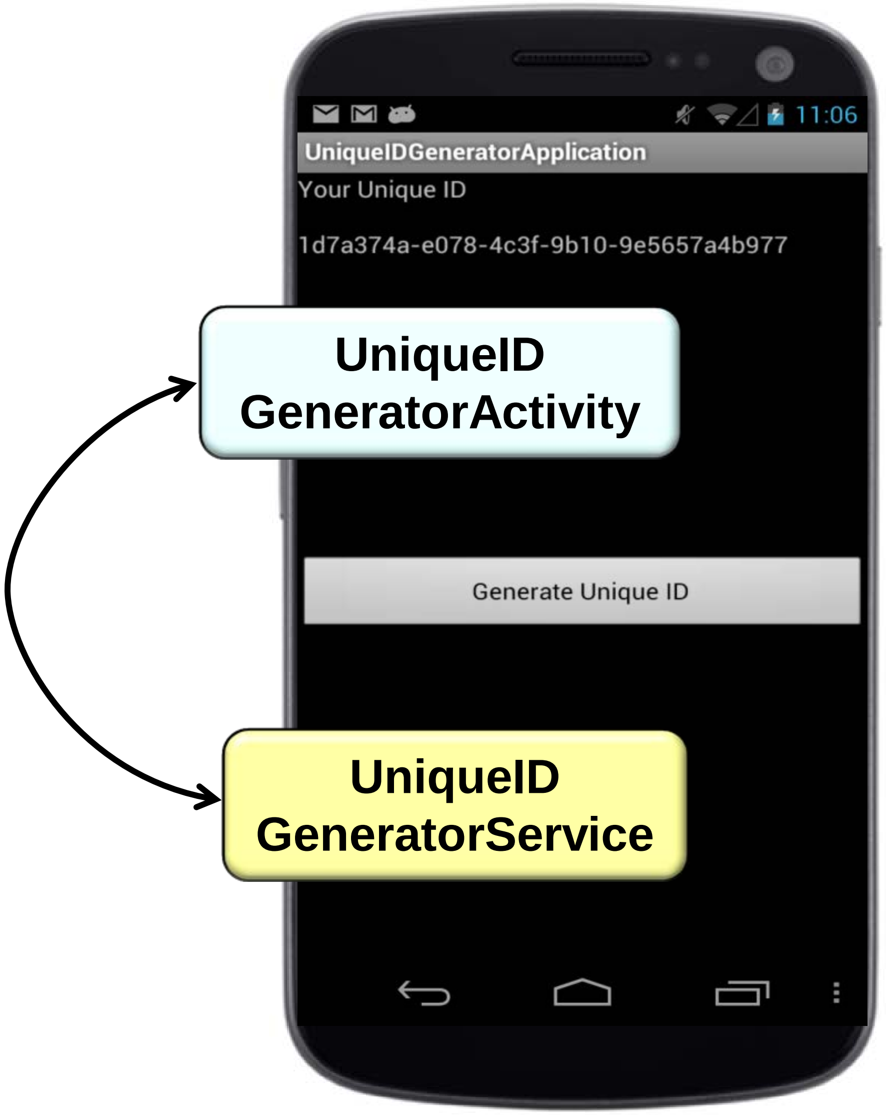
Learning Objectives in this Part of the Module

- Recognize how a Bound Service provides a Client-Server interface that allows two-way conversations between one or more Clients & the Service
- Understand how to develop a Bound Service with Messengers


11:06
UniqueIDGeneratorApplication
Your Unique ID
1d7a374a-e078-4c3f-9b10-9e5657a4b977
UniqueID GeneratorActivity
Generate Unique ID
UniqueID GeneratorService

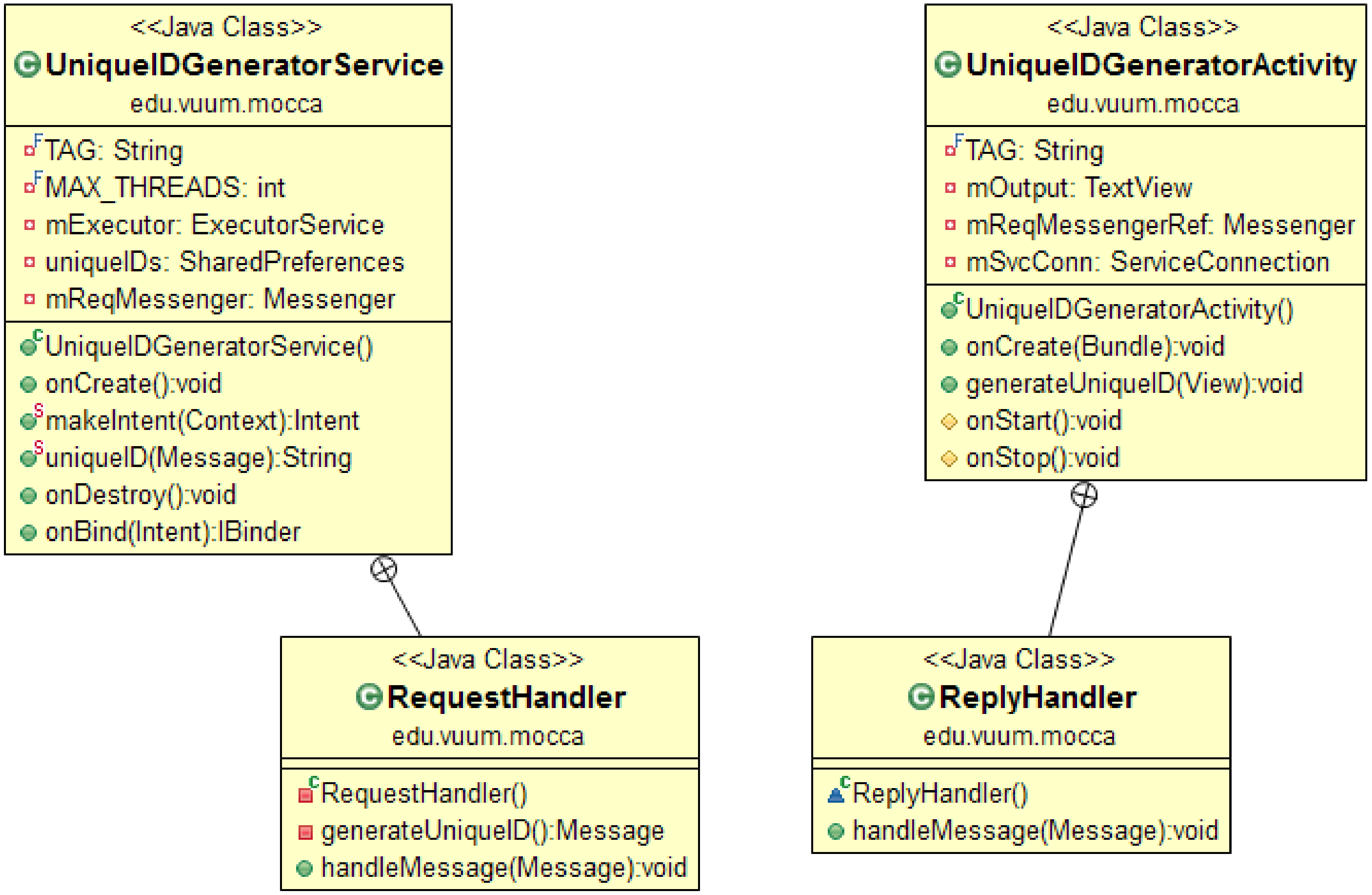
Learning Objectives in this Part of the Module

- Recognize how a Bound Service provides a Client-Server interface that allows two-way conversations between one or more Clients & the Service
- Understand how to develop a Bound Service with Messengers
  - e.g., a Unique ID generator application that uses a pair of Messengers to concurrently retrieve a system-wide persistent unique ID


11:06
UniqueIDGeneratorApplication
Your Unique ID
1d7a374a-e078-4c3f-9b10-9e5657a4b977
UniqueID GeneratorActivity
Generate Unique ID
UniqueID GeneratorService

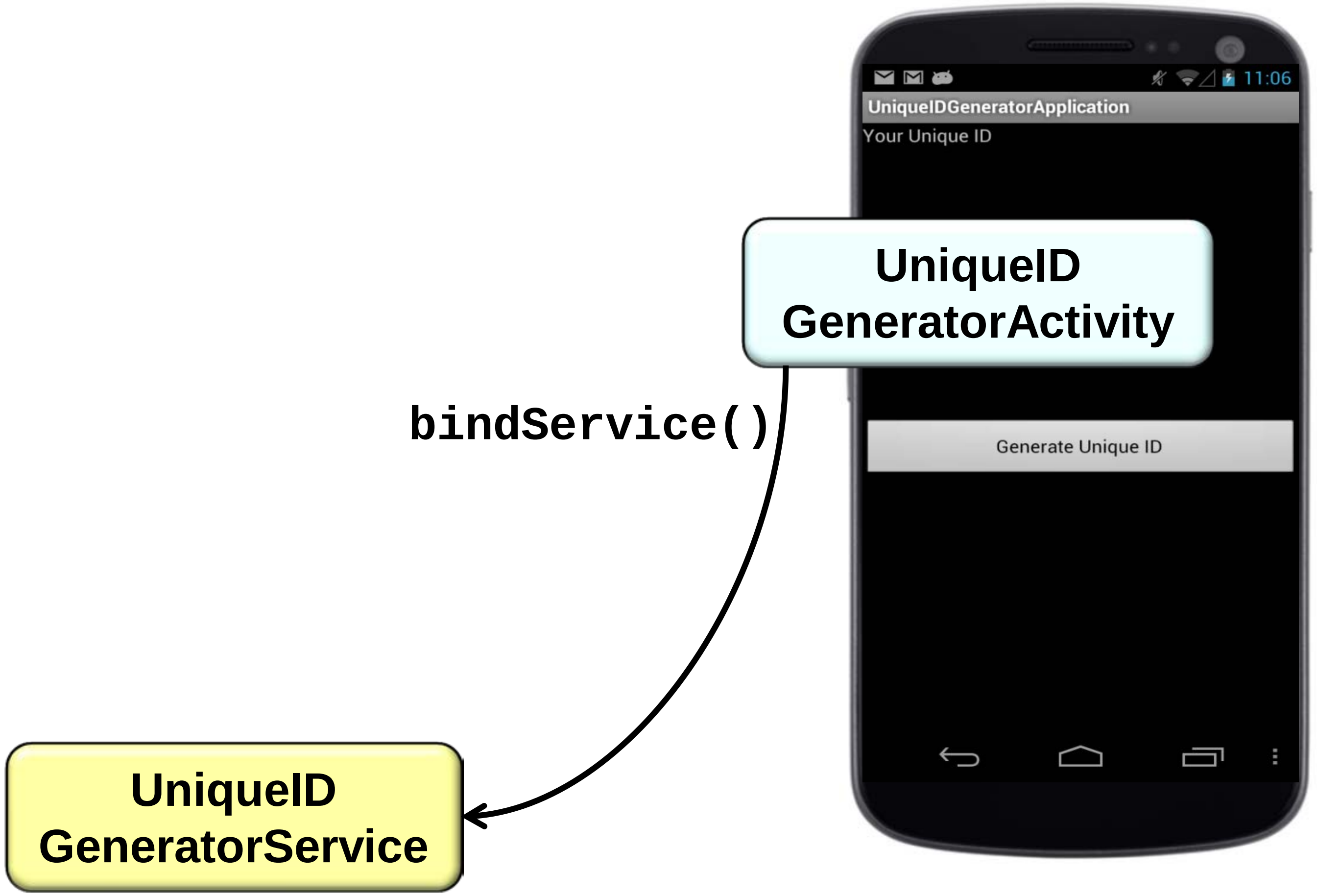
class.coursera.org/android-001/lecture/85 shows an AIDL-based version of this application

UniqueIDGeneratorApplication Overview



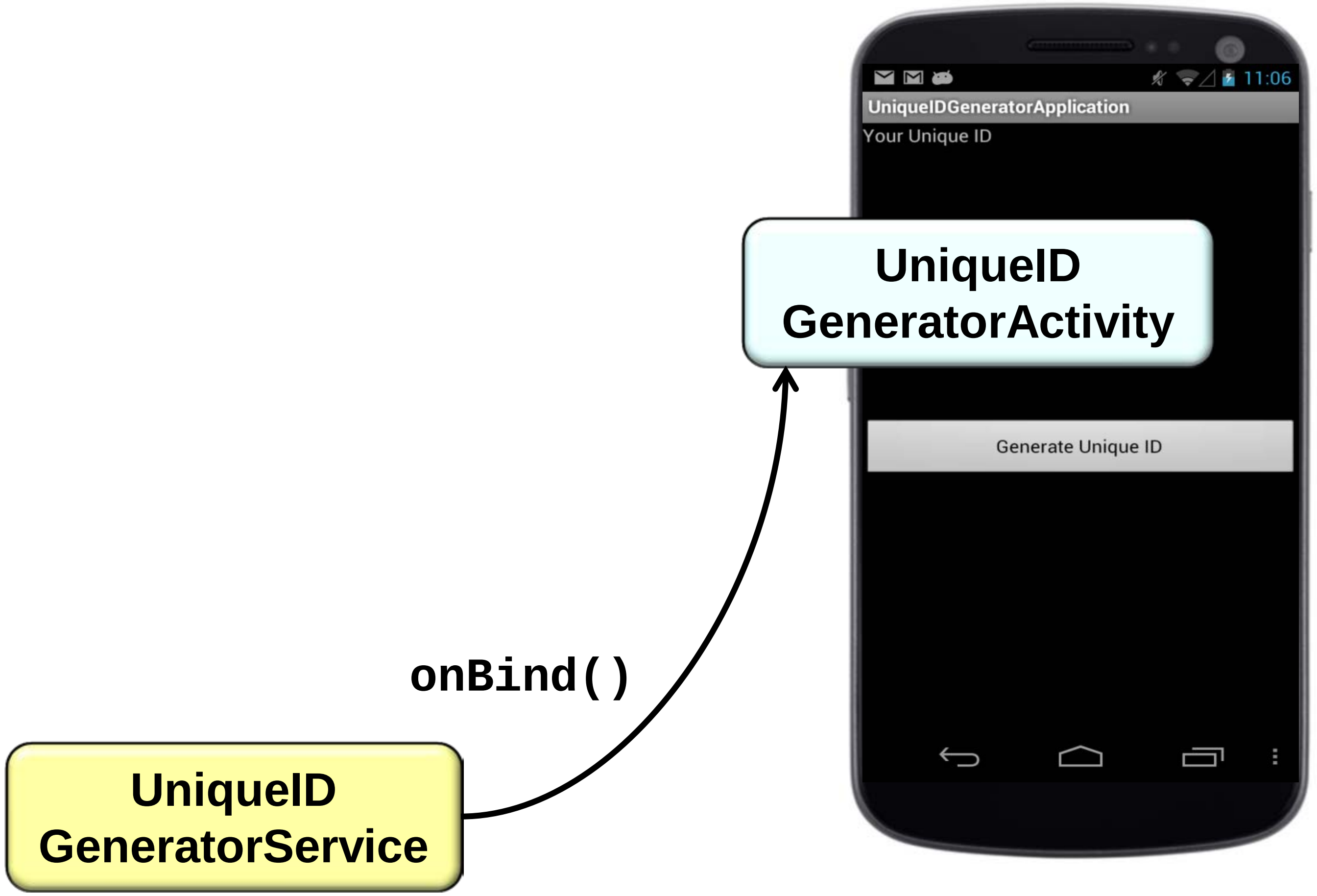
UniqueIDGeneratorApplication Overview

1. UniqueIDGeneratorActivity calls bindService() to launch the UniqueIDGeneratorService when its onStart() hook method is called


11:06
UniqueIDGeneratorApplication
Your Unique ID
UniqueID GeneratorActivity
bindService()
Generate Unique ID
UniqueID GeneratorService

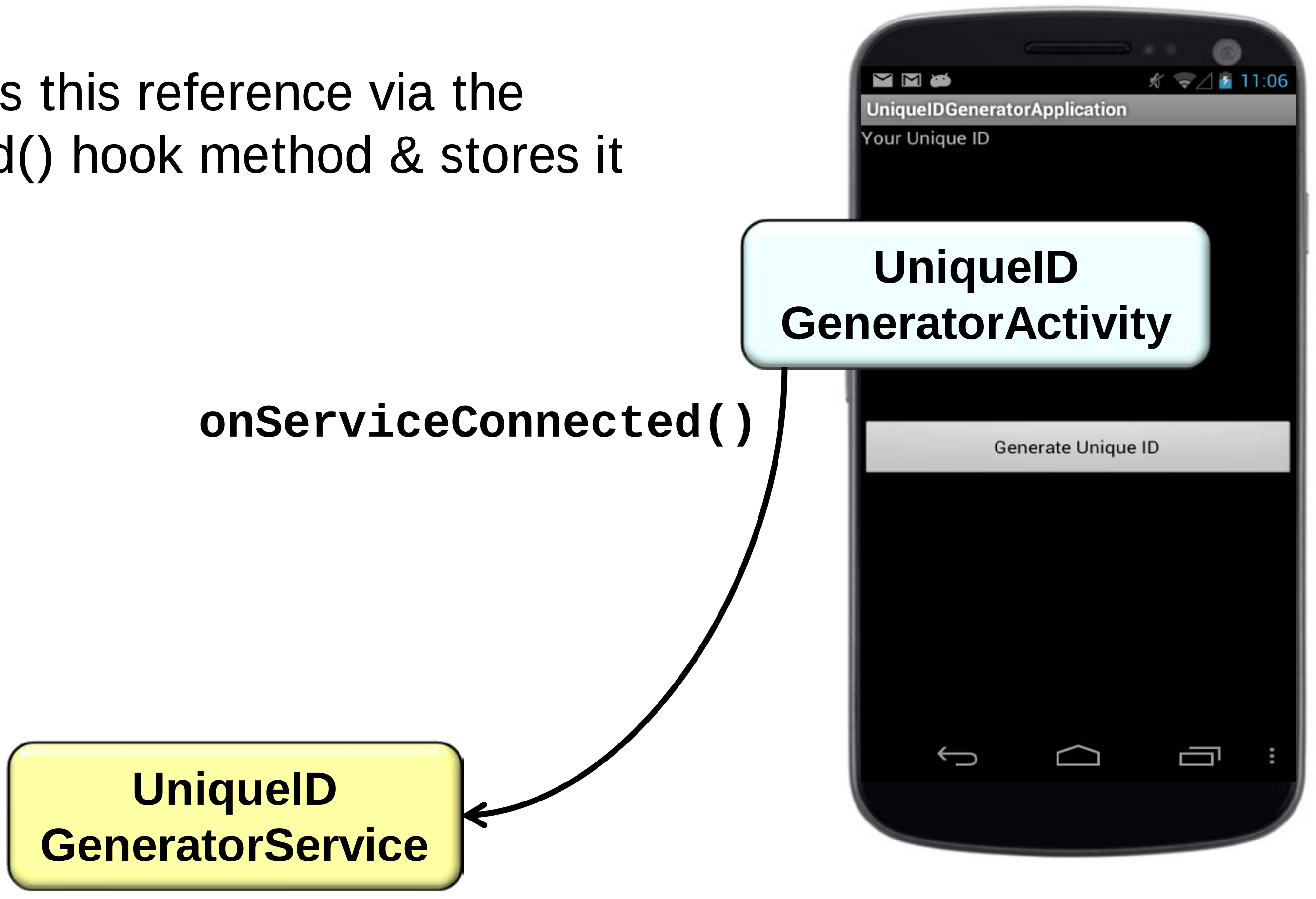
UniqueIDGeneratorApplication Overview

1. UniqueIDGeneratorActivity calls bindService() to launch the UniqueIDGeneratorService when its onStart() hook method is called
2. This Service returns a reference to a Messenger via the onBind() hook method


11:06
UniqueIDGeneratorApplication
Your Unique ID
UniqueID GeneratorActivity
Generate Unique ID
onBind()
UniqueID GeneratorService

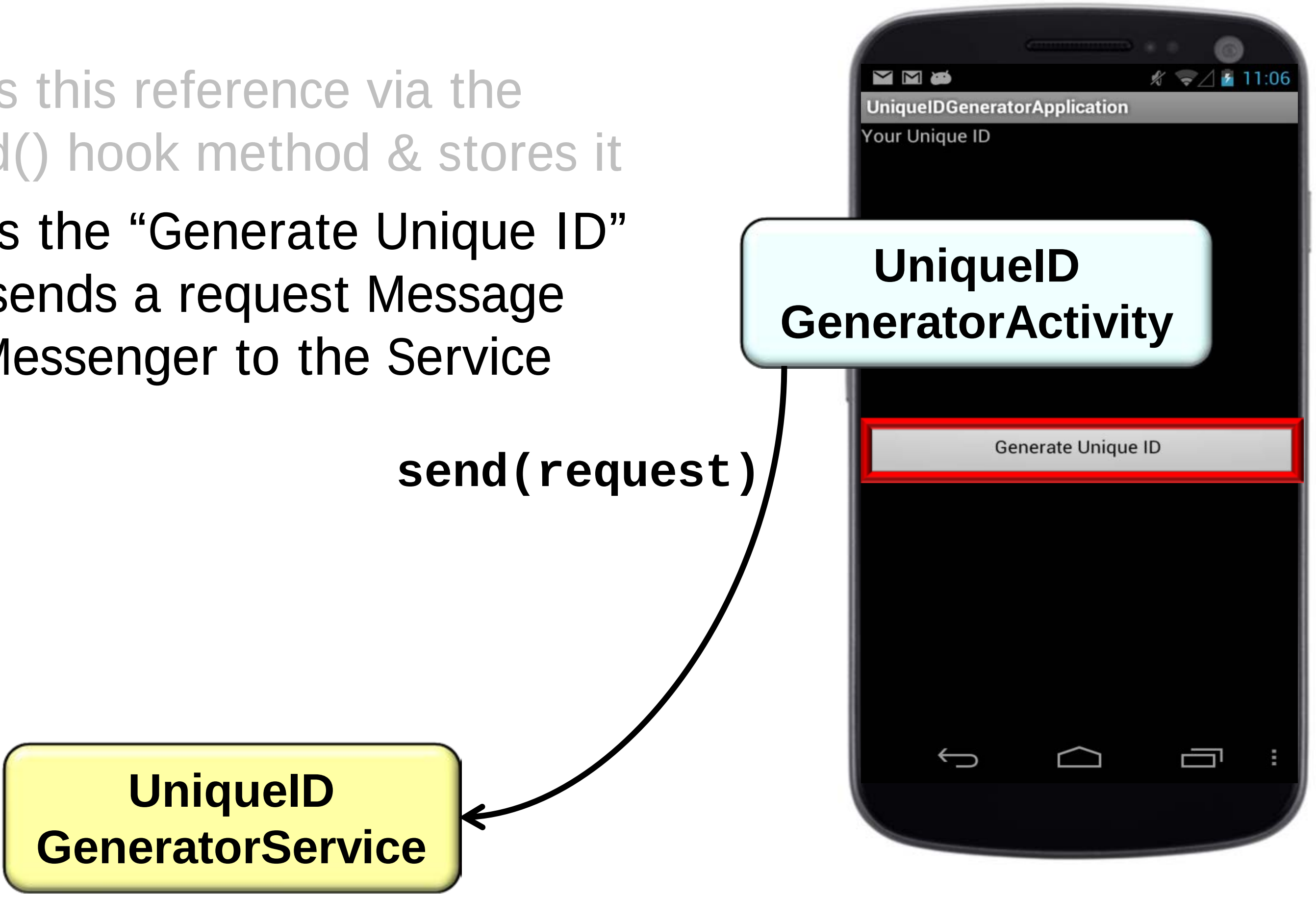
UniqueIDGeneratorApplication Overview

1. UniqueIDGeneratorActivity calls bindService() to launch the UniqueIDGeneratorService when its onStart() hook method is called
2. This Service returns a reference to a Messenger via the onBind() hook method
3. The Activity receives this reference via the onServiceConnected() hook method & stores it


11:06
UniqueIDGeneratorApplication
Your Unique ID
UniqueID GeneratorActivity
onServiceConnected()
Generate Unique ID
UniqueID GeneratorService

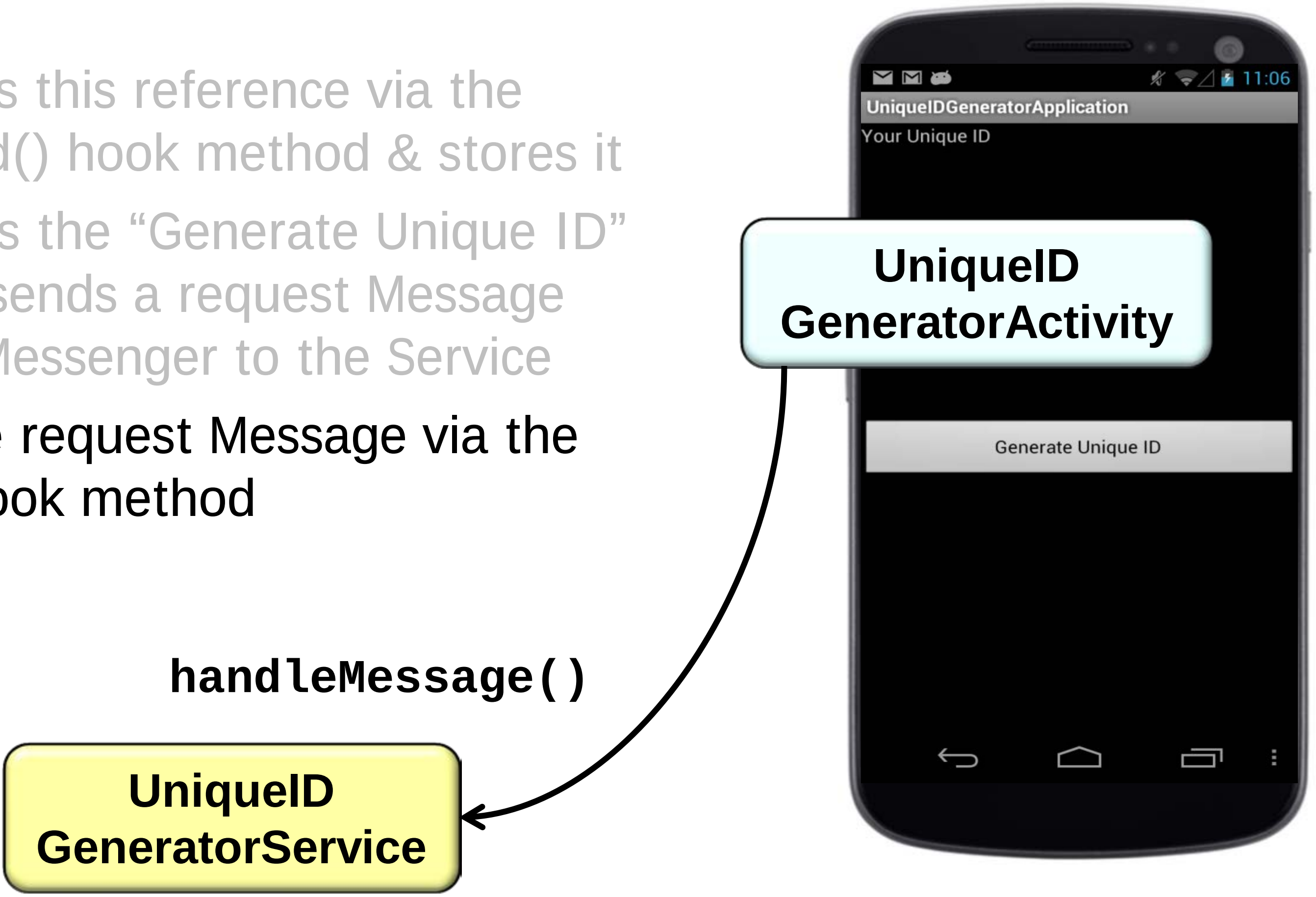
UniqueIDGeneratorApplication Overview

1. UniqueIDGeneratorActivity calls bindService() to launch the UniqueIDGeneratorService when its onStart() hook method is called
2. This Service returns a reference to a Messenger via the onBind() hook method
3. The Activity receives this reference via the onServiceConnected() hook method & stores it
4. When the user clicks the “Generate Unique ID” button the Activity sends a request Message containing a reply Messenger to the Service


11:06
UniqueIDGeneratorApplication
Your Unique ID
UniqueID GeneratorActivity
Generate Unique ID
send(request)
UniqueID GeneratorService

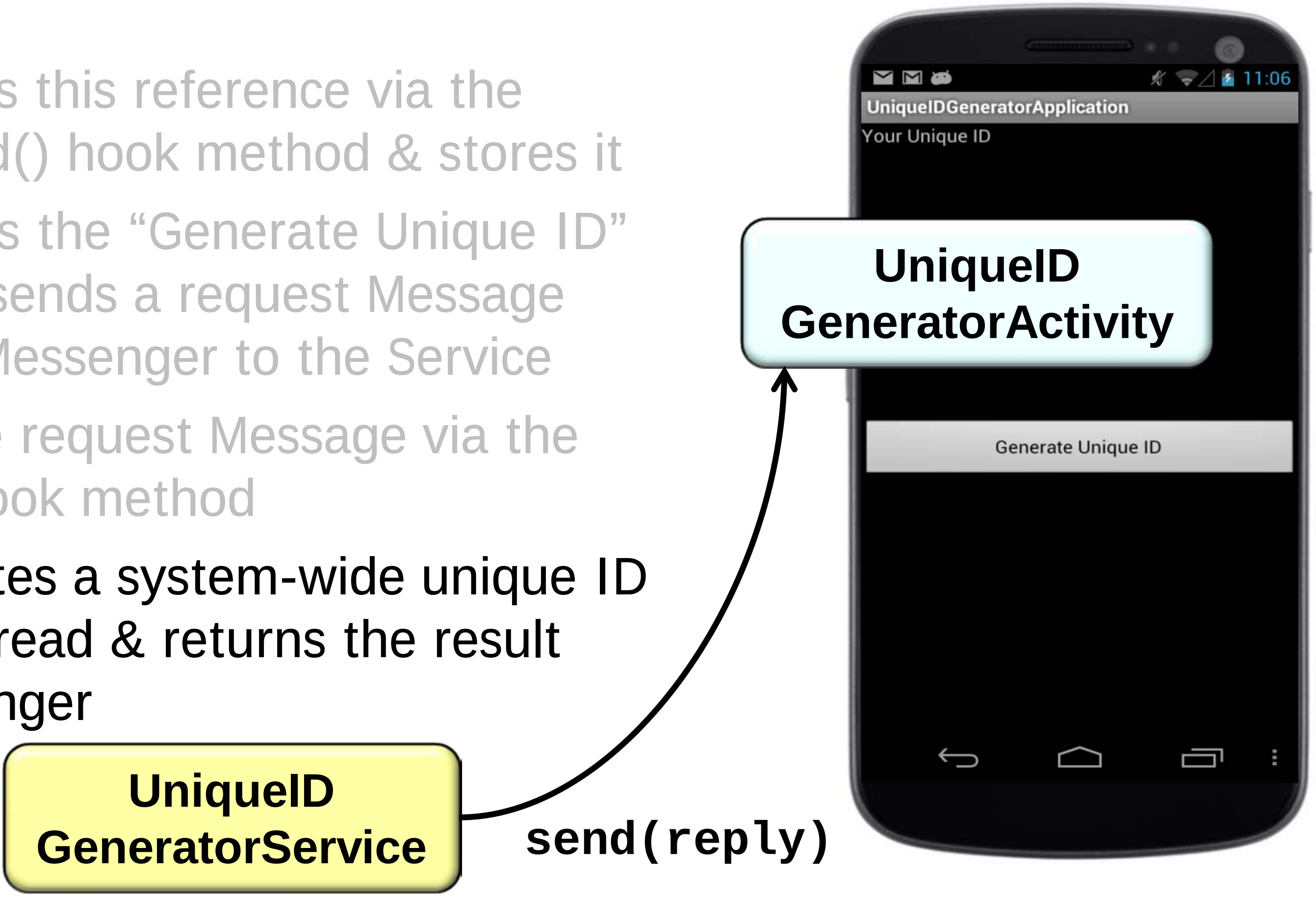
UniqueIDGeneratorApplication Overview

1. UniqueIDGeneratorActivity calls bindService() to launch the UniqueIDGeneratorService when its onStart() hook method is called
2. This Service returns a reference to a Messenger via the onBind() hook method
3. The Activity receives this reference via the onServiceConnected() hook method & stores it
4. When the user clicks the "Generate Unique ID" button the Activity sends a request Message containing a reply Messenger to the Service
5. Service receives the request Message via the handleMessage() hook method


11:06
UniqueIDGeneratorApplication
Your Unique ID
UniqueID GeneratorActivity
Generate Unique ID
handleMessage()
UniqueID GeneratorService

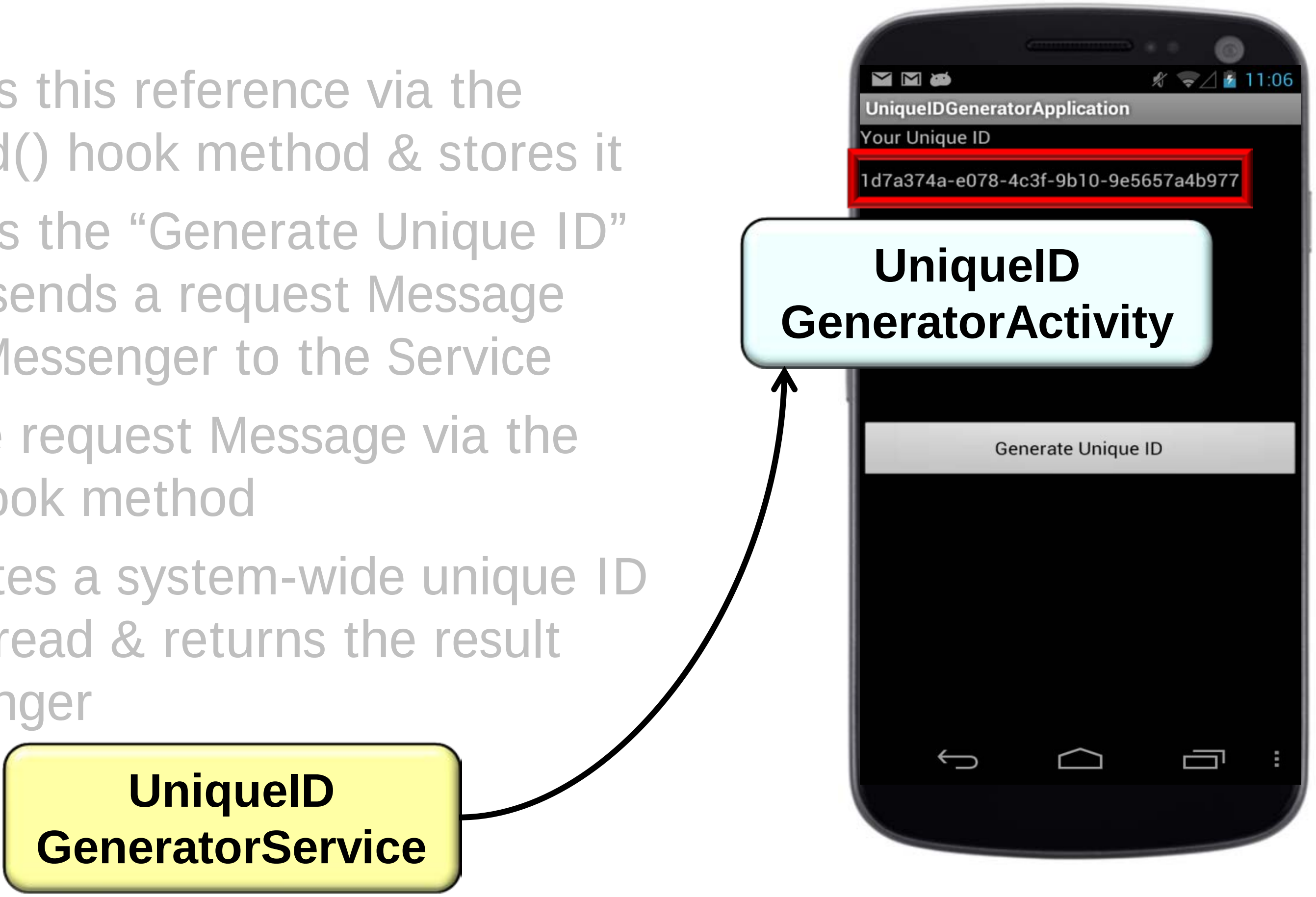
UniqueIDGeneratorApplication Overview

1. UniqueIDGeneratorActivity calls bindService() to launch the UniqueIDGeneratorService when its onStart() hook method is called
2. This Service returns a reference to a Messenger via the onBind() hook method
3. The Activity receives this reference via the onServiceConnected() hook method & stores it
4. When the user clicks the "Generate Unique ID" button the Activity sends a request Message containing a reply Messenger to the Service
5. Service receives the request Message via the handleMessage() hook method
6. The Service generates a system-wide unique ID in a background Thread & returns the result via the reply Messenger


11:06
UniqueIDGeneratorApplication
Your Unique ID
UniqueID GeneratorActivity
Generate Unique ID
UniqueID GeneratorService
send(reply)

UniqueIDGeneratorApplication Overview

1. UniqueIDGeneratorActivity calls bindService() to launch the UniqueIDGeneratorService when its onStart() hook method is called
2. This Service returns a reference to a Messenger via the onBind() hook method
3. The Activity receives this reference via the onServiceConnected() hook method & stores it
4. When the user clicks the "Generate Unique ID" button the Activity sends a request Message containing a reply Messenger to the Service
5. Service receives the request Message via the handleMessage() hook method
6. The Service generates a system-wide unique ID in a background Thread & returns the result via the reply Messenger
7. Activity displays the unique ID


11:06
UniqueIDGeneratorApplication
Your Unique ID
1d7a374a-e078-4c3f-9b10-9e5657a4b977
UniqueID GeneratorActivity
Generate Unique ID
UniqueID GeneratorService

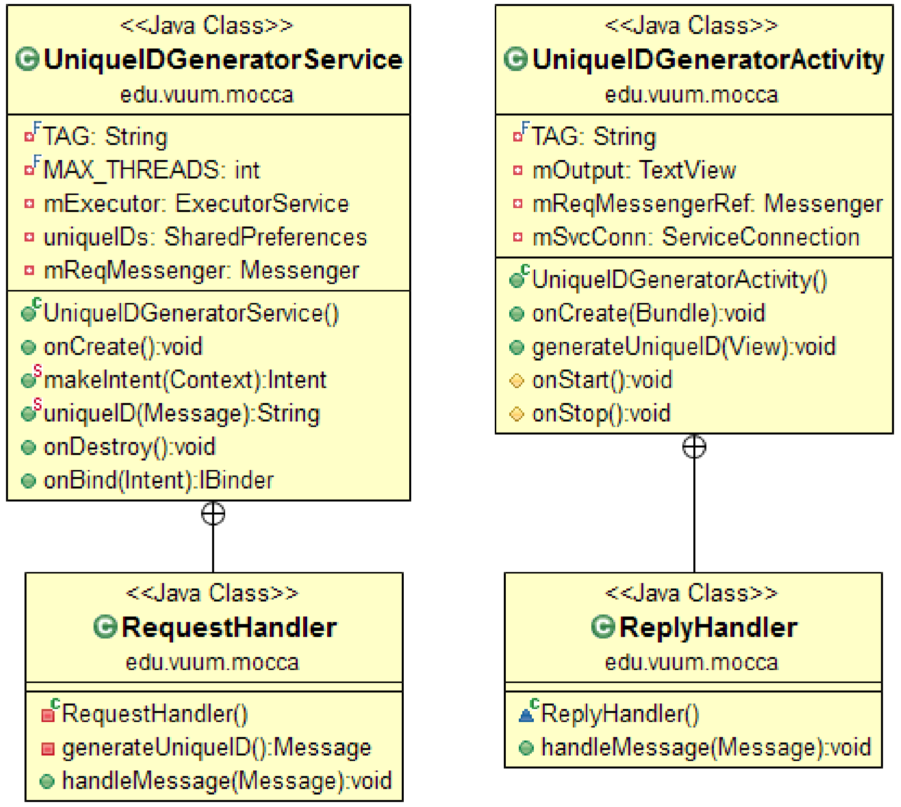
Tips to Understand UniqueIDGenerator Application

- Run/read the code & watch the video carefully to understand how it works

github.com/douglascraigschmidt/POSA-14/tree/master/ex/UniqueIDGeneratorApplication

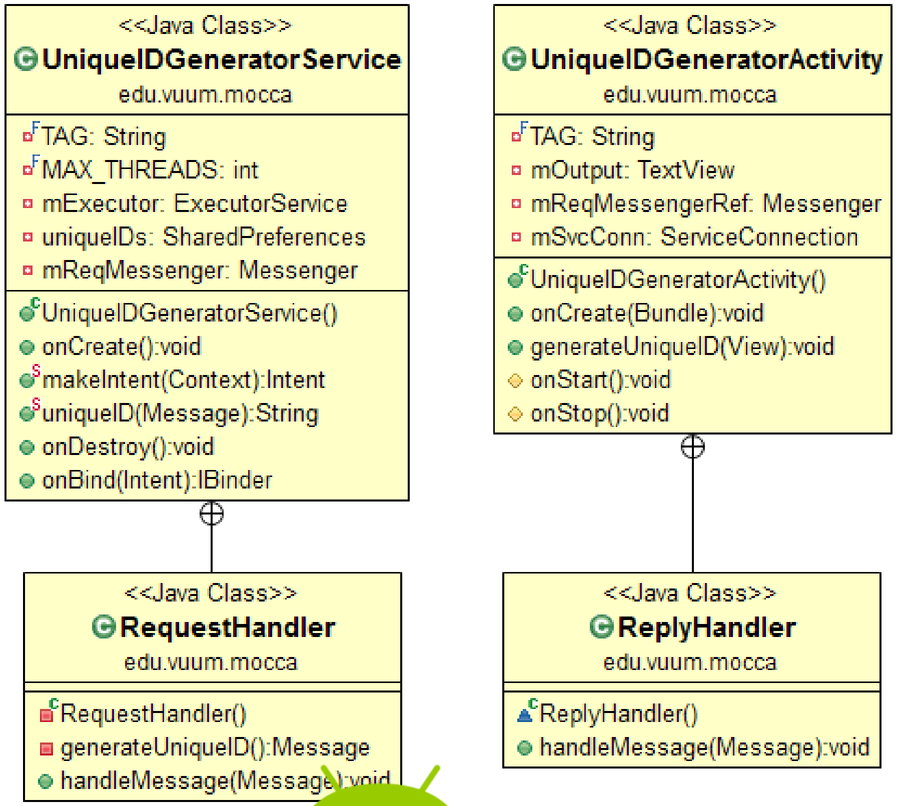
Tips to Understand UniqueIDGenerator Application

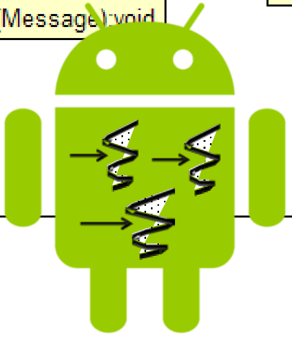
- Run/read the code & watch the video carefully to understand how it works
- This example is complex since many classes & Android mechanisms are used

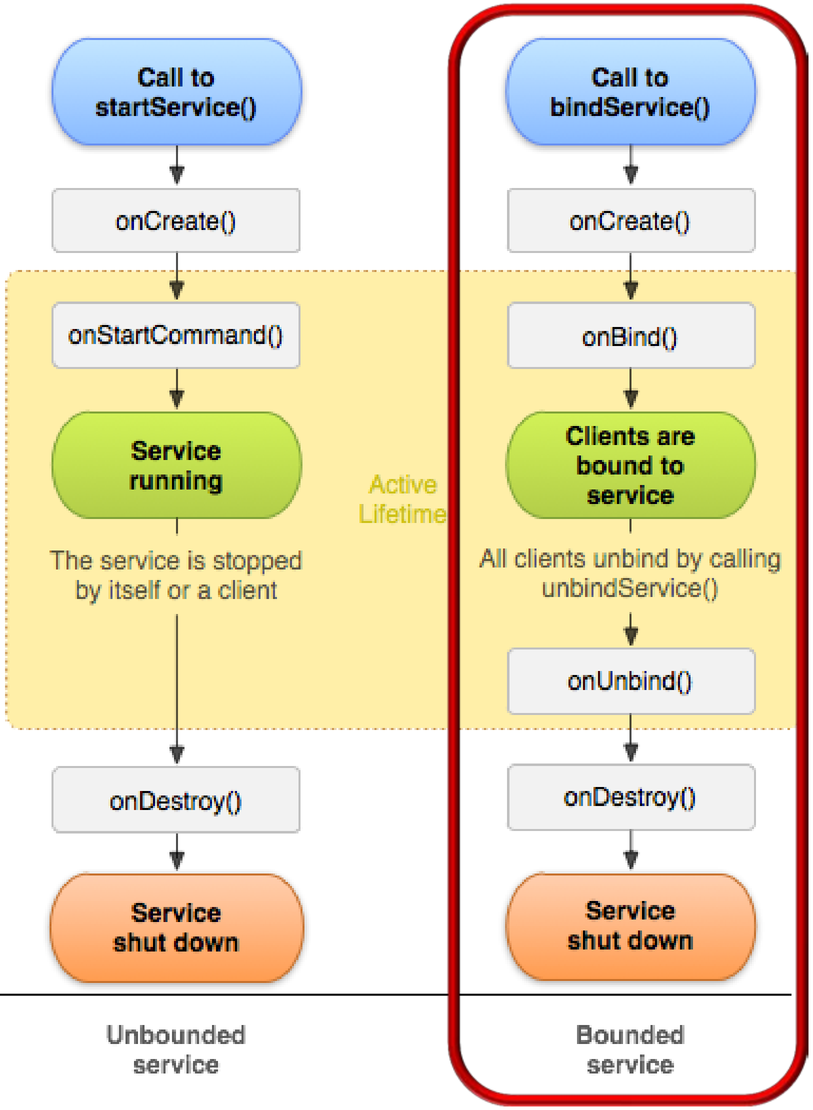


Tips to Understand UniqueIDGenerator Application

- Run/read the code & watch the video carefully to understand how it works
- This example is complex since many classes & Android mechanisms are used





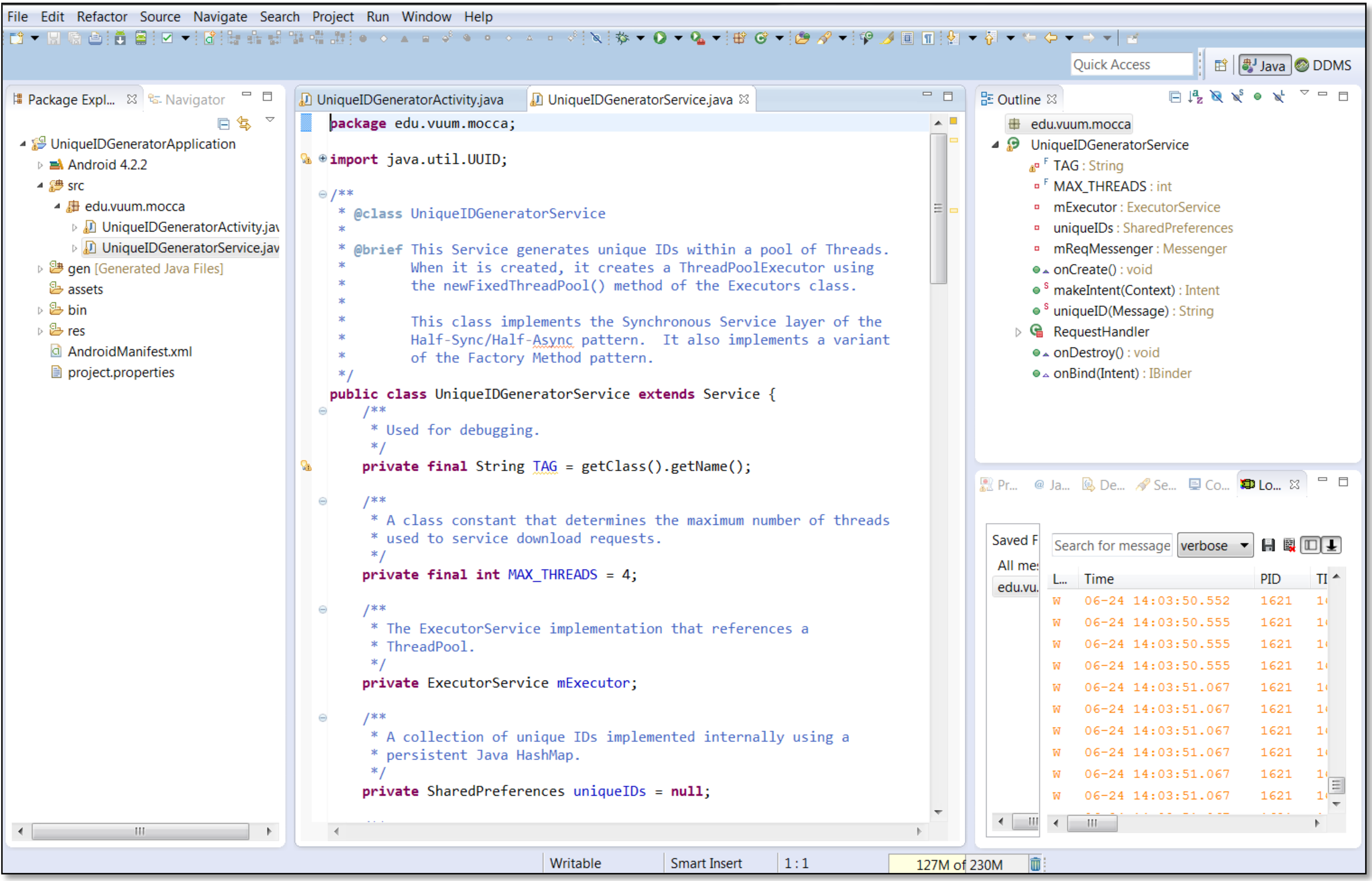


Tips to Understand UniqueIDGenerator Application

- Run/read the code & watch the video carefully to understand how it works
- This example is complex since many classes & Android mechanisms are used
  - We therefore analyze it from various perspectives

Launching & Initializing Bound Services

Programming the UniqueIDGenerator Application


File Edit Refactor Source Navigate Search Project Run Window Help
Quick Access
Java
DDMS
Package Expl...
Navigator
UniqueIDGeneratorApplication
Android 4.2.2
src
edu.vuum.mocca
UniqueIDGeneratorActivity.jav
UniqueIDGeneratorService.jav
gen [Generated Java Files]
assets
bin
res
AndroidManifest.xml
project.properties
UniqueIDGeneratorActivity.java
UniqueIDGeneratorService.java
Outline
Pr... Ja... De... Se... Co... Lo...
Writable
Smart Insert
1 : 1
127M of 230M

```java
package edu.vuum.mocca;

import java.util.UUID;

/**
 * @class UniqueIDGeneratorService
 *
 * @brief This Service generates unique IDs within a pool of Threads.
 *        When it is created, it creates a ThreadPoolExecutor using
 *        the newFixedThreadPool() method of the Executors class.
 *
 *        This class implements the Synchronous Service layer of the
 *        Half-Sync/Half-Async pattern.  It also implements a variant
 *        of the Factory Method pattern.
 */
public class UniqueIDGeneratorService extends Service {
    /**
     * Used for debugging.
     */
    private final String TAG = getClass().getName();

    /**
     * A class constant that determines the maximum number of threads
     * used to service download requests.
     */
    private final int MAX_THREADS = 4;

    /**
     * The ExecutorService implementation that references a
     * ThreadPool.
     */
    private ExecutorService mExecutor;

    /**
     * A collection of unique IDs implemented internally using a
     * persistent Java HashMap.
     */
    private SharedPreferences uniqueIDs = null;
```

Outline:

```
edu.vuum.mocca
UniqueIDGeneratorService
    TAG : String
    MAX_THREADS : int
    mExecutor : ExecutorService
    uniqueIDs : SharedPreferences
    mReqMessenger : Messenger
    onCreate() : void
    makeIntent(Context) : Intent
    uniqueID(Message) : String
    RequestHandler
    onDestroy() : void
    onBind(Intent) : IBinder
```

LogCat:

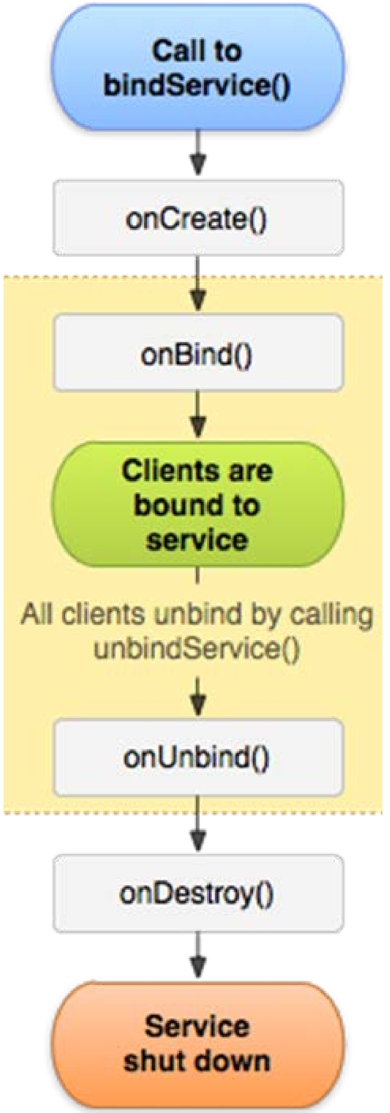
```
Saved F    Search for message    verbose
All me:
edu.vu.    L...  Time                     PID    TI
           W     06-24 14:03:50.552       1621   1
           W     06-24 14:03:50.555       1621   1
           W     06-24 14:03:50.555       1621   1
           W     06-24 14:03:50.555       1621   1
           W     06-24 14:03:51.067       1621   1
           W     06-24 14:03:51.067       1621   1
           W     06-24 14:03:51.067       1621   1
           W     06-24 14:03:51.067       1621   1
           W     06-24 14:03:51.067       1621   1
           W     06-24 14:03:51.067       1621   1
```

Launching a Bound Service

- A Bound Service allows components to bind to it by calling bindService() to create a "persistent" connection

```
Intent intent =
  UniqueIDGeneratorService.makeIntent(this);
bindService(intent, mSvcConn,
            Context.BIND_AUTO_CREATE);
```

UniqueID Generator Activity



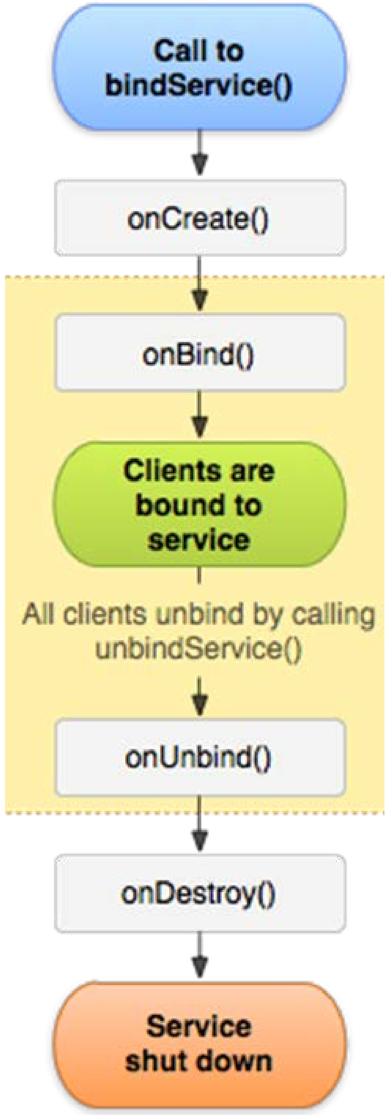
developer.android.com/guide/components/services.html#CreatingBoundService

Launching a Bound Service

- A Bound Service allows components to bind to it by calling bindService() to create a “persistent” connection

```
Intent intent =
  UniqueIDGeneratorService.makeIntent(this);
bindService(intent, mSvcConn,
            Context.BIND_AUTO_CREATE);
```

UniqueID Generator Activity

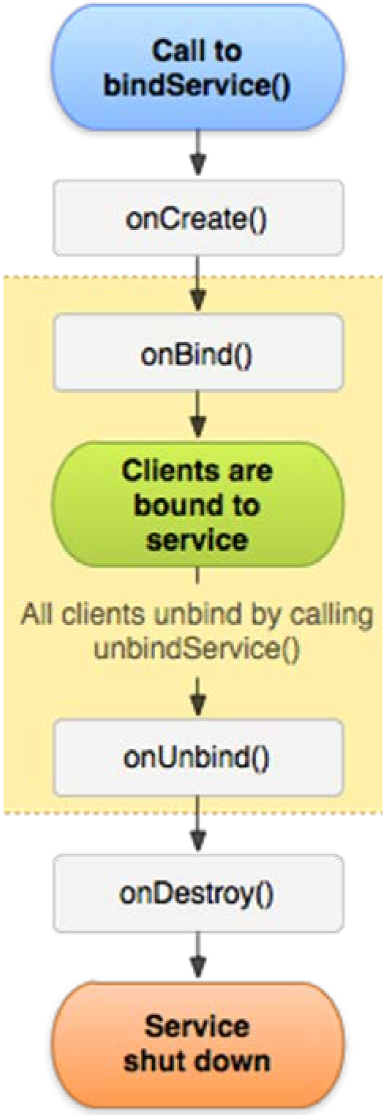


Launching a Bound Service

- A Bound Service allows components to bind to it by calling bindService() to create a "persistent" connection
  - The client must provide ServiceConnection object to monitor the connection with the Service

```
Intent intent =
  UniqueIDGeneratorService.makeIntent(this);
bindService(intent, mSvcConn,
            Context.BIND_AUTO_CREATE);
```

UniqueID Generator Activity

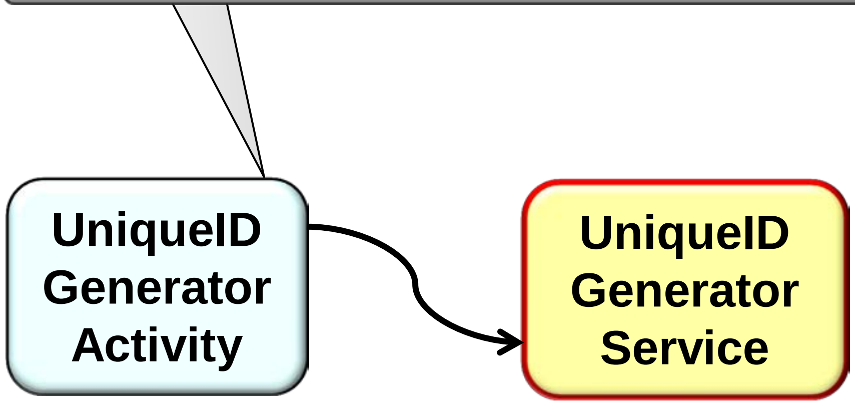


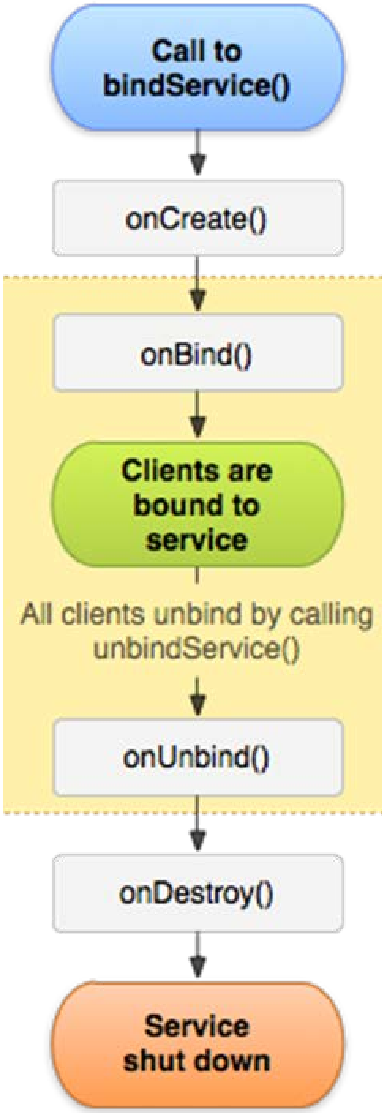
developer.android.com/reference/android/content/ServiceConnection.html

Launching a Bound Service

- A Bound Service allows components to bind to it by calling bindService() to create a "persistent" connection
  - The client must provide ServiceConnection object to monitor the connection with the Service
  - If the Service isn't already running it will be activated

```
Intent intent =
  UniqueIDGeneratorService.makeIntent(this);
bindService(intent, mSvcConn,
            Context.BIND_AUTO_CREATE);
```



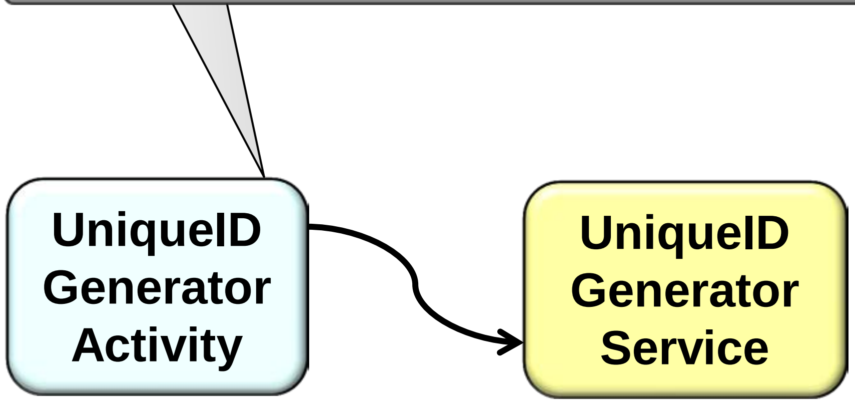


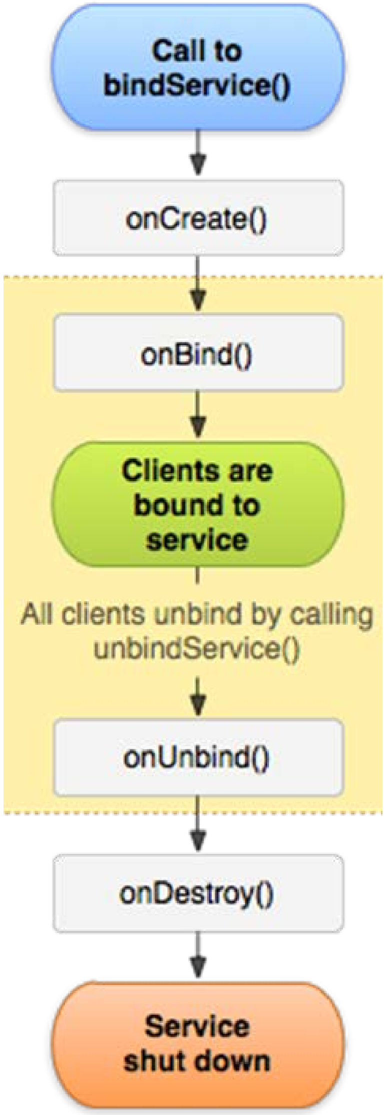
See www.dre.vanderbilt.edu/~schmidt/PDF/Activator.pdf for info on *Activator*

Launching a Bound Service

- A Bound Service allows components to bind to it by calling bindService() to create a “persistent” connection
  - The client must provide ServiceConnection object to monitor the connection with the Service
  - If the Service isn’t already running it will be activated
  - bindService() is not a blocking call

```
Intent intent =
  UniqueIDGeneratorService.makeIntent(this);
bindService(intent, mSvcConn,
            Context.BIND_AUTO_CREATE);
```





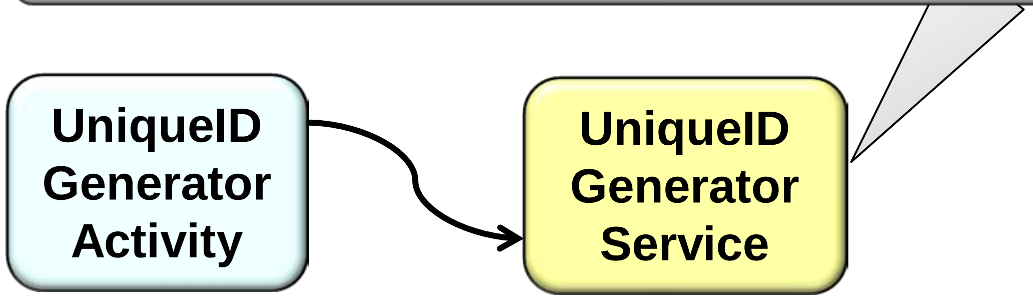
Initializing a Bound Service

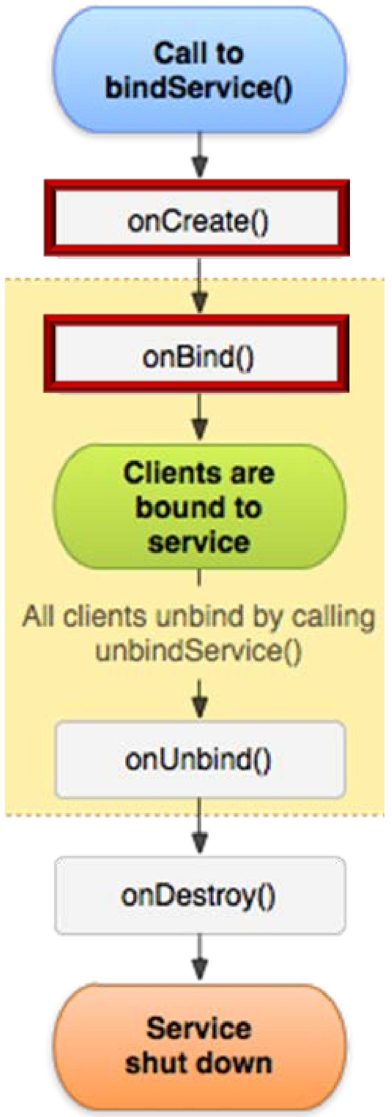
- After Service is running Android invokes its onCreate() & onBind() hook methods

```
public class UniqueIDGeneratorService
       extends Service {
  ...
  public void onCreate() { ... }

  public Ibinder onBind(Intent intent){ ... }

  ...
```





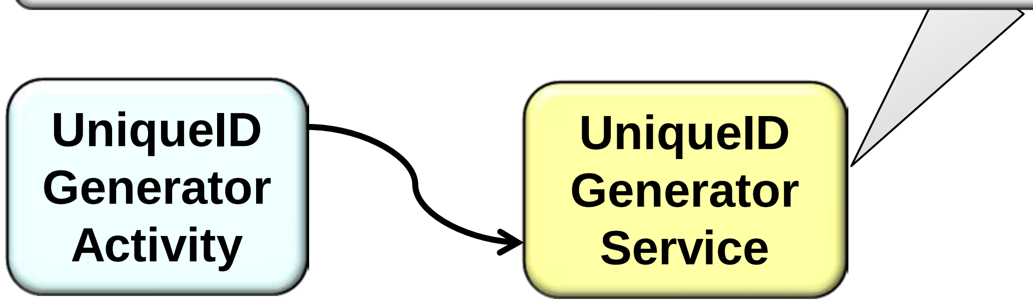
Initializing a Bound Service

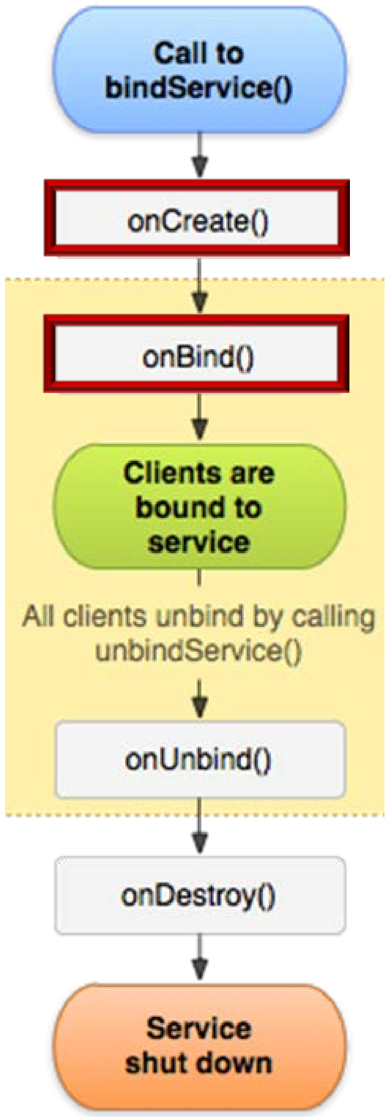
- After Service is running Android invokes its onCreate() & onBind() hook methods

```
public class UniqueIDGeneratorService
       extends Service {
  ...
  public void onCreate() { ... }

  public Ibinder onBind(Intent intent){ ... }

  ...
```





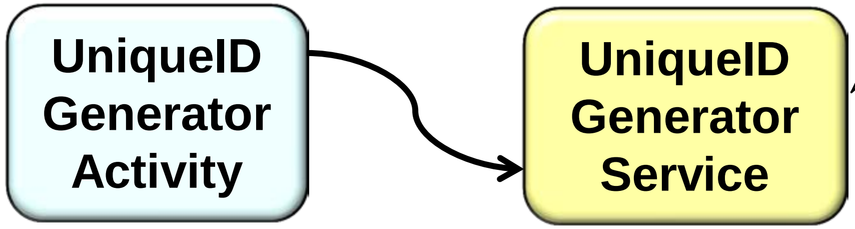
Initializing a Bound Service

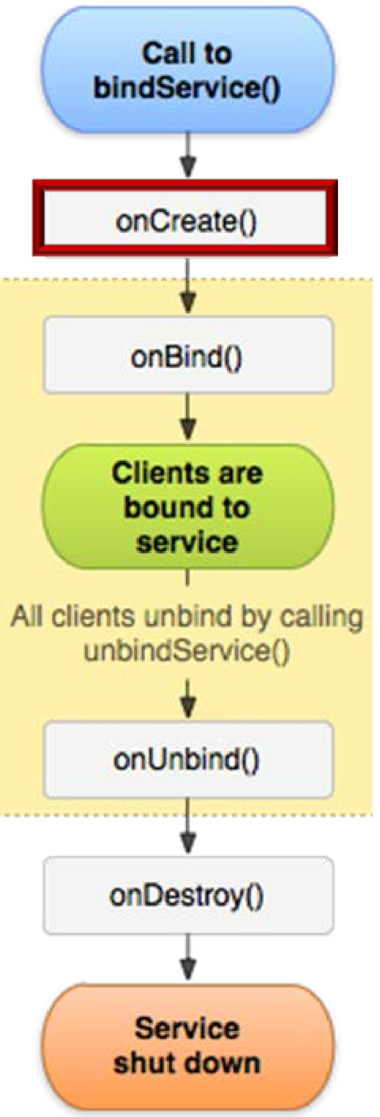
- After Service is running Android invokes its onCreate() & onBind() hook methods
  - onCreate() initializes Service-level data members

```
public void onCreate() {
  mReqMessenger =
    new Messenger(new RequestHandler());

  uniqueIDs =
    PreferenceManager.getDefaultSharedPreferences(this);

  mExecutor =
    Executors.newFixedThreadPool(MAX_THREADS);
}
...
```





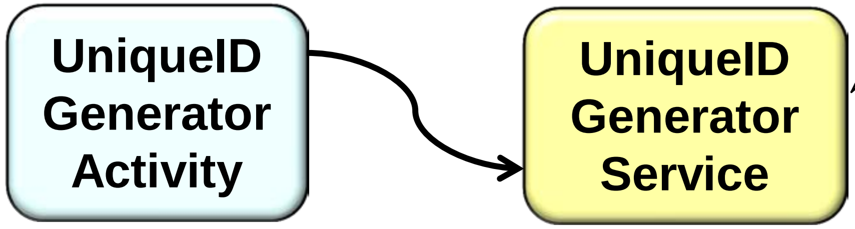
Initializing a Bound Service

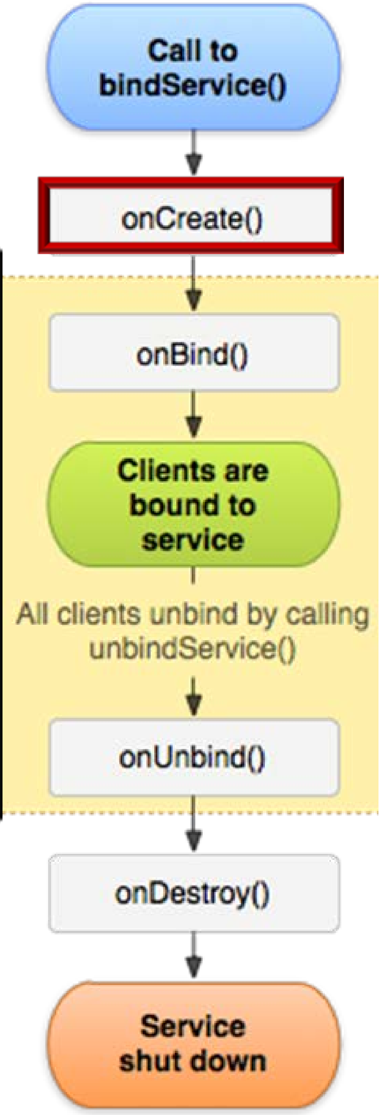
- After Service is running Android invokes its onCreate() & onBind() hook methods
  - onCreate() initializes Service-level data members

```
public void onCreate() {
  mReqMessenger =
    new Messenger(new RequestHandler());

  uniqueIDs =
    PreferenceManager.getDefaultSharedPreferences(this);

  mExecutor =
    Executors.newFixedThreadPool(MAX_THREADS);
}
...
```





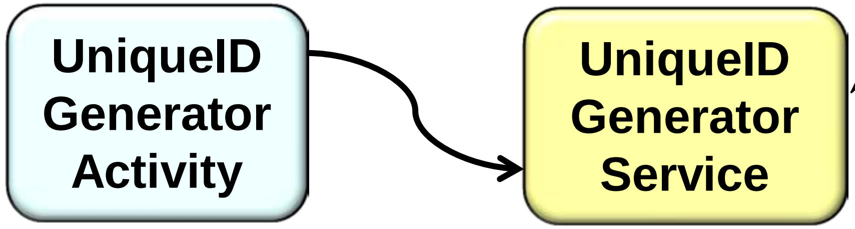
Initializing a Bound Service

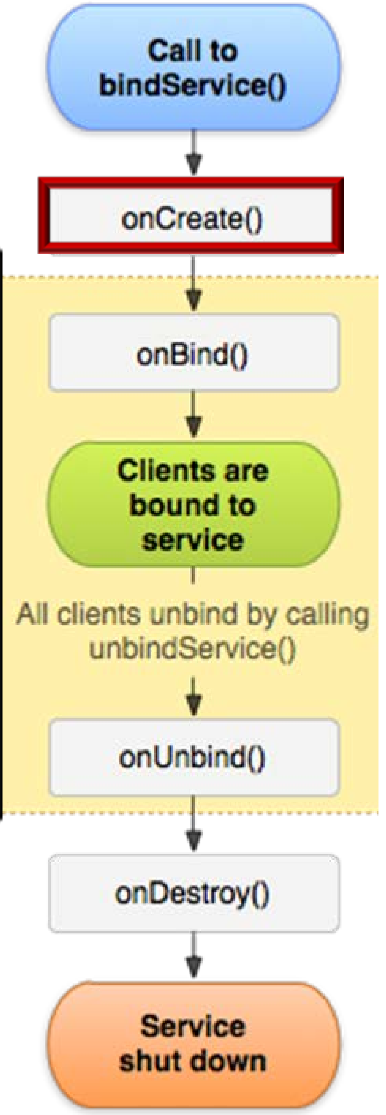
- After Service is running Android invokes its onCreate() & onBind() hook methods
  - onCreate() initializes Service-level data members

```
public void onCreate() {
  mReqMessenger =
    new Messenger(new RequestHandler());

  uniqueIDs =
    PreferenceManager.getDefaultSharedPreferences(this);

  mExecutor =
    Executors.newFixedThreadPool(MAX_THREADS);
}
...
```





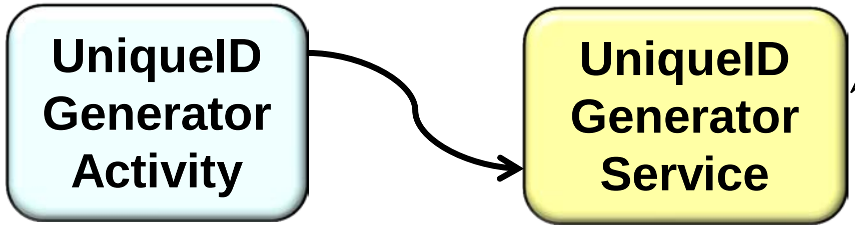
Initializing a Bound Service

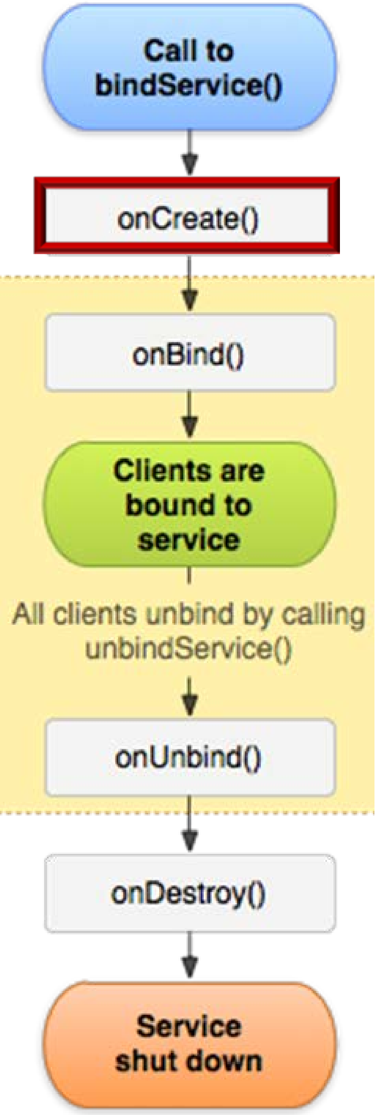
- After Service is running Android invokes its onCreate() & onBind() hook methods
  - onCreate() initializes Service-level data members

```
public void onCreate() {
  mReqMessenger =
    new Messenger(new RequestHandler());

  uniqueIDs =
    PreferenceManager.getDefaultSharedPreferences(this);

  mExecutor =
    Executors.newFixedThreadPool(MAX_THREADS);
}
...
```

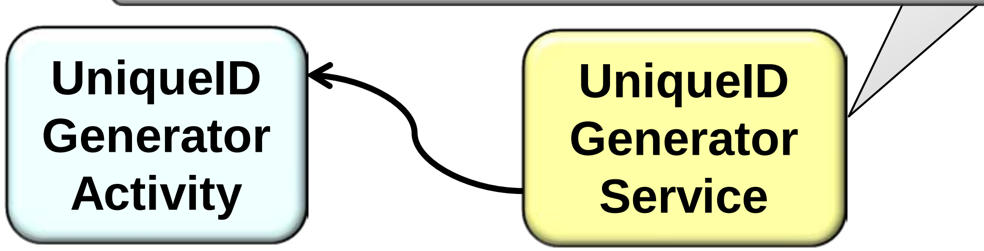


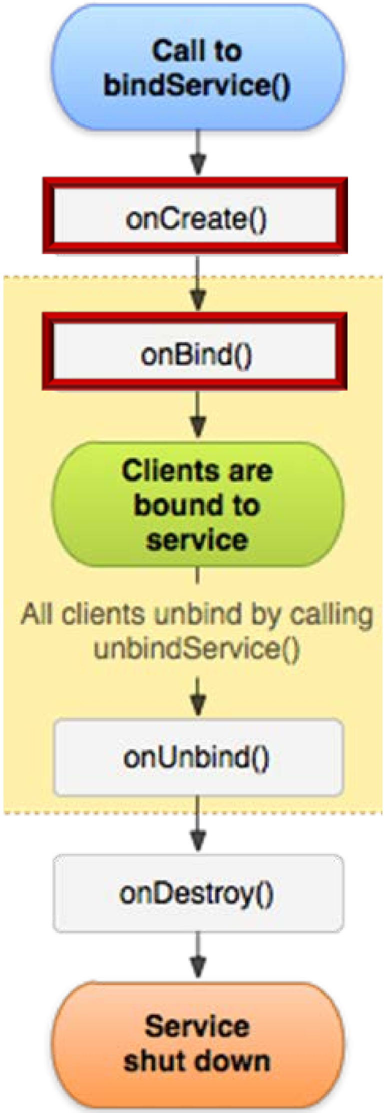


Initializing a Bound Service

- After Service is running Android invokes its onCreate() & onBind() hook methods
  - onCreate() initializes Service-level data members
  - onBind() returns an IBinder that enables the client to communication with the Bound Service

```
...
public IBinder onBind(Intent intent) {
  return mReqMessenger.getBinder();
}
...
```



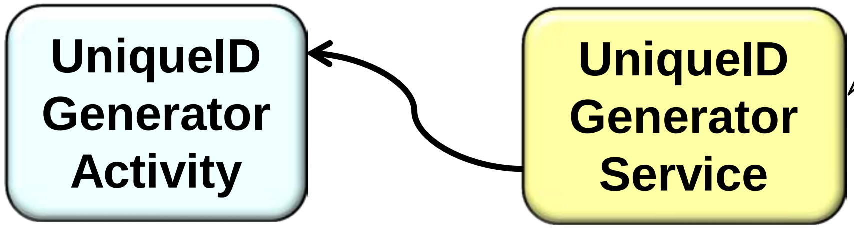


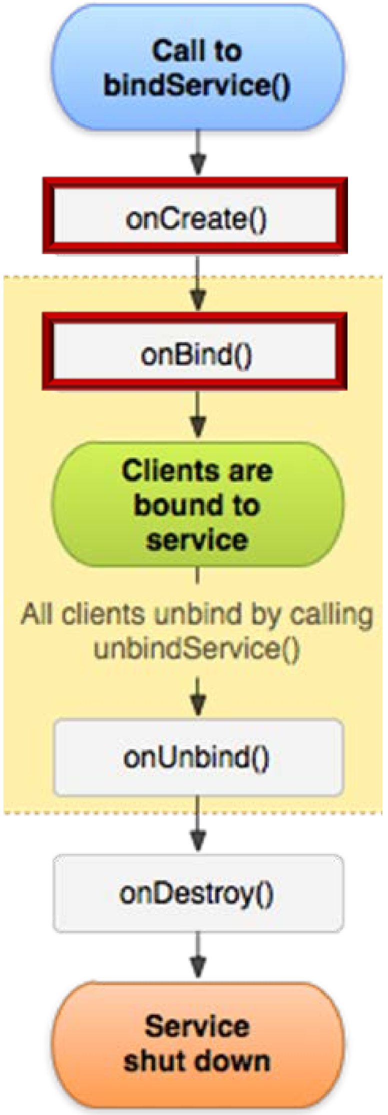
Initializing a Bound Service

- After Service is running Android invokes its onCreate() & onBind() hook methods
  - onCreate() initializes Service-level data members
  - onBind() returns an IBinder that enables the client to communication with the Bound Service

The object returned from onBind() is typically initialized in onCreate() or as a data member

```
...
public IBinder onBind(Intent intent) {
  return mReqMessenger.getBinder();
}
...
```



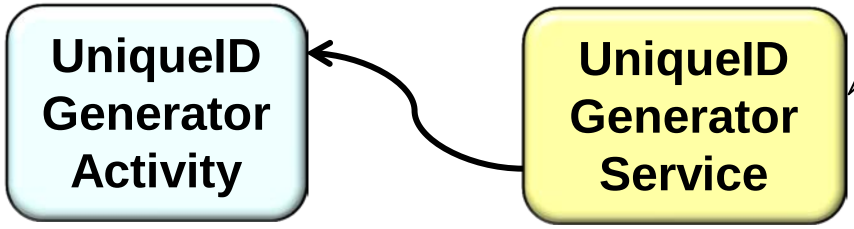


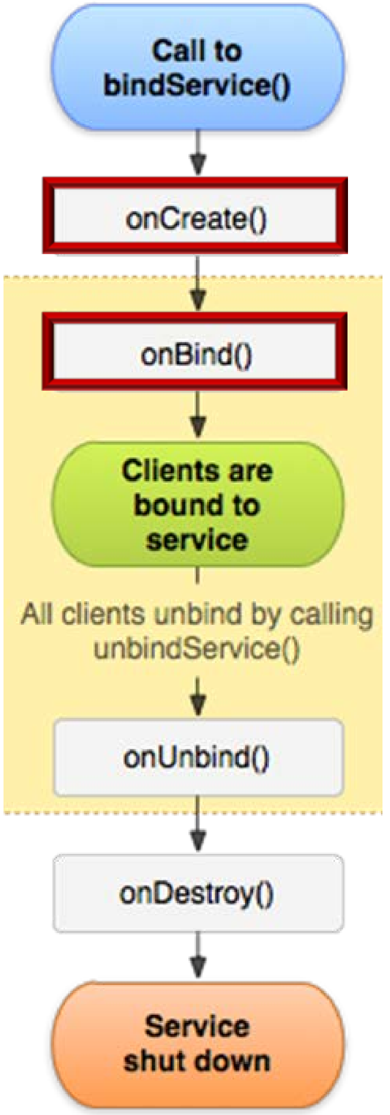
Initializing a Bound Service

- After Service is running Android invokes its onCreate() & onBind() hook methods
  - onCreate() initializes Service-level data members
  - onBind() returns an IBinder that enables the client to communication with the Bound Service

Returns Ibinder the Messenger uses to communicate with its associated request Handler

```
...
public IBinder onBind(Intent intent) {
  return mReqMessenger.getBinder();
}
...
```



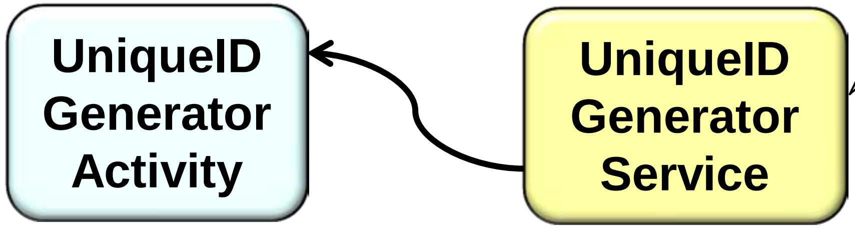


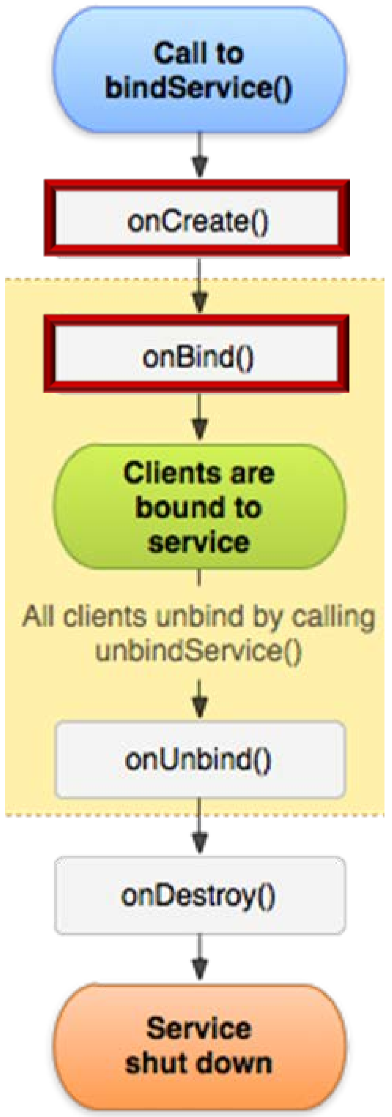
developer.android.com/reference/android/os/Messenger.html#getBinder()

Initializing a Bound Service

- After Service is running Android invokes its onCreate() & onBind() hook methods
  - onCreate() initializes Service-level data members
  - onBind() returns an IBinder that enables the client to communication with the Bound Service
  - onBind() is only called to retrieve the Ibinder when the first client binds to the Service

```
...
public IBinder onBind(Intent intent) {
  return mReqMessenger.getBinder();
}
...
```





developer.android.com/guide/components/bound-services.html

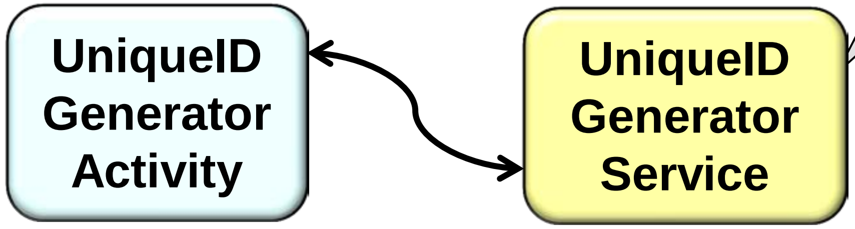
Connecting & Interacting with Bound Services

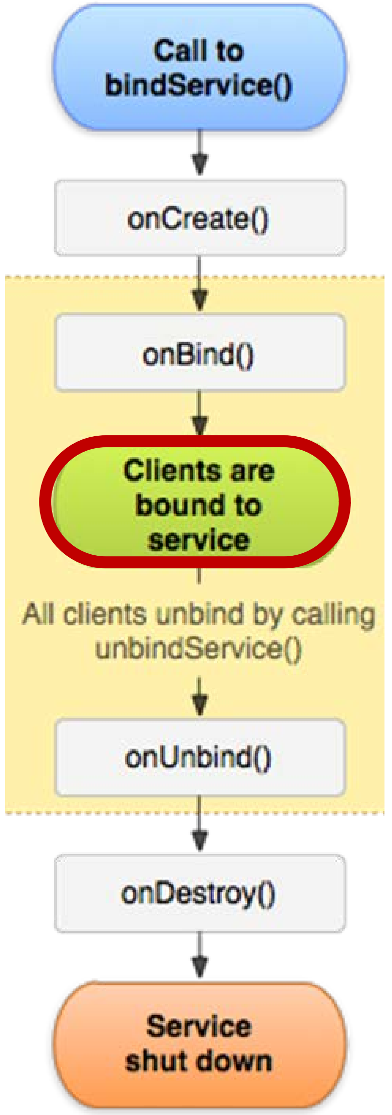
Connecting to a Bound Service

- A callback-driven protocol is used to establish a connection

```
private Messenger mReqMessengerRef = null;

ServiceConnection mSvcConn =
                          new ServiceConnection() {
  public void onServiceConnected
        (ComponentName className, IBinder binder) {
    mReqMessengerRef = new Messenger(binder);
  }
...
```



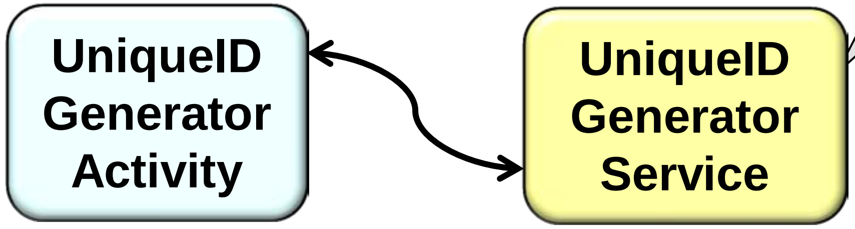


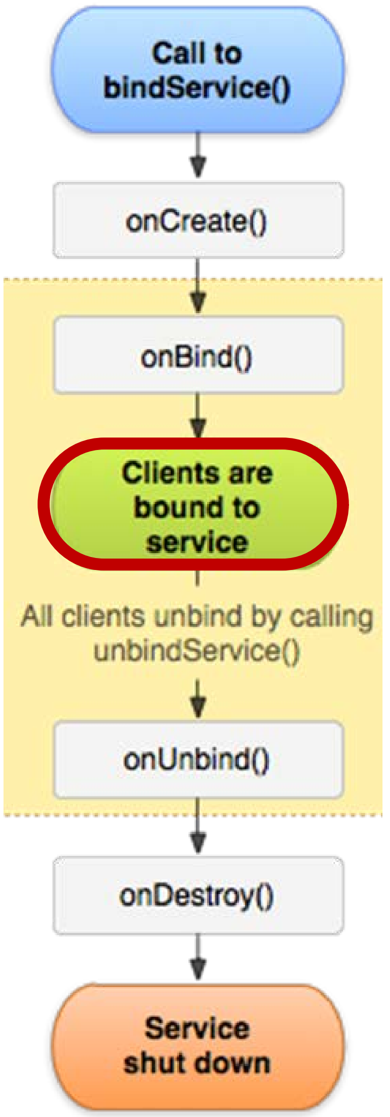
Connecting to a Bound Service

- A callback-driven protocol is used to establish a connection
  - The client implements an onServiceConnected() hook method to get a reference to an object in the Service

```
private Messenger mReqMessengerRef = null;

ServiceConnection mSvcConn =
                          new ServiceConnection() {
  public void onServiceConnected
        (ComponentName className, IBinder binder) {
    mReqMessengerRef = new Messenger(binder);
  }
...
```



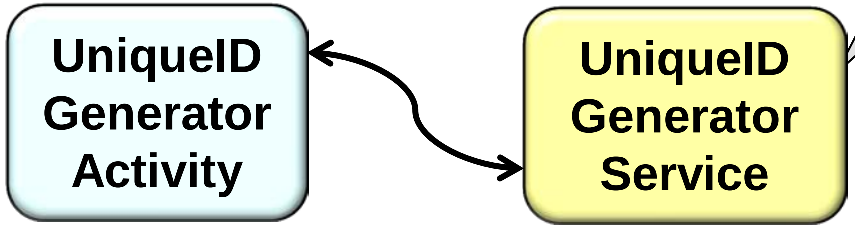


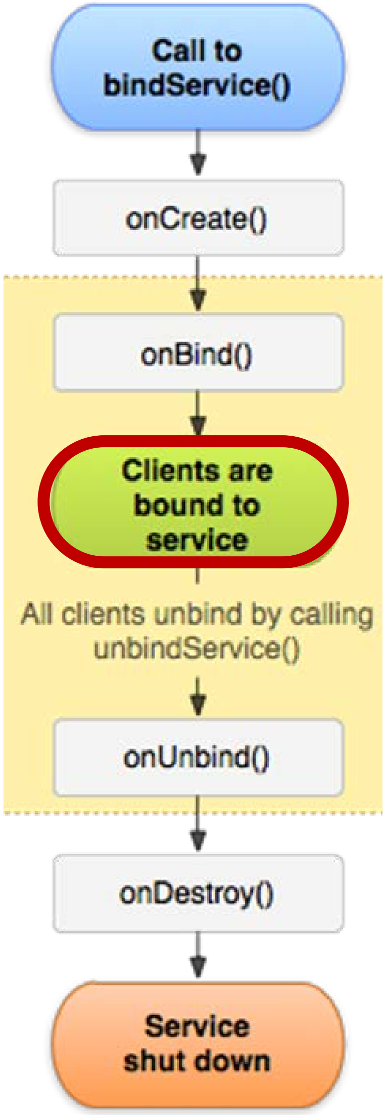
Connecting to a Bound Service

- A callback-driven protocol is used to establish a connection
  - The client implements an onServiceConnected() hook method to get a reference to an object in the Service

```
private Messenger mReqMessengerRef = null;

ServiceConnection mSvcConn =
                          new ServiceConnection() {
  public void onServiceConnected
        (ComponentName className, IBinder binder) {
    mReqMessengerRef = new Messenger(binder);
  }
...
```





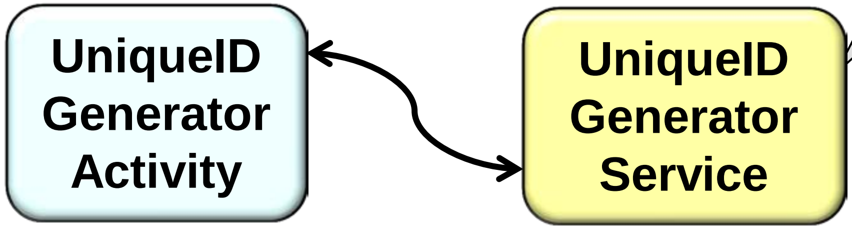
Connecting to a Bound Service

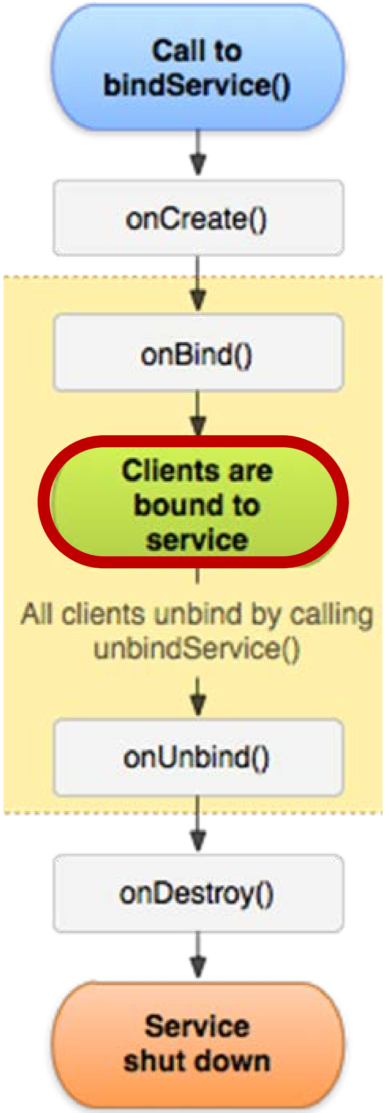
- A callback-driven protocol is used to establish a connection
  - The client implements an onServiceConnected() hook method to get a reference to an object in the Service

Store a reference to the Service's Messenger

```
private Messenger mReqMessengerRef = null;

ServiceConnection mSvcConn =
                          new ServiceConnection() {
  public void onServiceConnected
        (ComponentName className, IBinder binder) {
    mReqMessengerRef = new Messenger(binder);
  }
...
```



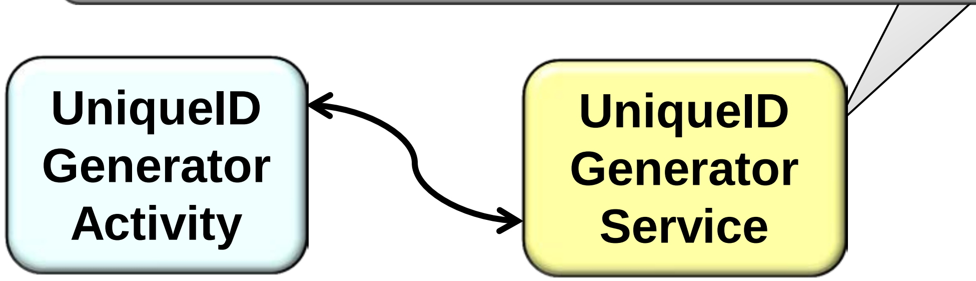


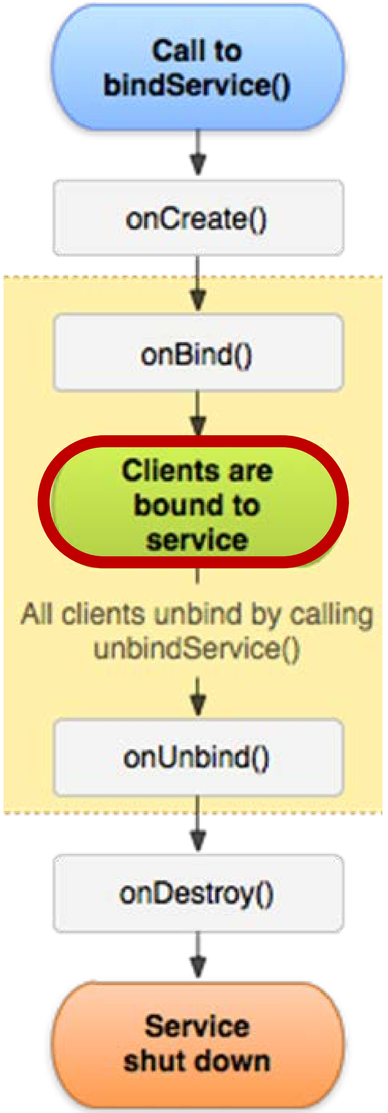
Connecting to a Bound Service

- A callback-driven protocol is used to establish a connection
  - The client implements an onServiceConnected() hook method to get a reference to an object in the Service

```
private Messenger mReqMessengerRef = null;

ServiceConnection mSvcConn =
                         new ServiceConnection() {
  ...
  public void onServiceDisconnected
        (ComponentName className) {
    mReqMessengerRef = null;
  }
...
```





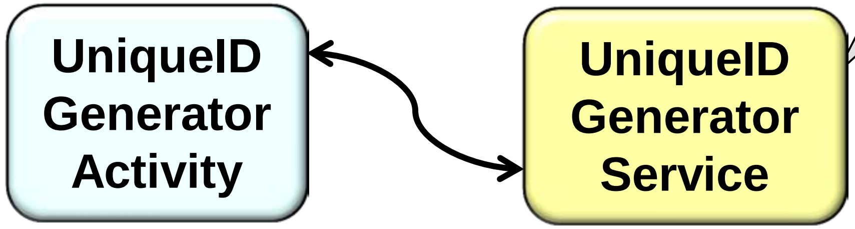
Connecting to a Bound Service

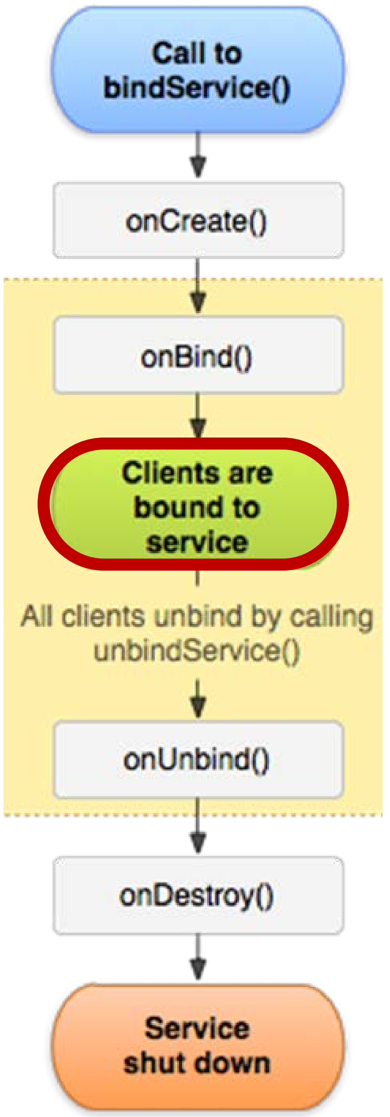
- A callback-driven protocol is used to establish a connection
  - The client implements an onServiceConnected() hook method to get a reference to an object in the Service

```
private Messenger mReqMessengerRef = null;

ServiceConnection mSvcConn =
                          new ServiceConnection() {
  ...
  public void onServiceDisconnected
        (ComponentName className) {
    mReqMessengerRef = null;
  }
...
```

Called back when the process hosting the Service has crashed or is killed





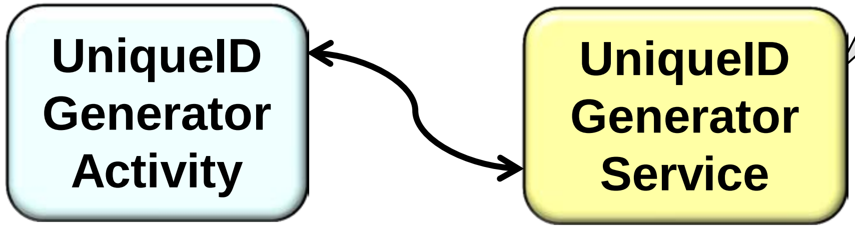
Connecting to a Bound Service

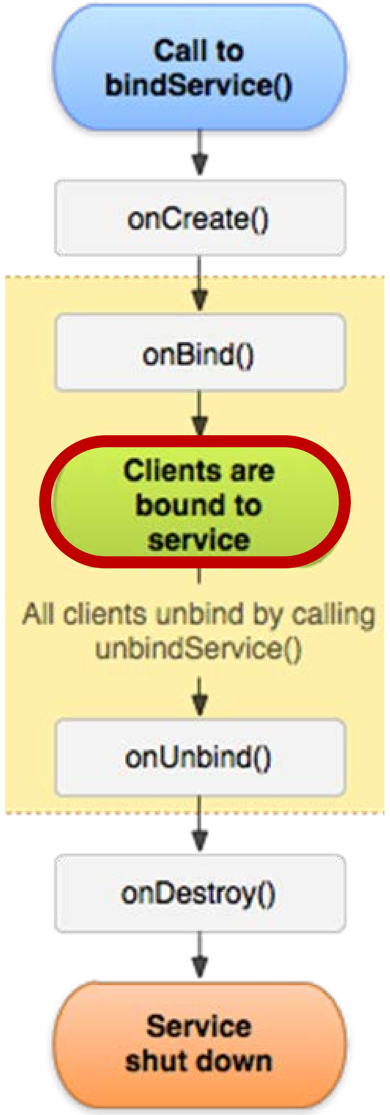
- A callback-driven protocol is used to establish a connection
  - The client implements an onServiceConnected() hook method to get a reference to an object in the Service

```
private Messenger mReqMessengerRef = null;

ServiceConnection mSvcConn =
                          new ServiceConnection() {
  ...
  public void onServiceDisconnected
        (ComponentName className) {
    mReqMessengerRef = null;
  }
...
```

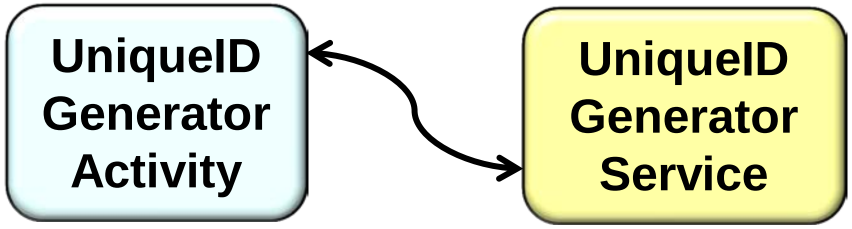
This callback does not remove the Service Connection

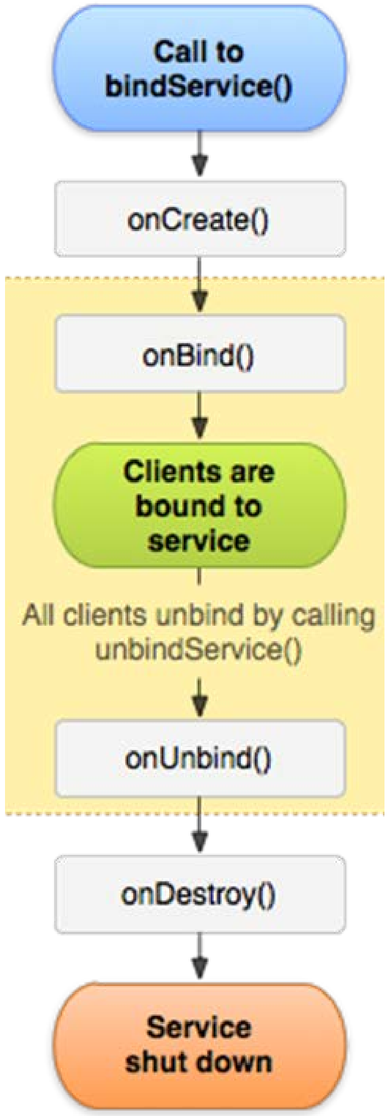




Interacting with a Bound Service

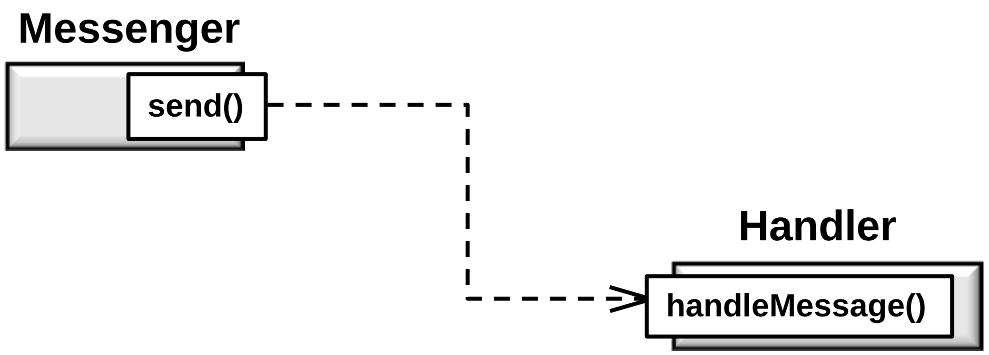
- A Bound Service offers clients an interface they can use to interact with the Service

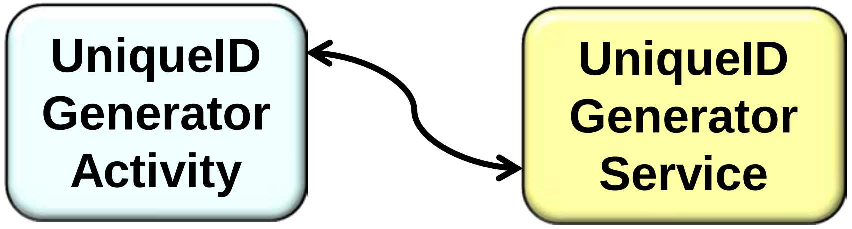


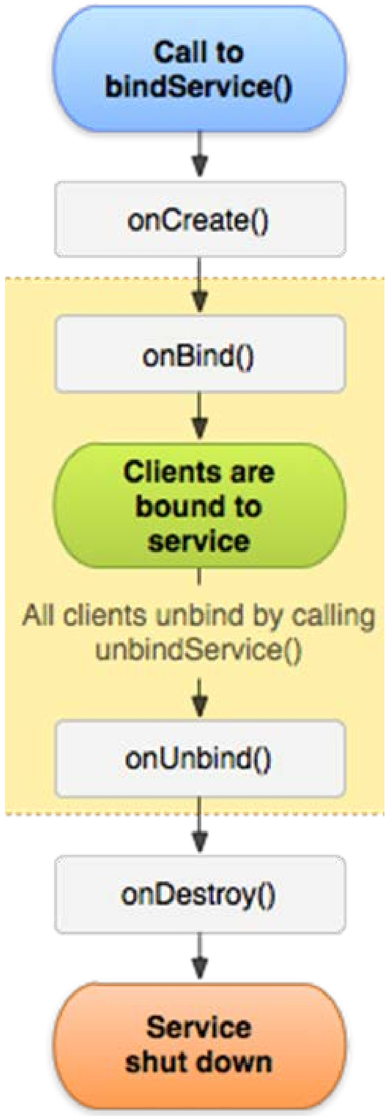


Interacting with a Bound Service

- A Bound Service offers clients an interface they can use to interact with the Service
  - This interface can be generic
    - e.g., using Messengers & Handlers for inter- or intra-process communication



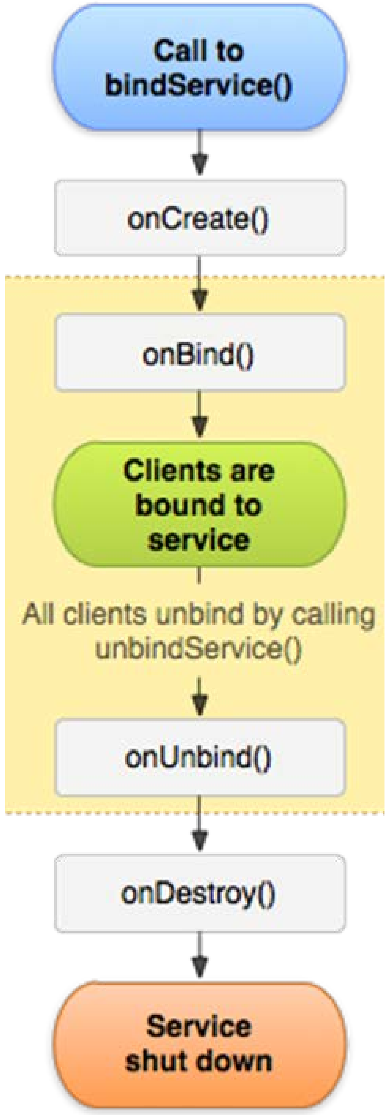


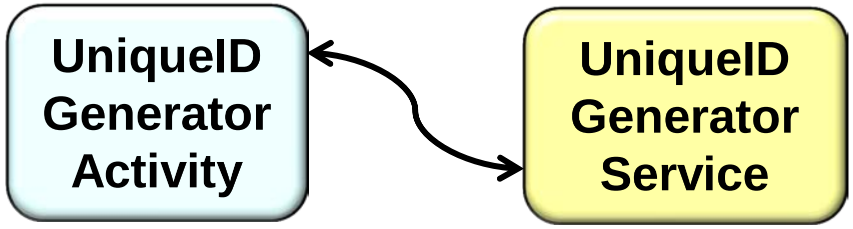


Interacting with a Bound Service

- A Bound Service offers clients an interface they can use to interact with the Service
  - This interface can be generic
  - This interface can also be type-specific
    - e.g., using the Android Interface Definition Language (AIDL) for inter- or intra-process communication

```
interface UniqueIDGenerator {
    String uniqueID();
}
```

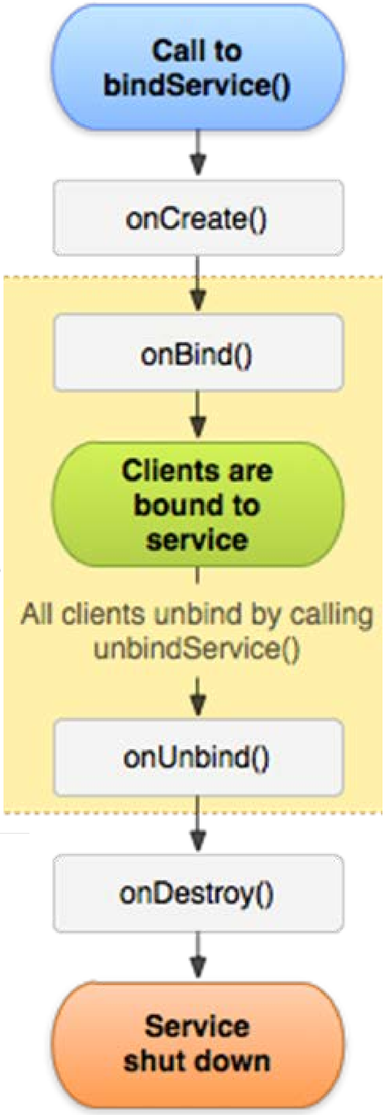


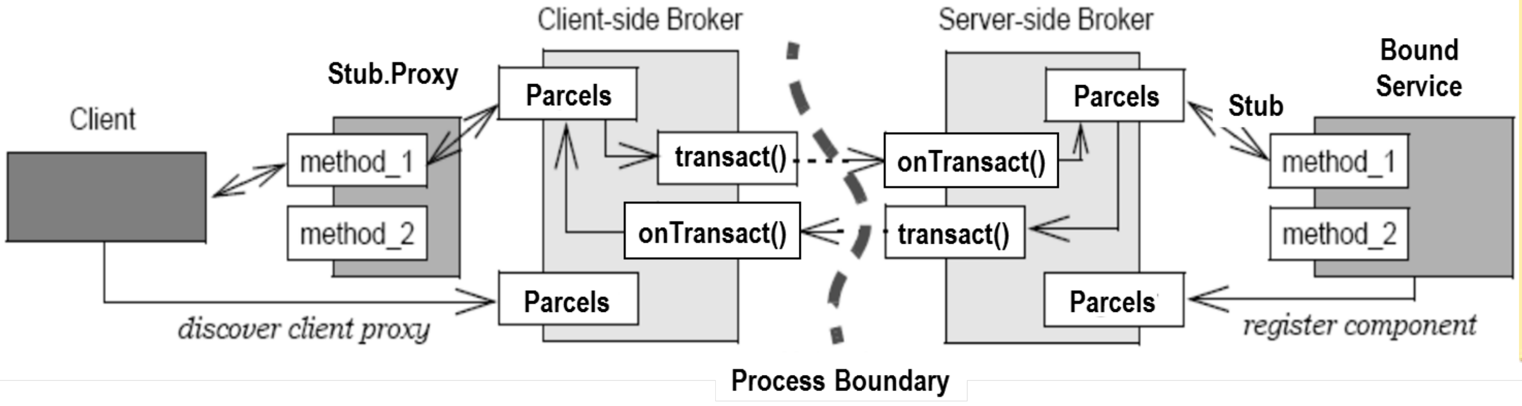


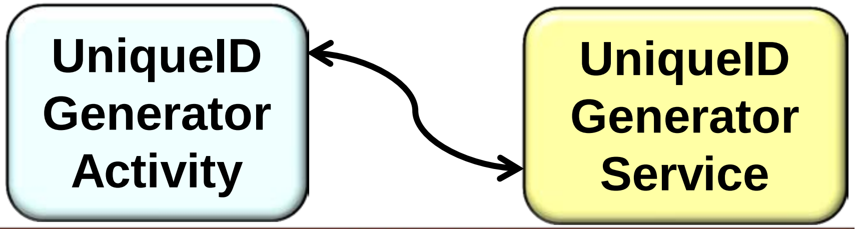
See upcoming part on “Programming Bound Services with AIDL”

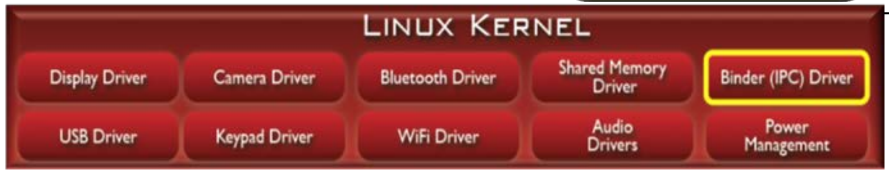
Interacting with a Bound Service

- A Bound Service offers clients an interface they can use to interact with the Service
  - This interface can be generic
  - This interface can also be type-specific
  - Both approaches use the Android Binder framework
    - Implements patterns like *Broker* & *Proxy*



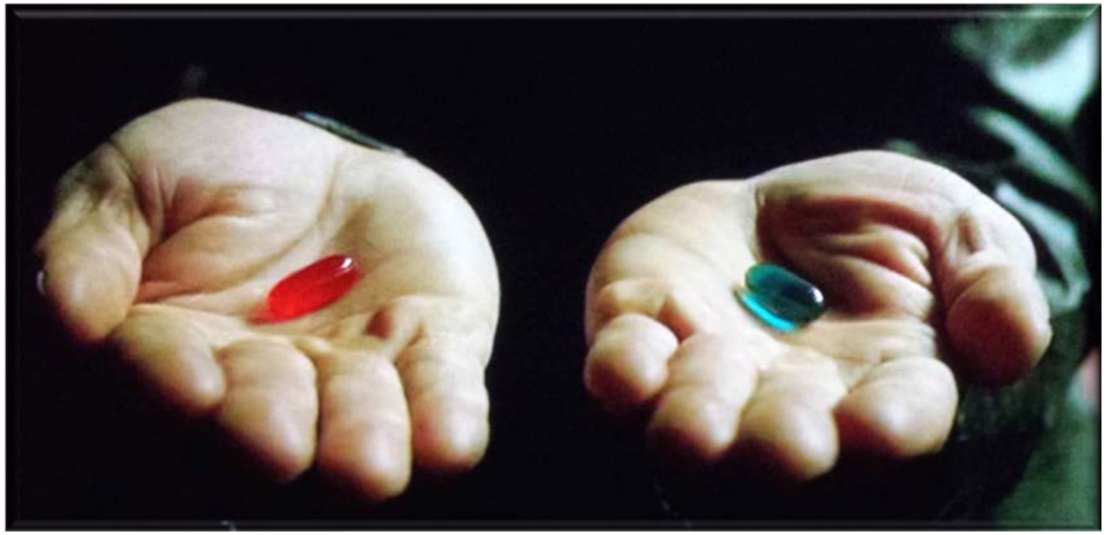


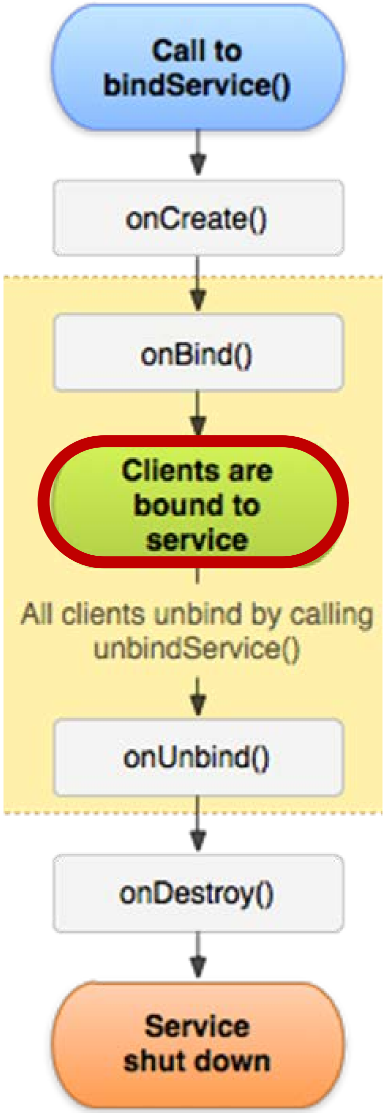


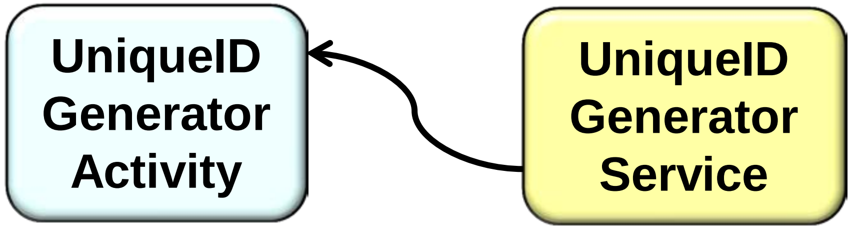


Interacting with a Bound Service

- A Bound Service offers clients an interface they can use to interact with the Service
- The UniqueIDGenerator application uses a pair of Messengers to communicate





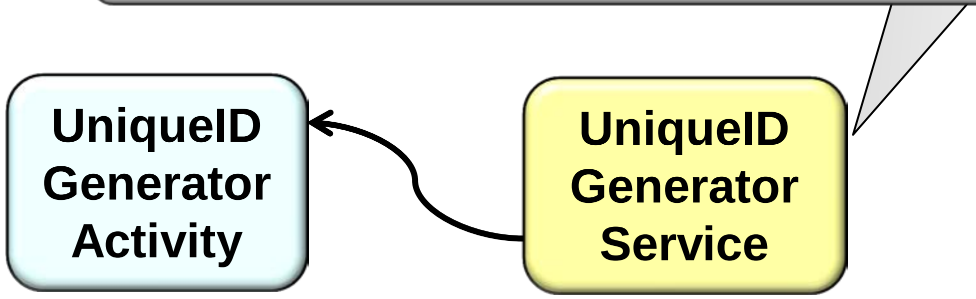


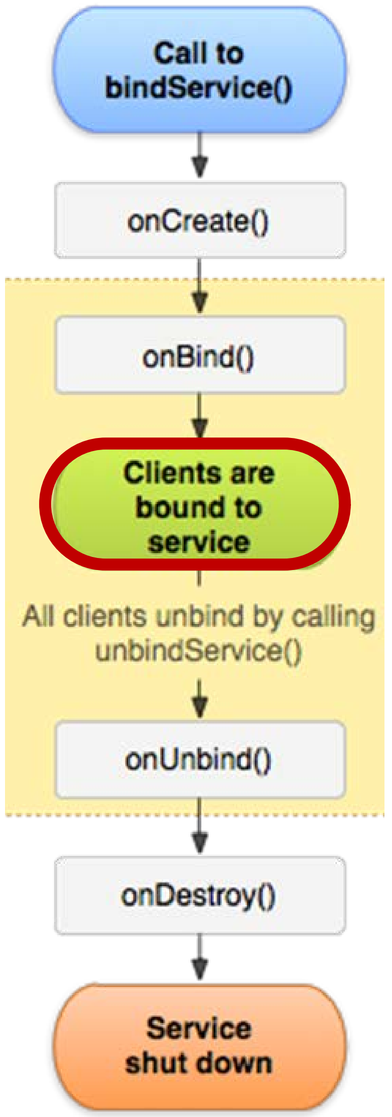
Interacting with a Bound Service

- A Bound Service offers clients an interface they can use to interact with the Service
- The UniqueIDGenerator application uses a pair of Messengers to communicate

```
public void onCreate() {
  mReqMessenger =
    new Messenger(new RequestHandler());
  ...

private class RequestHandler extends Handler {
  public void handleMessage(Message request) {
    ...
    mExecutor.execute(...);
    ...
```



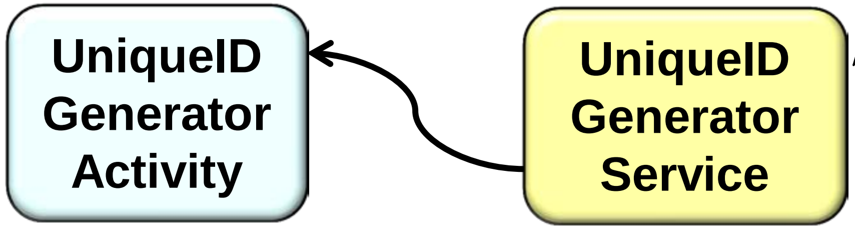


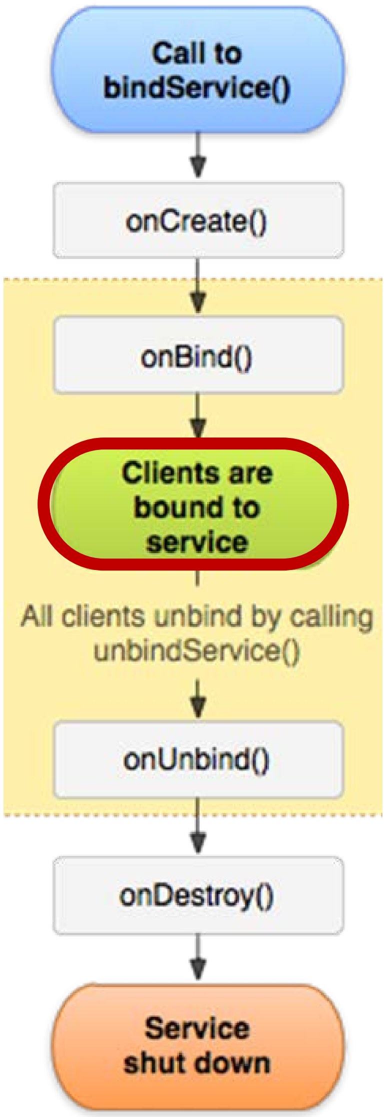
Interacting with a Bound Service

- A Bound Service offers clients an interface they can use to interact with the Service
- The UniqueIDGenerator application uses a pair of Messengers to communicate

```
public void onCreate() {
  mReqMessenger =
    new Messenger(new RequestHandler());
  ...

private class RequestHandler extends Handler {
  public void handleMessage(Message request) {
    ...
    mExecutor.execute(...);
    ...
```

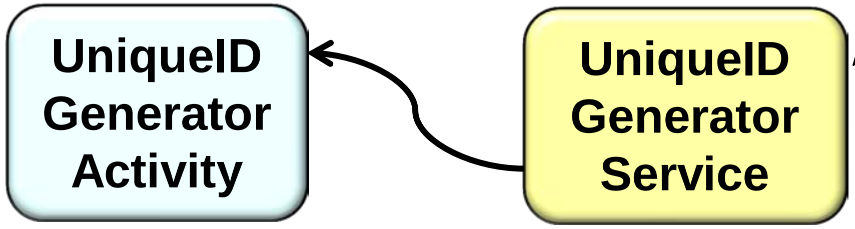


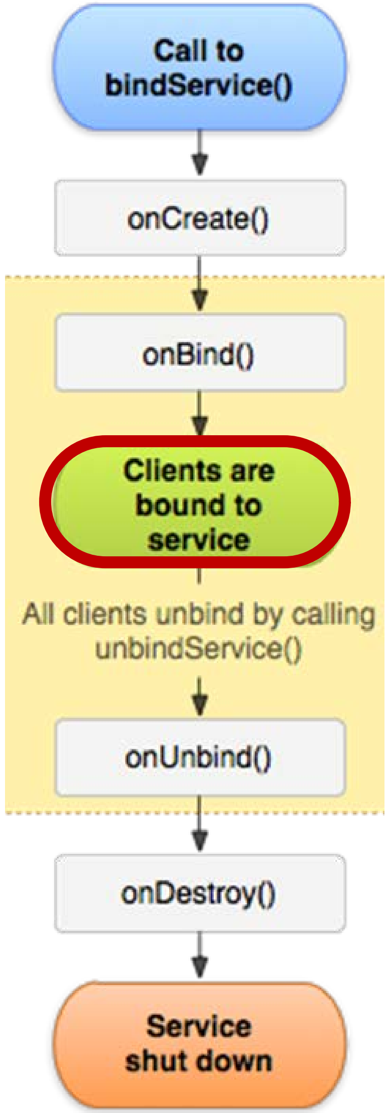


Interacting with a Bound Service

- A Bound Service offers clients an interface they can use to interact with the Service
- The UniqueIDGenerator application uses a pair of Messengers to communicate

```
...
public IBinder onBind(Intent intent) {
  return mReqMessenger.getBinder();
}
...
```



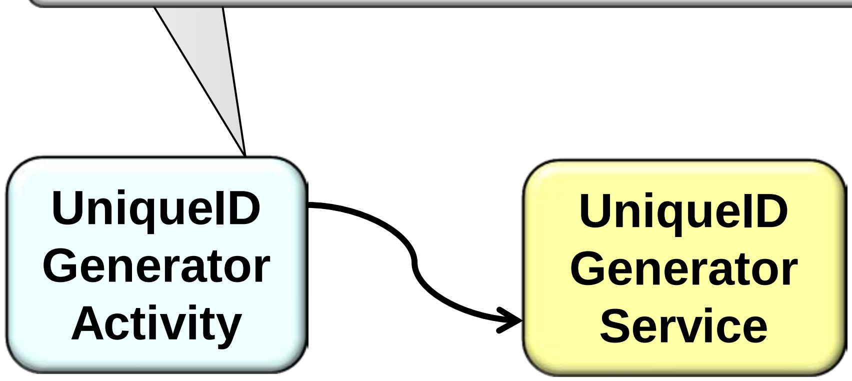


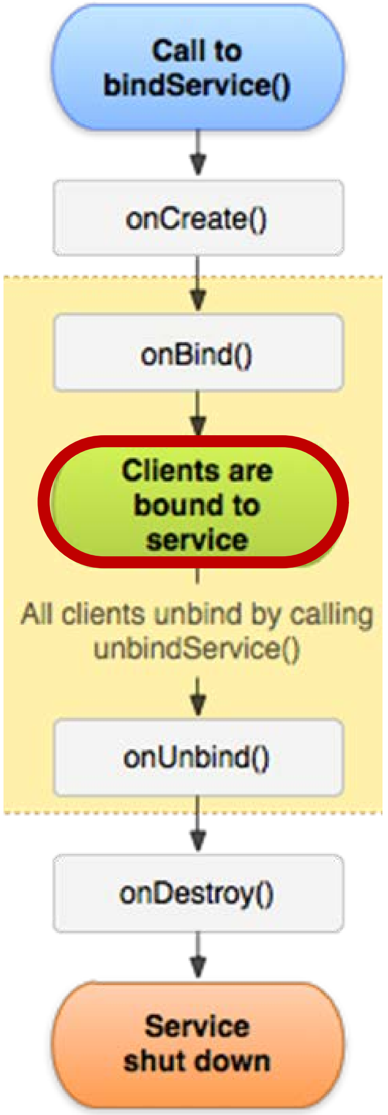
Interacting with a Bound Service

- A Bound Service offers clients an interface they can use to interact with the Service
- The UniqueIDGenerator application uses a pair of Messengers to communicate

```
public void generateUniqueID(View view) {
  Message request = Message.obtain();
  request.replyTo =
    new Messenger(new ReplyHandler());
  ...

  mReqMessengerRef.send(request);
  ...
```



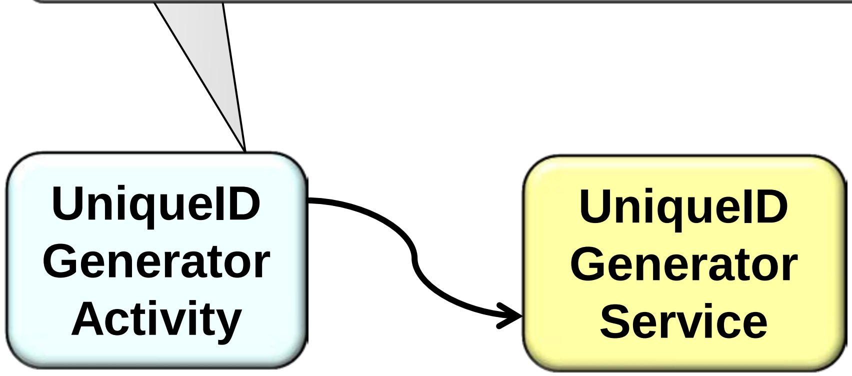


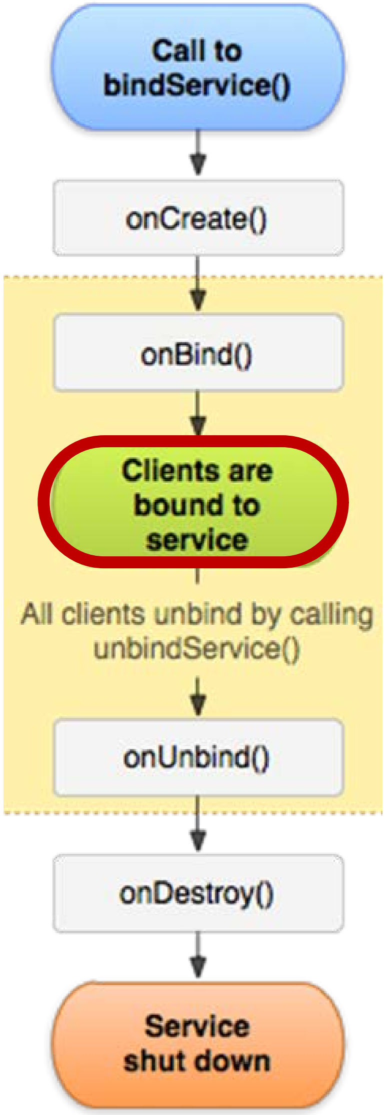
Interacting with a Bound Service

- A Bound Service offers clients an interface they can use to interact with the Service
- The UniqueIDGenerator application uses a pair of Messengers to communicate

```
public void generateUniqueID(View view) {
  Message request = Message.obtain();
  request.replyTo =
    new Messenger(new ReplyHandler());
  ...

  mReqMessengerRef.send(request);
  ...
```



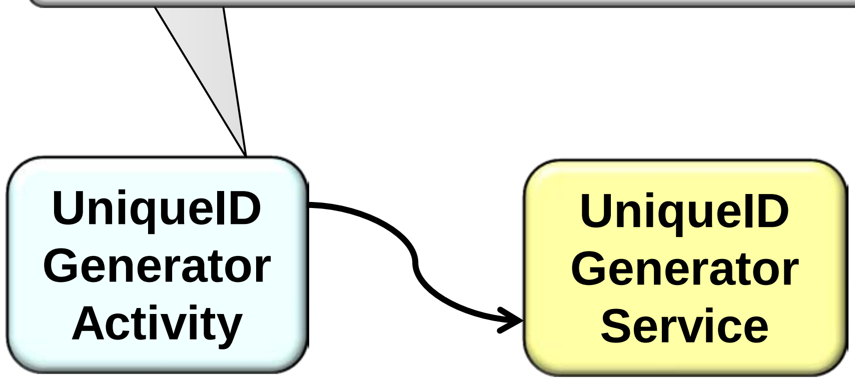


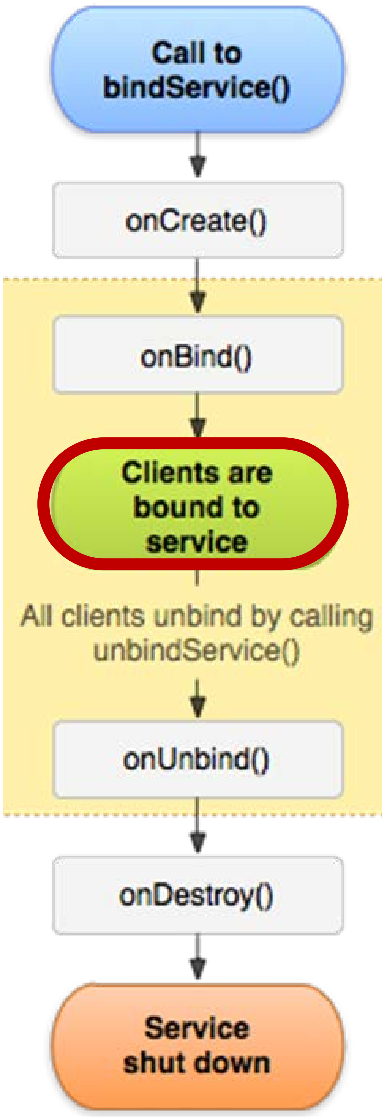
Interacting with a Bound Service

- A Bound Service offers clients an interface they can use to interact with the Service
- The UniqueIDGenerator application uses a pair of Messengers to communicate

```
public void generateUniqueID(View view) {
  Message request = Message.obtain();
  request.replyTo =
    new Messenger(new ReplyHandler());
  ...

  mReqMessengerRef.send(request);
  ...
```

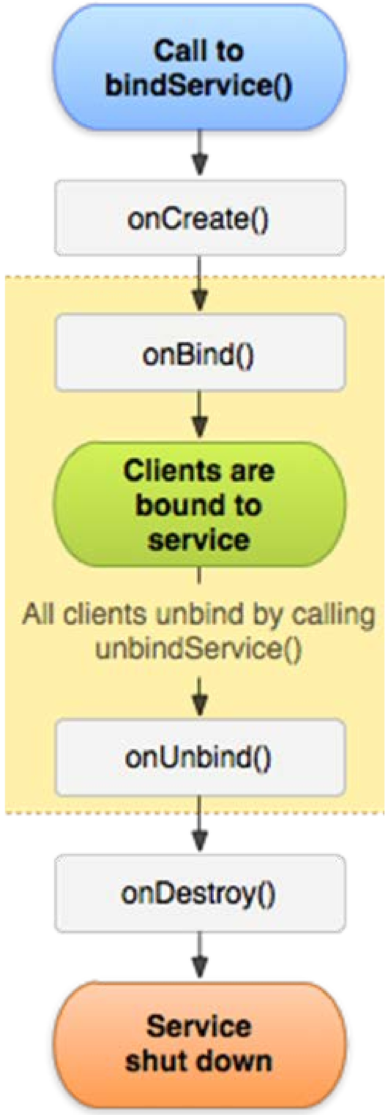


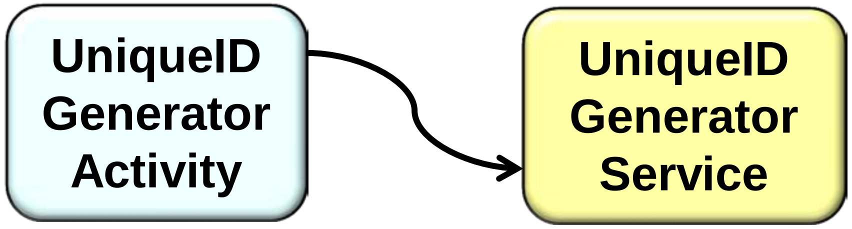


Stopping Bound Services

Stopping a Bound Service

- When a Bound Service is launched, it has a lifecycle that depends on the component(s) accessing it

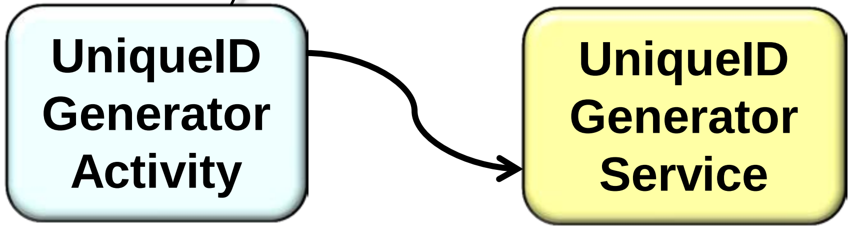


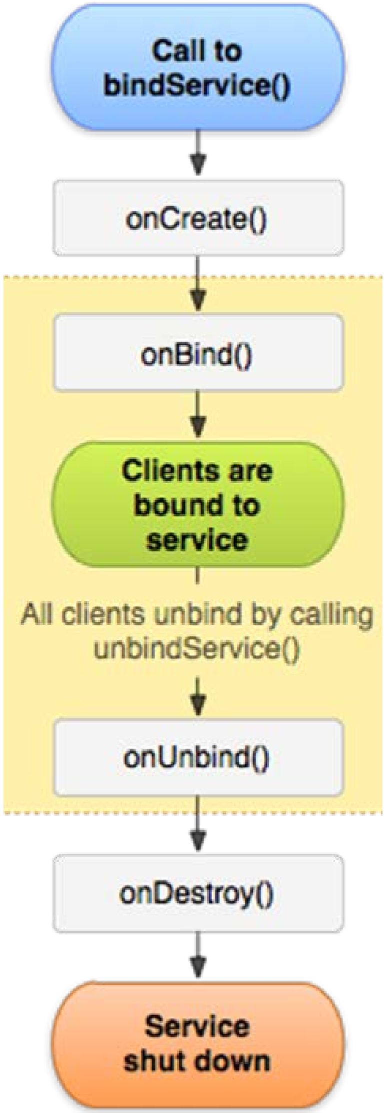


Stopping a Bound Service

- When a Bound Service is launched, it has a lifecycle that depends on the component(s) accessing it
  - i.e., it typically doesn't run in the background indefinitely, but instead is managed automatically

When a client is done interacting with a Bound Service, it calls unbindService() to unbind



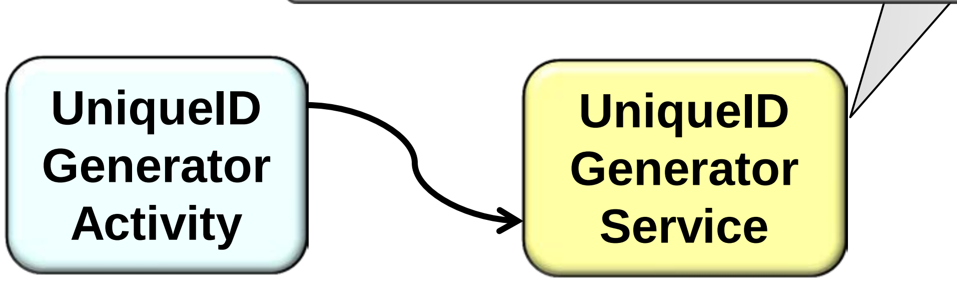


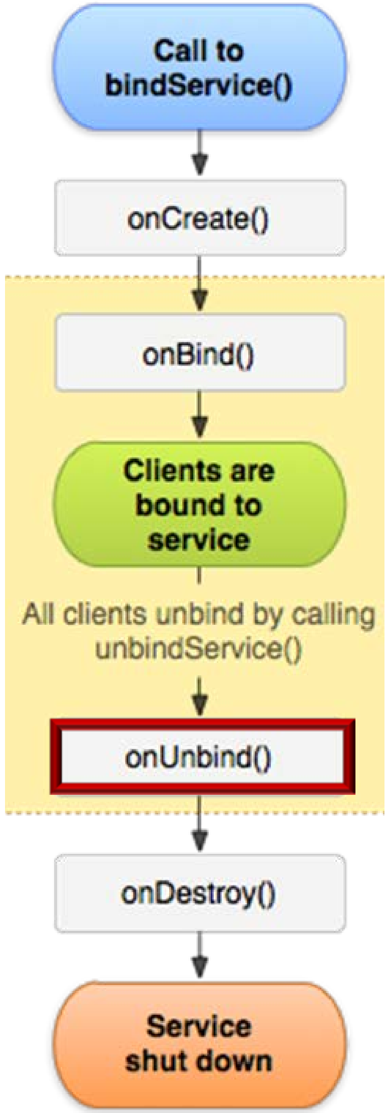
developer.android.com/guide/components/bound-services.html#Lifecycle

Stopping a Bound Service

- When a Bound Service is launched, it has a lifecycle that depends on the component(s) accessing it
  - i.e., it typically doesn't run in the background indefinitely, but instead is managed automatically

```
public abstract class Service extends ... {
  ...
  public boolean onUnbind(Intent intent) {
    return false;
  }
  ...
```

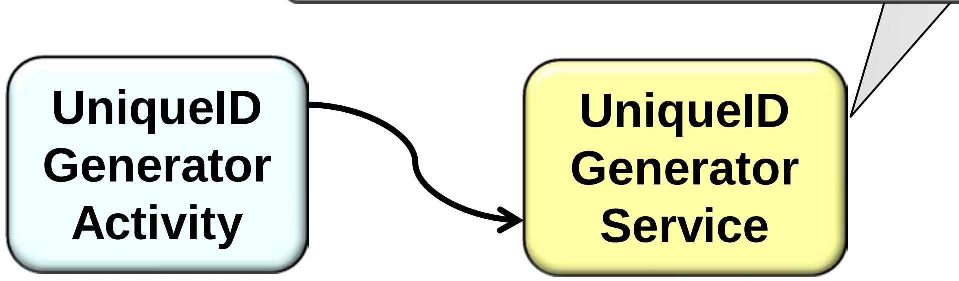


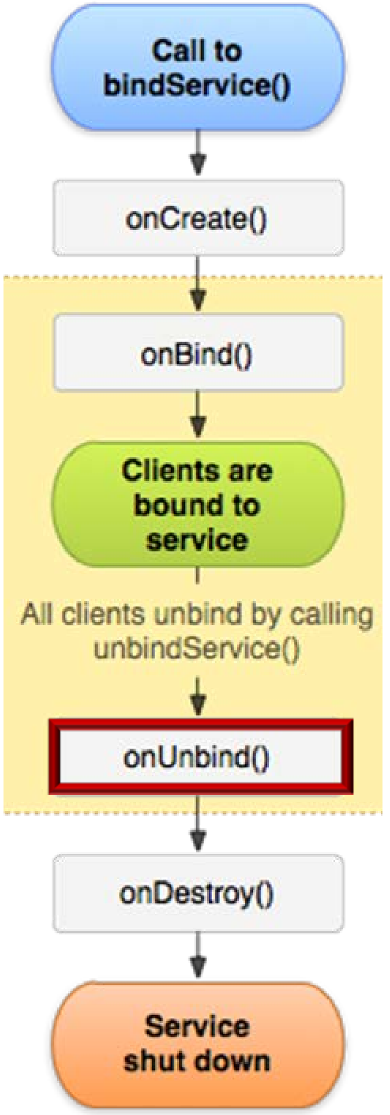


Stopping a Bound Service

- When a Bound Service is launched, it has a lifecycle that depends on the component(s) accessing it
  - i.e., it typically doesn't run in the background indefinitely, but instead is managed automatically

```
public abstract class Service extends ... {
  ...
  public boolean onUnbind(Intent intent) {
    return false;
  }
  ...
```



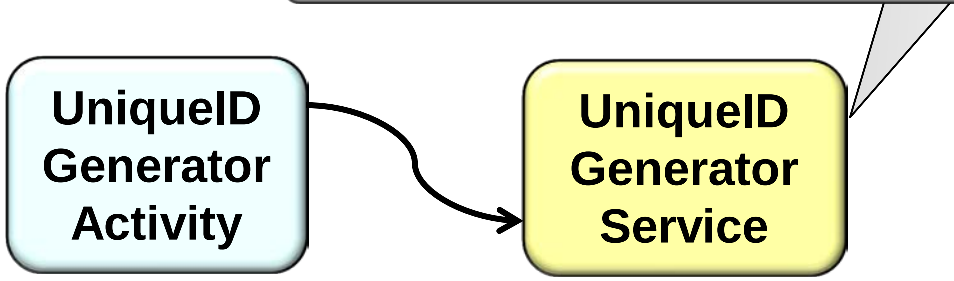


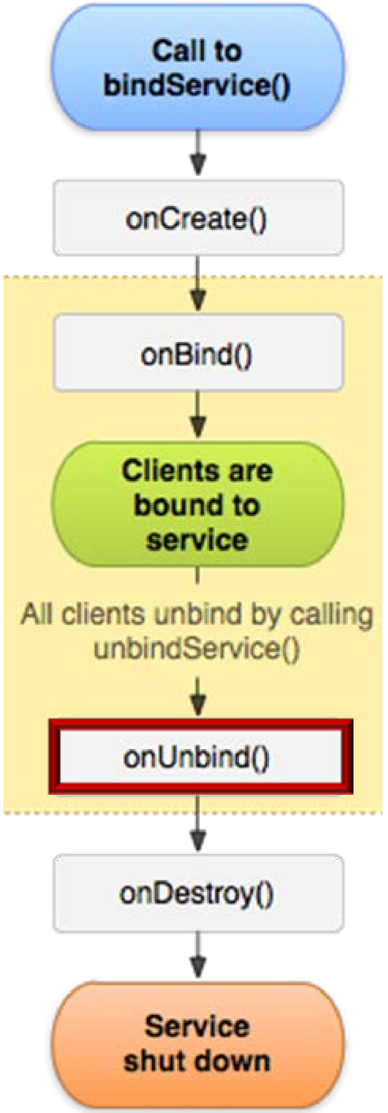
Stopping a Bound Service

- When a Bound Service is launched, it has a lifecycle that depends on the component(s) accessing it
  - i.e., it typically doesn't run in the background indefinitely, but instead is managed automatically

Returning true from onUnbind() enables onRebind() to be called when new clients bind to a Service

```
public class UniqueIDGenerorService
       extends Service {
  ...
  public boolean onUnbind(Intent intent) {
    return true;
  }
  ...
```



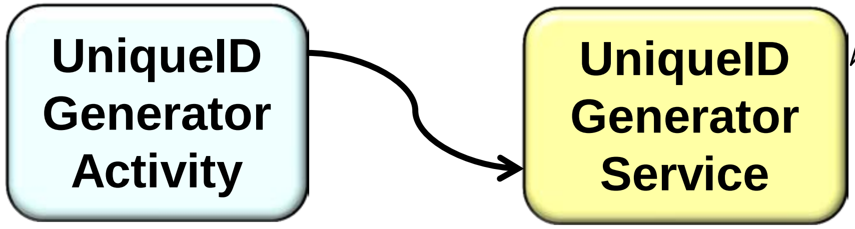


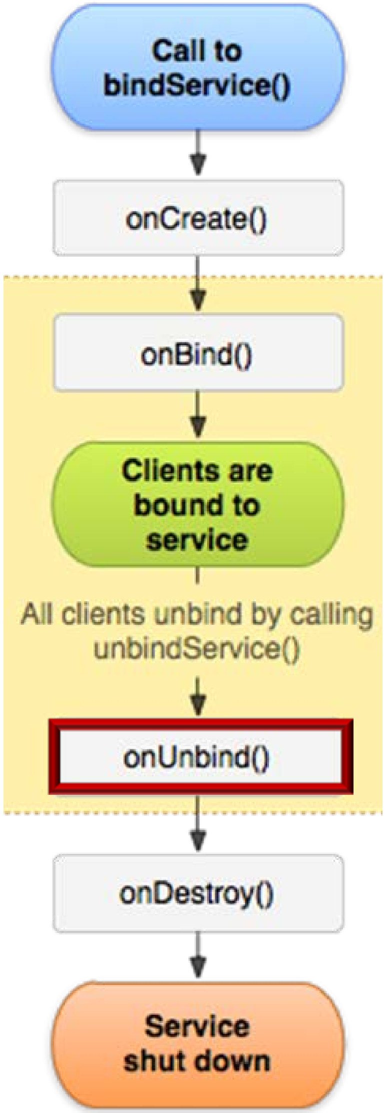
Stopping a Bound Service

- When a Bound Service is launched, it has a lifecycle that depends on the component(s) accessing it
  - i.e., it typically doesn't run in the background indefinitely, but instead is managed automatically

onRebind() can be used if a Bound Service is also a Started Service

```
public class UniqueIDGenerorService
           extends Service {
   ...
   public boolean onRebind(Intent intent) {
     ...
```



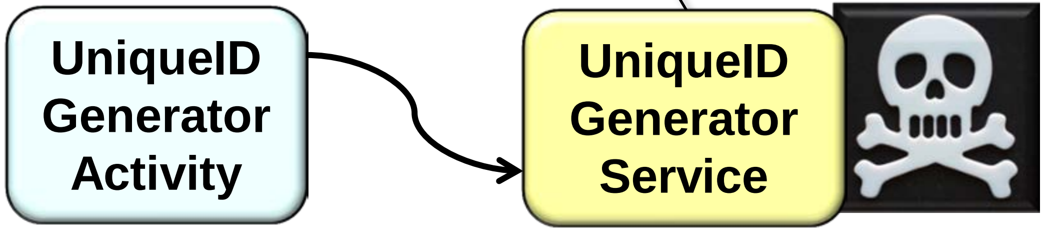


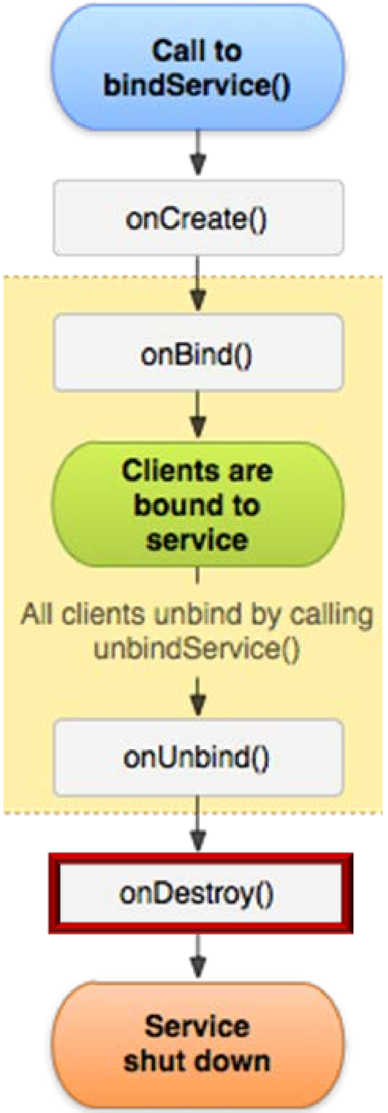
developer.android.com/guide/components/services.html#Lifecycle

Stopping a Bound Service

- When a Bound Service is launched, it has a lifecycle that depends on the component(s) accessing it
  - i.e., it typically doesn't run in the background indefinitely, but instead is managed automatically

A Bound Service is typically destroyed when there are no clients bound to the



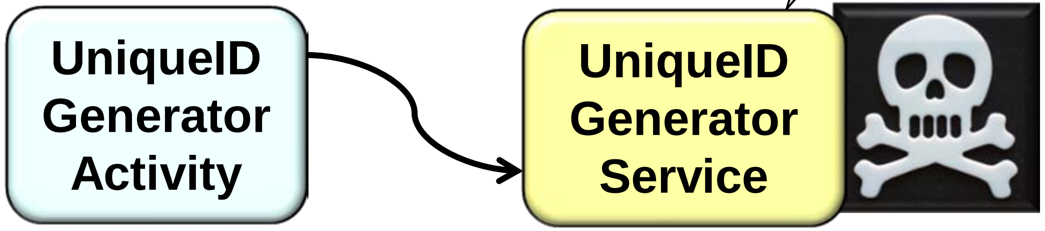


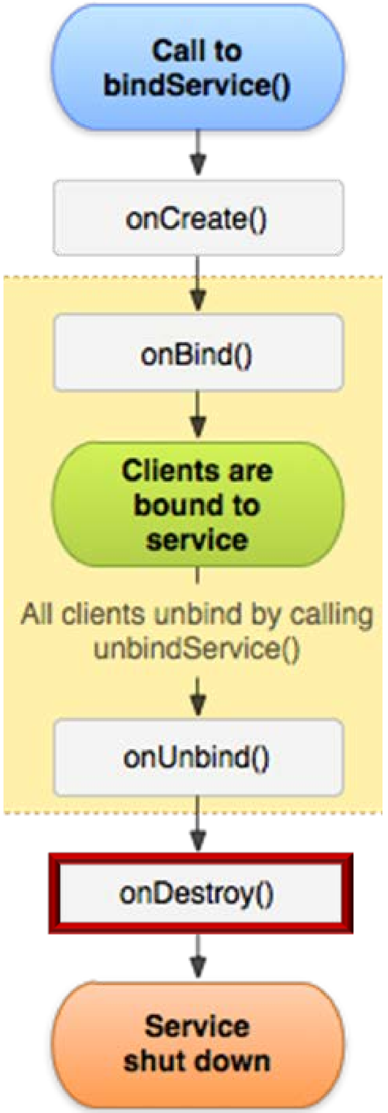
Stopping a Bound Service

- When a Bound Service is launched, it has a lifecycle that depends on the component(s) accessing it
  - i.e., it typically doesn't run in the background indefinitely, but instead is managed automatically

Cleanup any resources that were allocated

```
...
public void onDestroy() {
    mExecutor.shutdown();
}
...
```



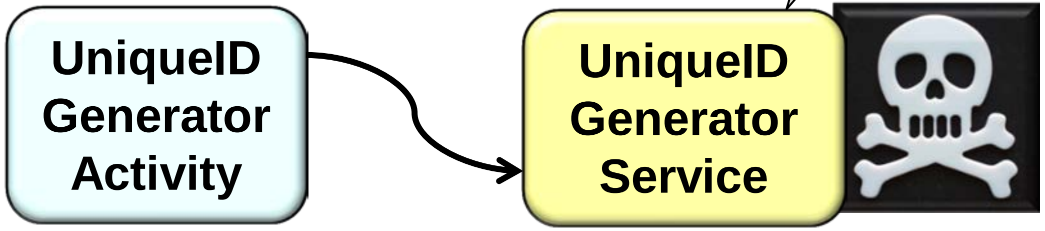


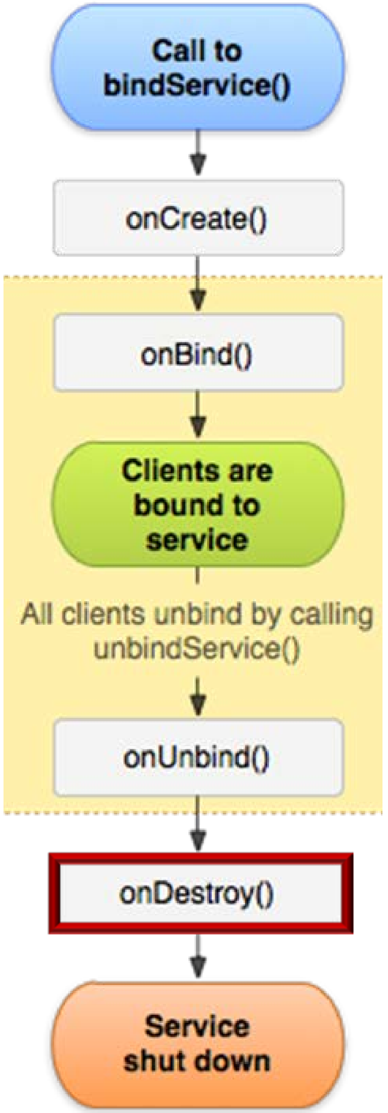
Stopping a Bound Service

- When a Bound Service is launched, it has a lifecycle that depends on the component(s) accessing it
  - i.e., it typically doesn't run in the background indefinitely, but instead is managed automatically

Cleanup any resources that were allocated

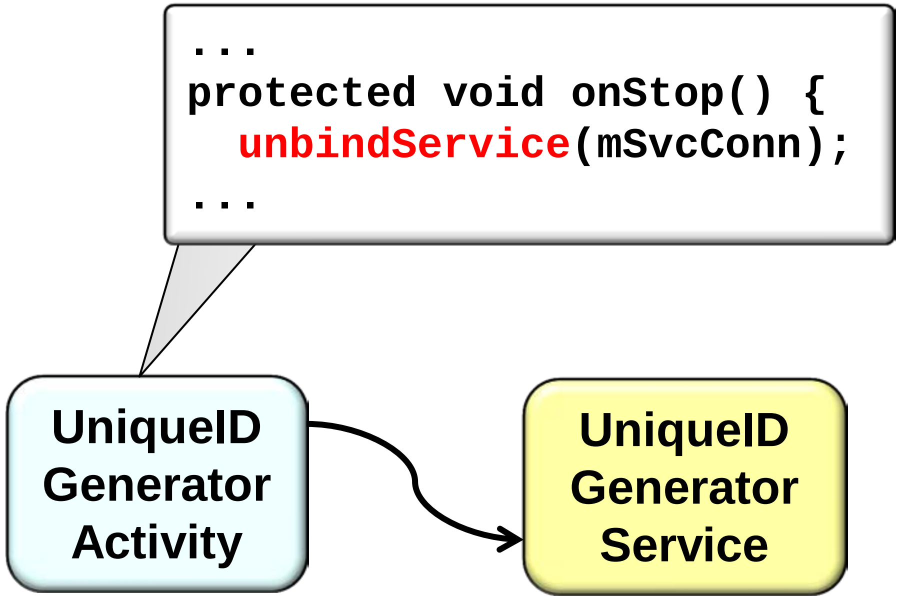
```
...
public void onDestroy() {
    mExecutor.shutdown();
}
...
```



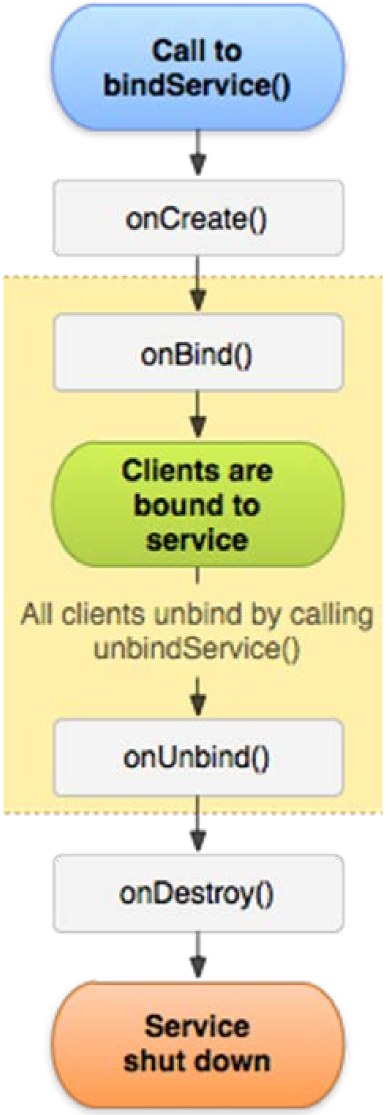


Stopping a Bound Service

- When a Bound Service is launched, it has a lifecycle that depends on the component(s) accessing it
- An Activity should call unbindService() when it stops or when it's done interacting with a Bound Service

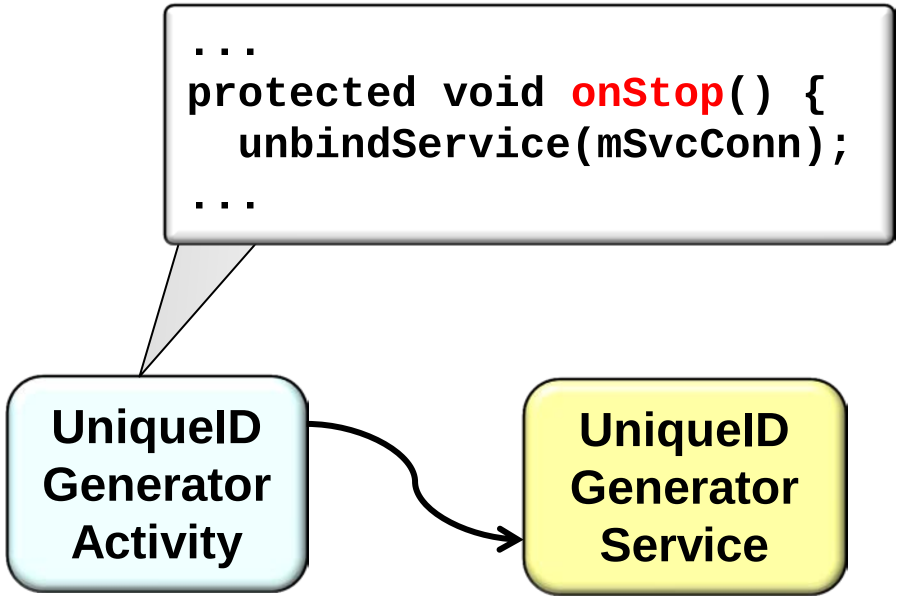


```
...
protected void onStop() {
  unbindService(mSvcConn);
...
```

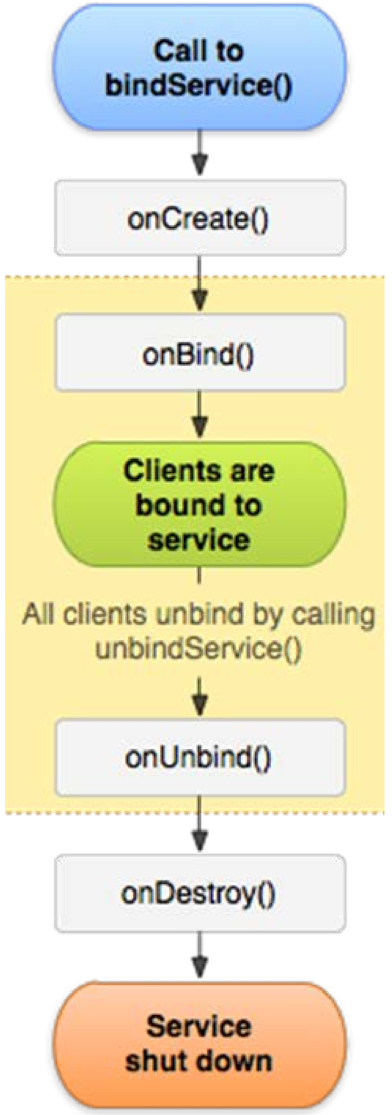


Stopping a Bound Service

- When a Bound Service is launched, it has a lifecycle that depends on the component(s) accessing it
- An Activity should call unbindService() when it stops or when it's done interacting with a Bound Service



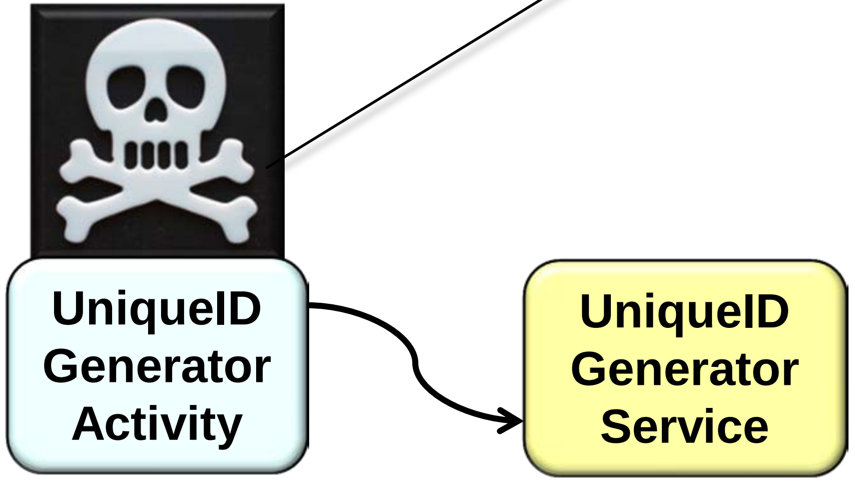
```
...
protected void onStop() {
  unbindService(mSvcConn);
...
```

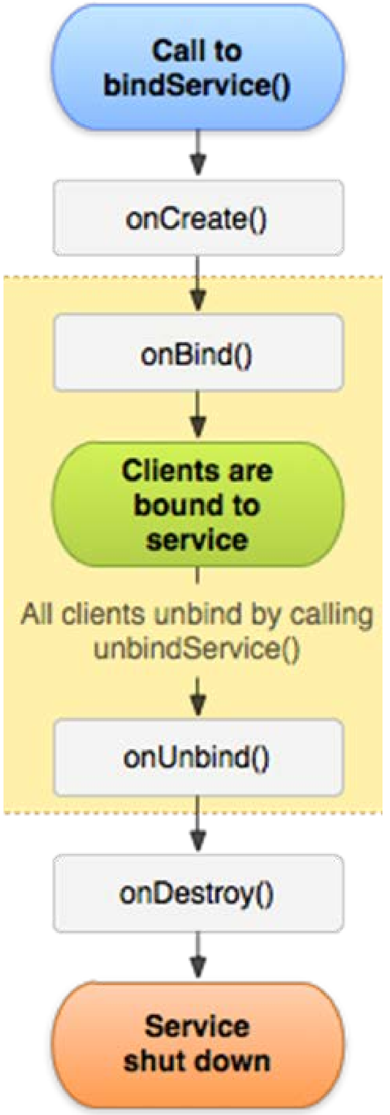


Stopping a Bound Service

- When a Bound Service is launched, it has a lifecycle that depends on the component(s) accessing it
- An Activity should call unbindService() when it stops or when it's done interacting with a Bound Service

When a client Activity is destroyed Android automatically unbinds it from any Bound Service it's connected to

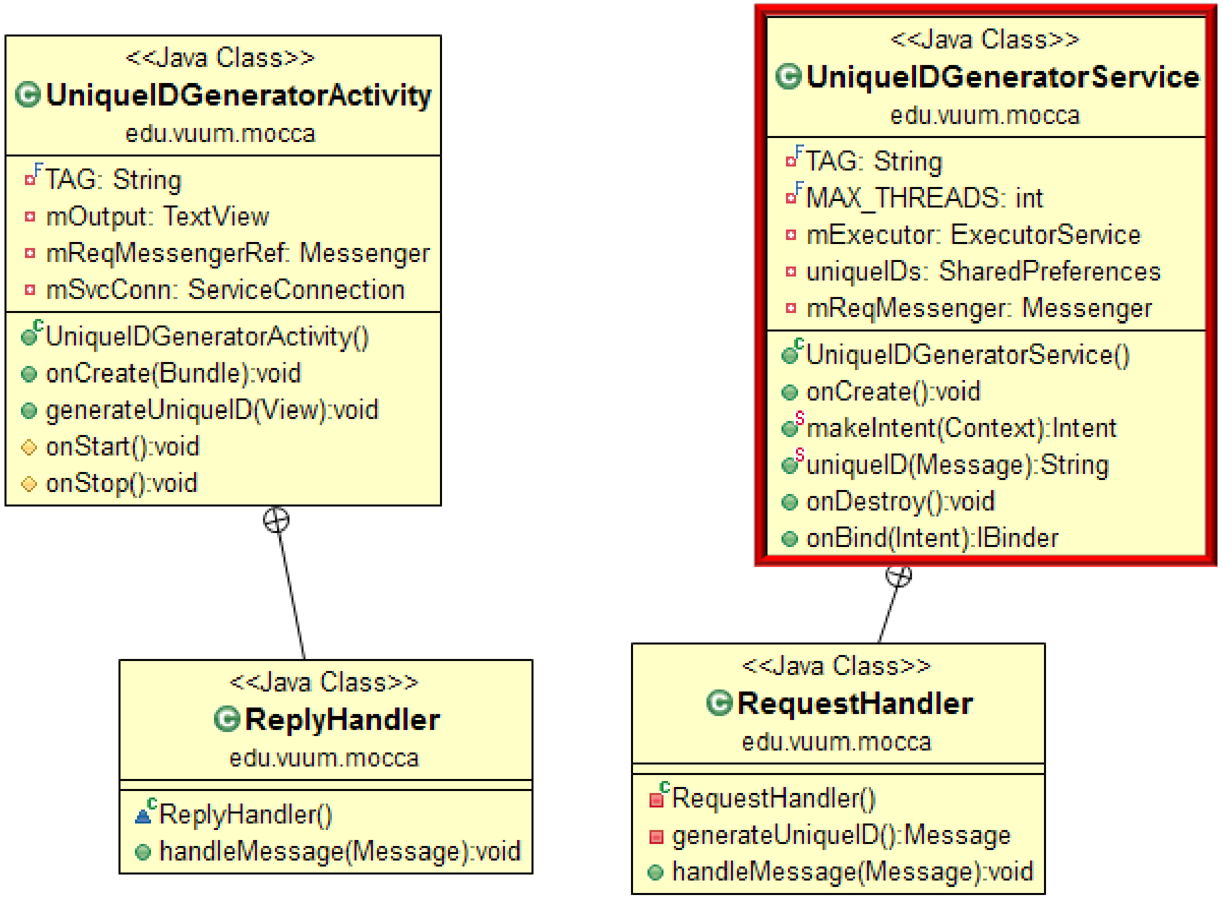


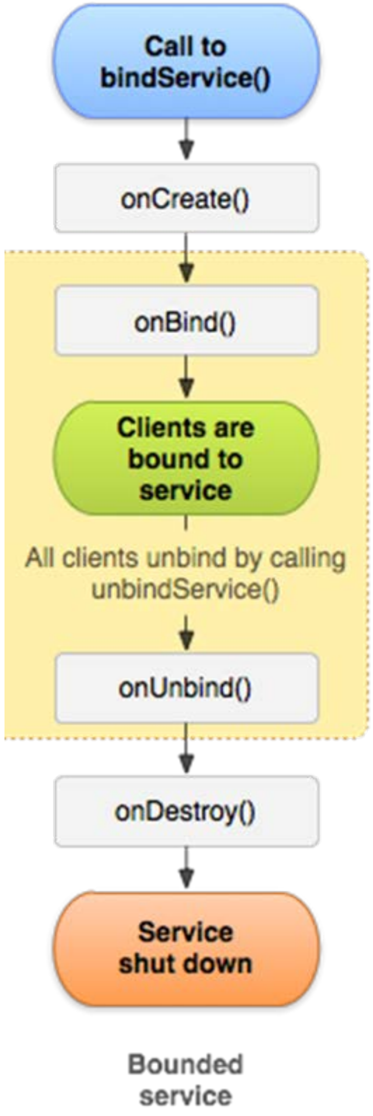


Summary

Summary

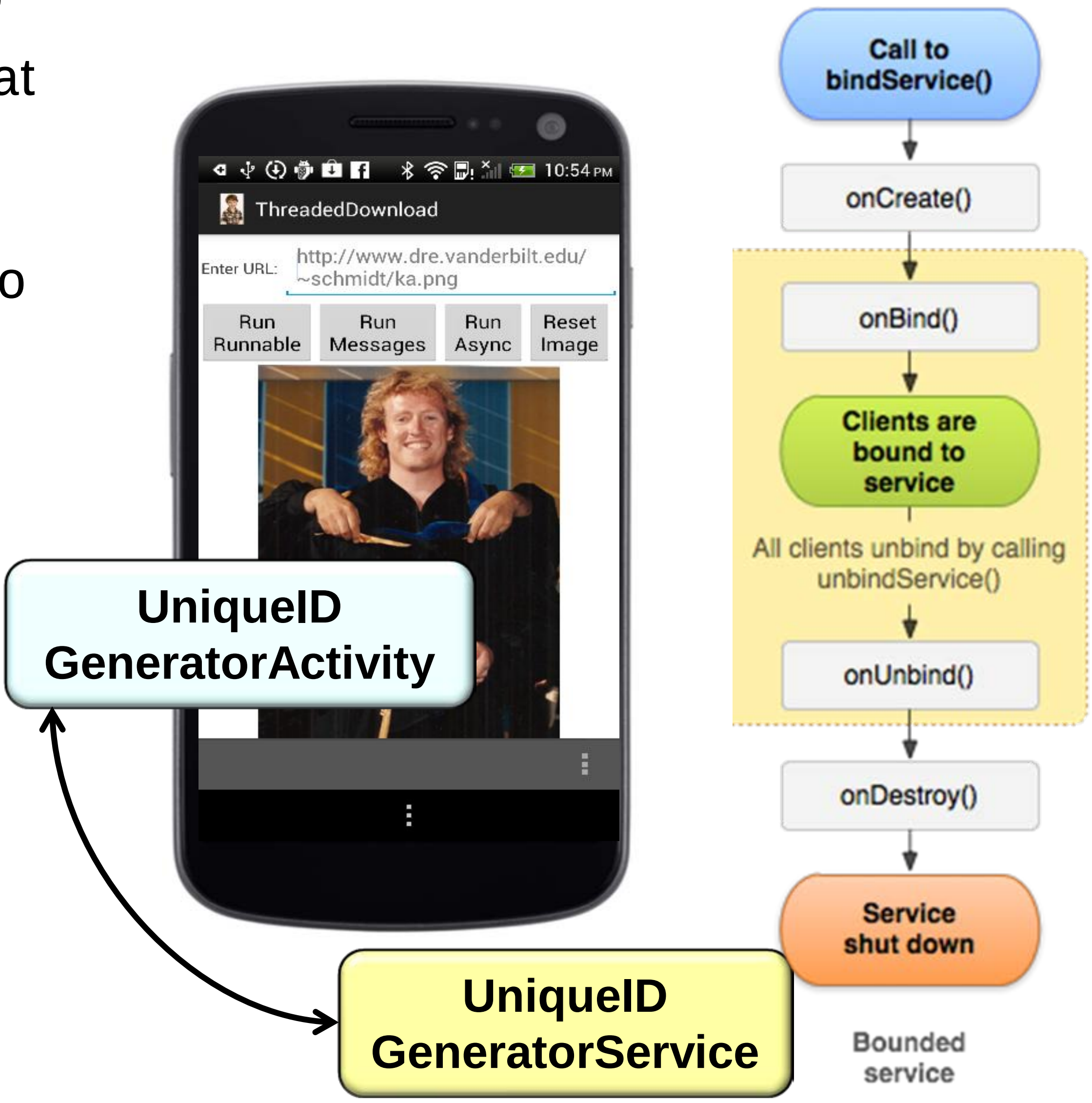
- A Bound Service is the "server" in a client-server interaction that runs on an Android device





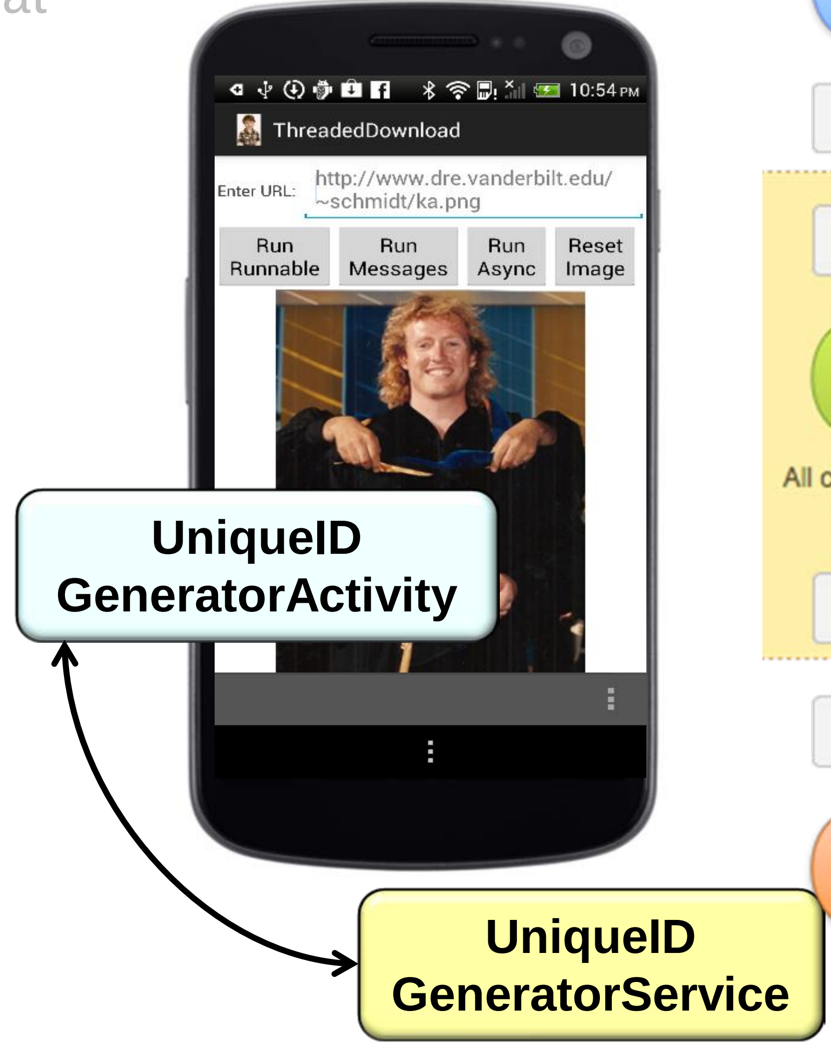
Summary

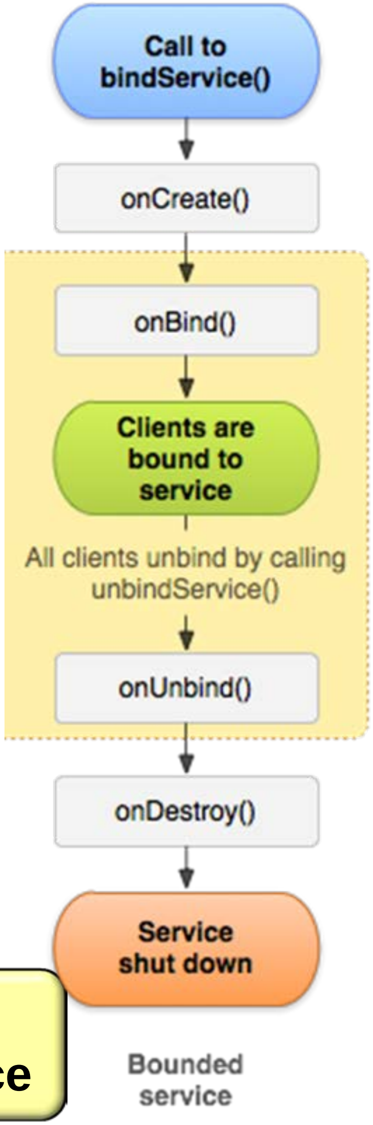
- A Bound Service is the "server" in a client-server interaction that runs on an Android device
  - e.g., the UniqueIDGenerator Application allows Activities to bind to the Service, send requests, receive replies, & perform IPC



Summary

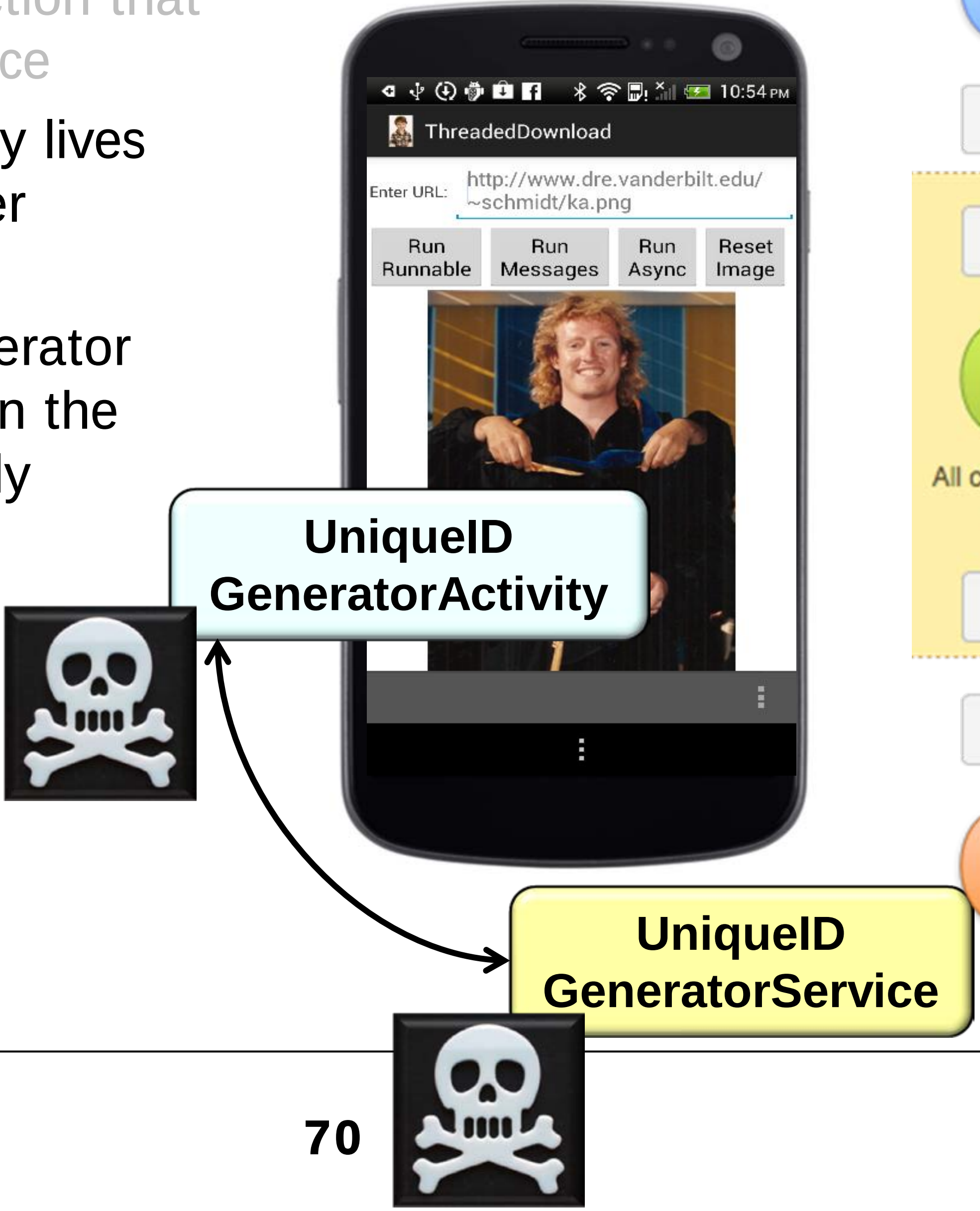
- A Bound Service is the "server" in a client-server interaction that runs on an Android device
- A Bound Service typically lives only while it serves other Application components

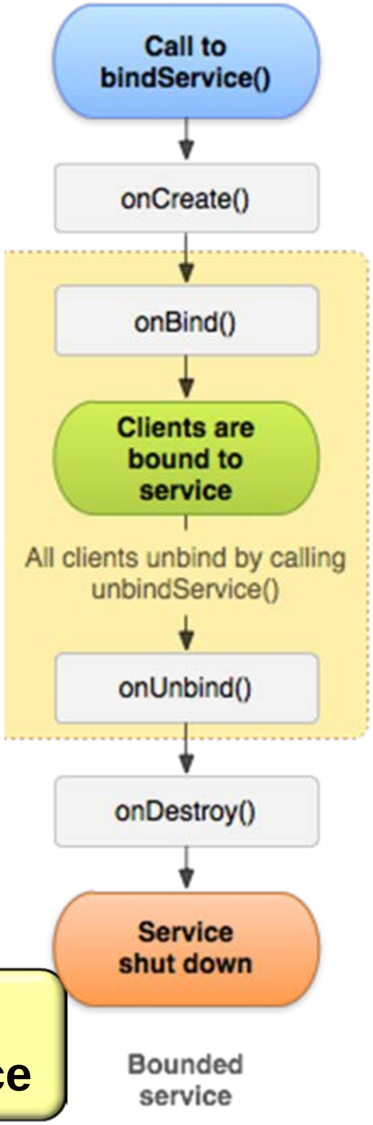

ThreadedDownload
Enter URL:
http://www.dre.vanderbilt.edu/~schmidt/ka.png
Run Runnable
Run Messages
Run Async
Reset Image
10:54 PM
UniqueID GeneratorActivity
UniqueID GeneratorService



Summary

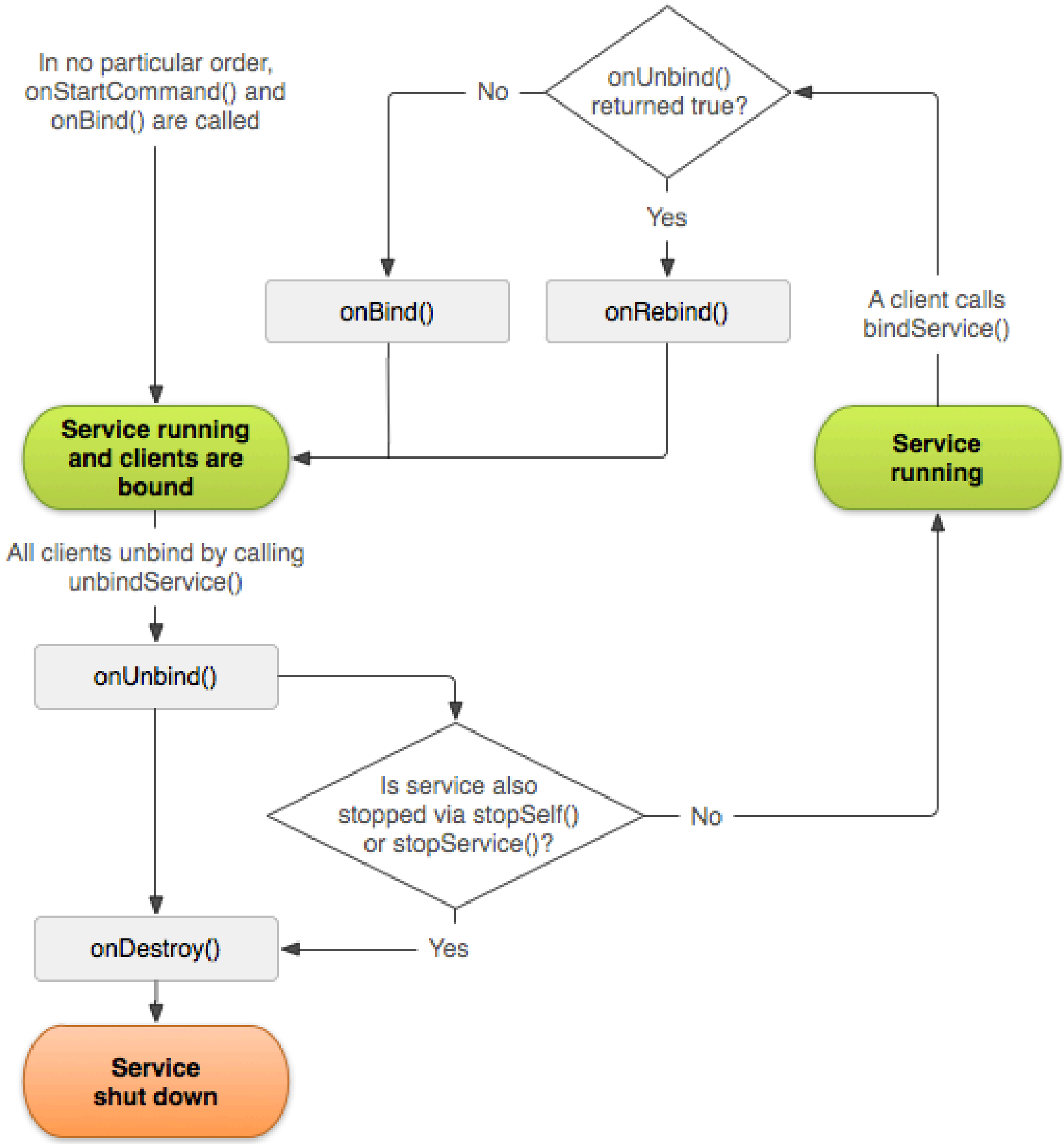
- A Bound Service is the "server" in a client-server interaction that runs on an Android device
- A Bound Service typically lives only while it serves other Application components
  - i.e., the UniqueIDGenerator Service does not run in the background indefinitely


10:54 PM
ThreadedDownload
Enter URL:
http://www.dre.vanderbilt.edu/~schmidt/ka.png
Run Runnable
Run Messages
Run Async
Reset Image
UniqueID GeneratorActivity
UniqueID GeneratorService



Summary

- A Bound Service is the “server” in a client-server interaction that runs on an Android device
- A Bound Service typically lives only while it serves other Application components
- It’s also possible to define “hybrid” models that combine Bound & Started Services

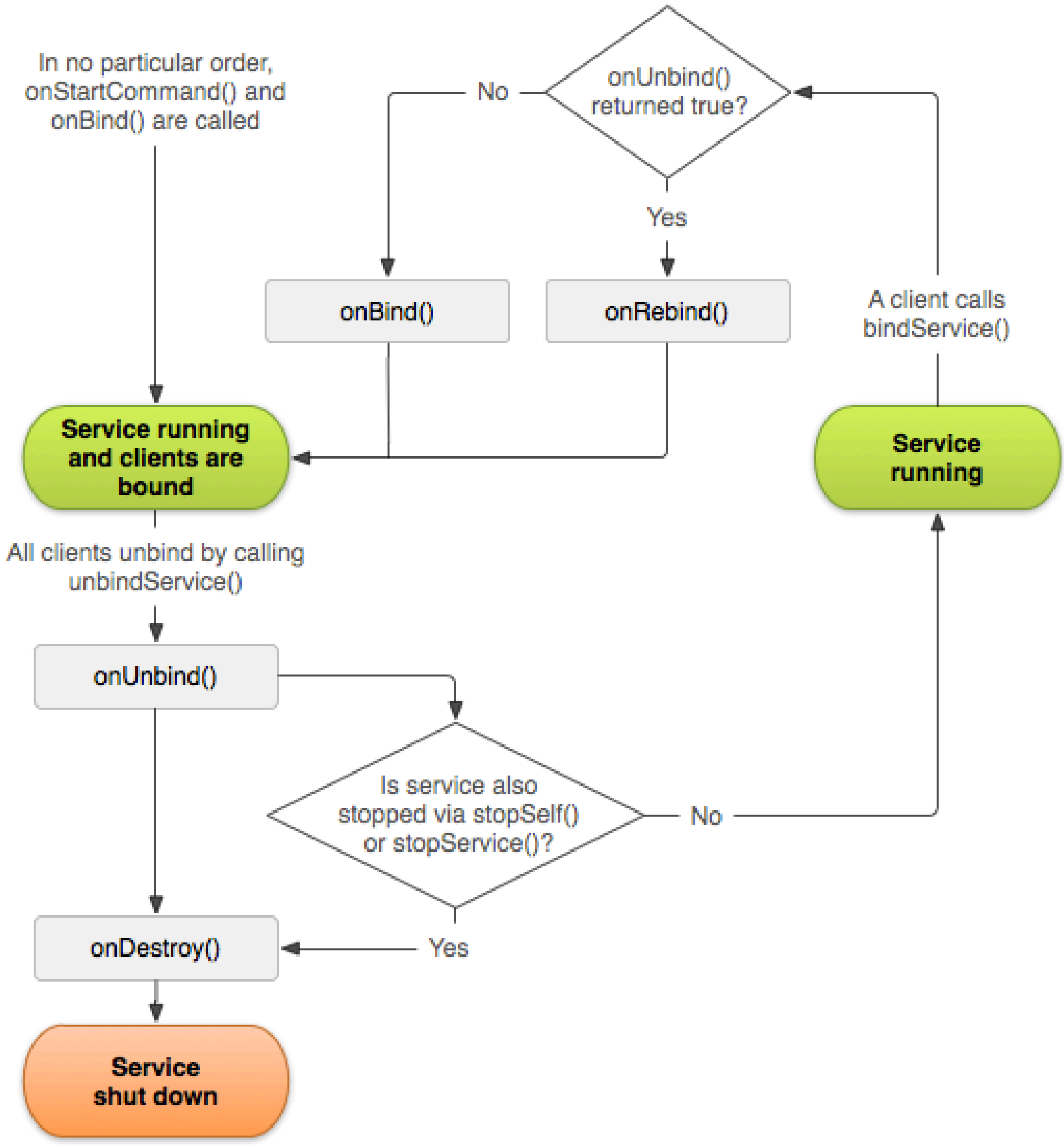


In no particular order, onStartCommand() and onBind() are called
onUnbind() returned true?
No
Yes
onBind()
onRebind()
A client calls bindService()
Service running and clients are bound
Service running
All clients unbind by calling unbindService()
onUnbind()
Is service also stopped via stopSelf() or stopService()?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

developer.android.com/guide/components/bound-services.html#Lifecycle

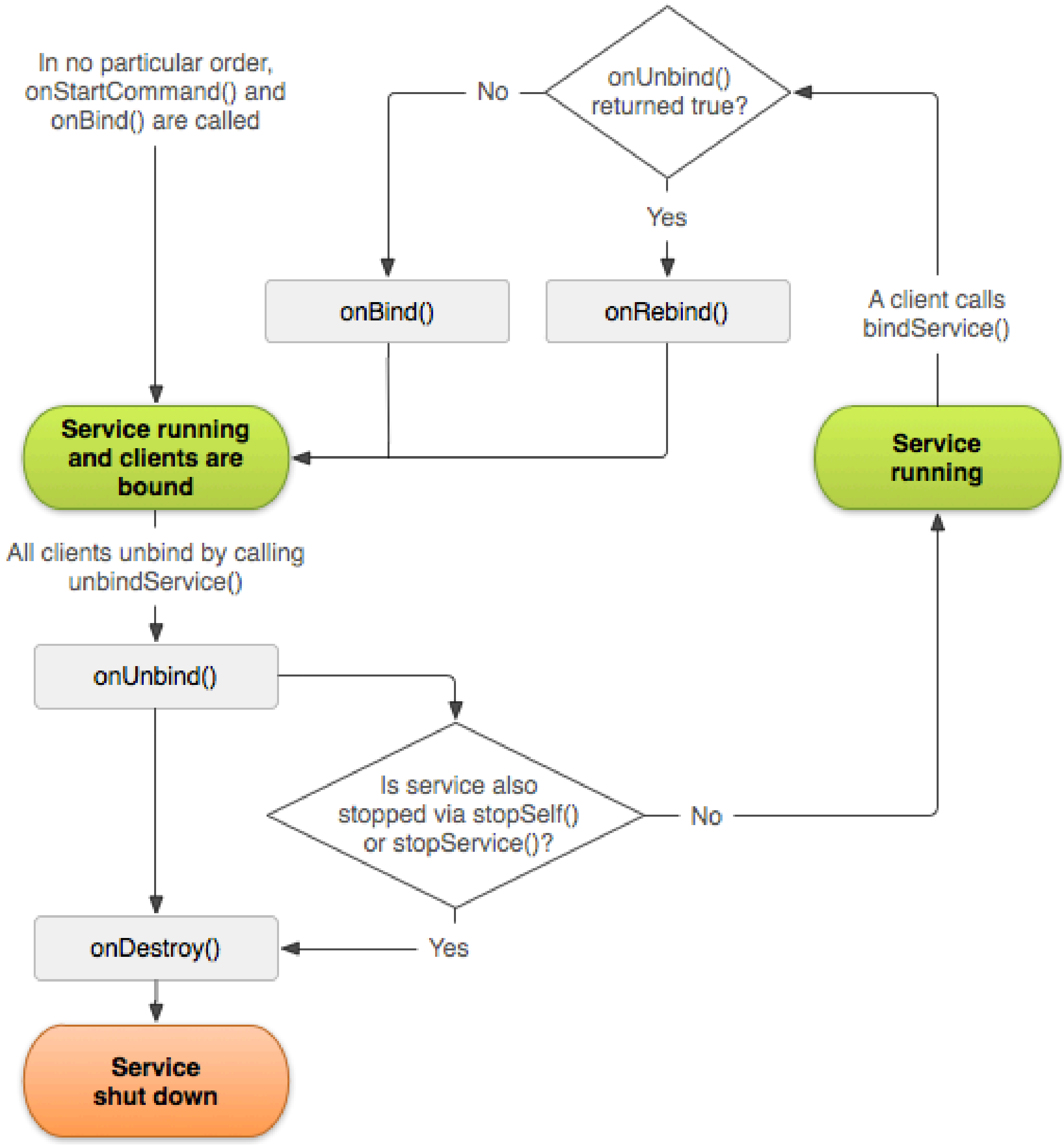
Summary

- A Bound Service is the "server" in a client-server interaction that runs on an Android device
- A Bound Service typically lives only while it serves other Application components
- It's also possible to define "hybrid" models that combine Bound & Started Services
  - If a Bound Service implements onStartCommand() it won't be destroyed when it's unbound from all clients


In no particular order, onStartCommand() and onBind() are called
No
onUnbind() returned true?
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

Summary

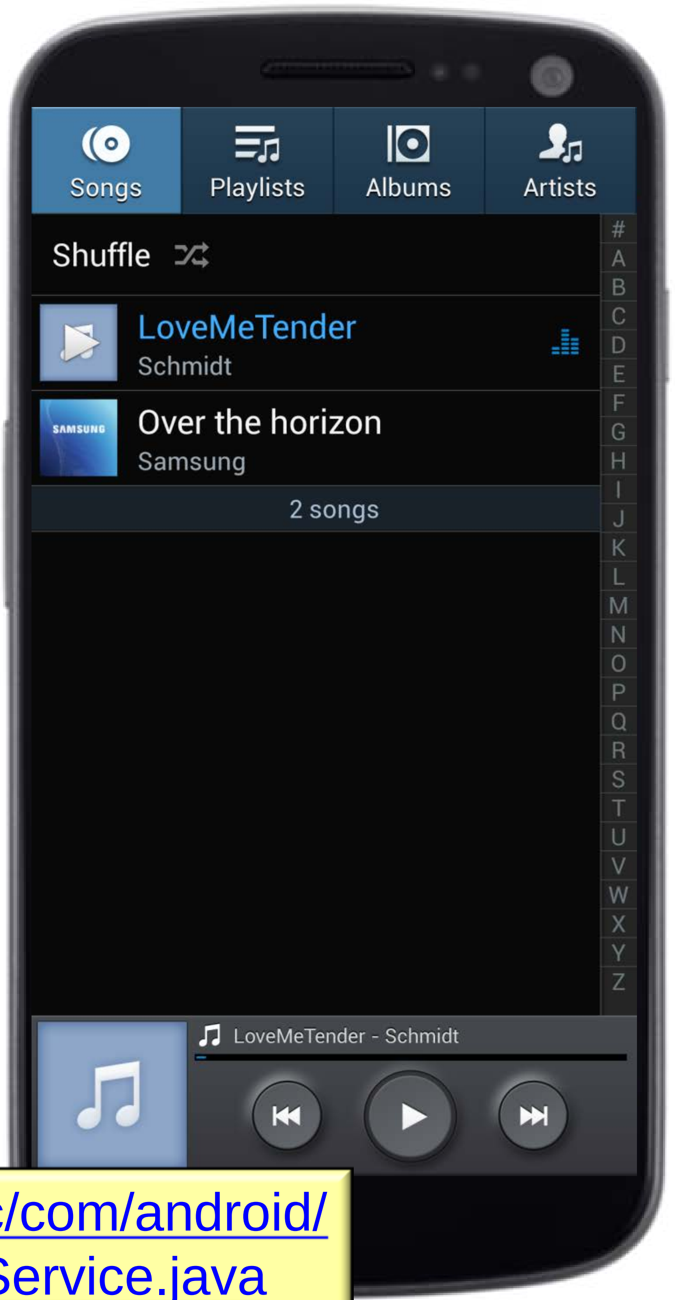
- A Bound Service is the "server" in a client-server interaction that runs on an Android device
- A Bound Service typically lives only while it serves other Application components
- It's also possible to define "hybrid" models that combine Bound & Started Services
  - If a Bound Service implements onStartCommand() it won't be destroyed when it is unbound from all clients
  - If onUnbind() returns "true" the onRebind() hook method will be called the next time a client binds to the Service


In no particular order, onStartCommand() and onBind() are called
No
onUnbind() returned true?
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
onDestroy()
Yes
Service shut down

[developer.android.com/reference/android/app/Service.html #onRebind(android.content.Intent)](developer.android.com/reference/android/app/Service.html#onRebind(android.content.Intent))

Summary

- A Bound Service is the "server" in a client-server interaction that runs on an Android device
- A Bound Service typically lives only while it serves other Application components
- It's also possible to define "hybrid" models that combine Bound & Started Services
- Android's MusicPlaybackService is an example of a hybrid Service



packages/apps/Music/src/com/android/music/MusicPlaybackService.java