Service in Android

EAST WEST UNIVERSITY

excellence in education

Department of Computer Science & Engineering

What is Service?

- One of android application components - Activities, **Services**, Content Providers, Broadcast Receivers

- Usually used to do long-term background work
  - Perform long-running processes without user intervention
  - Have no User Interface

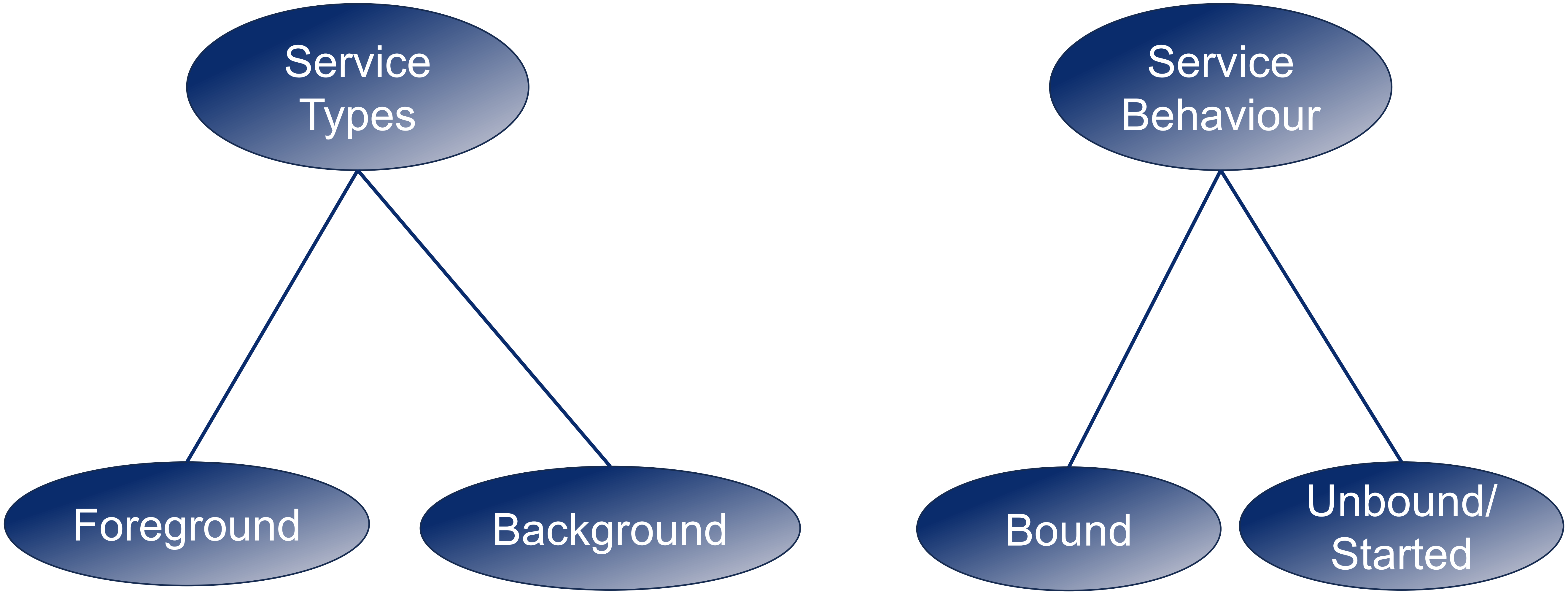
- Can be connected to other components and do inter-process communication (IPC)

- Can run in another process
  - **Service is initiated in UI Thread**

- Need to be declared in AndroidManefest.xml
  - Service can interactive with other component (exported = true)

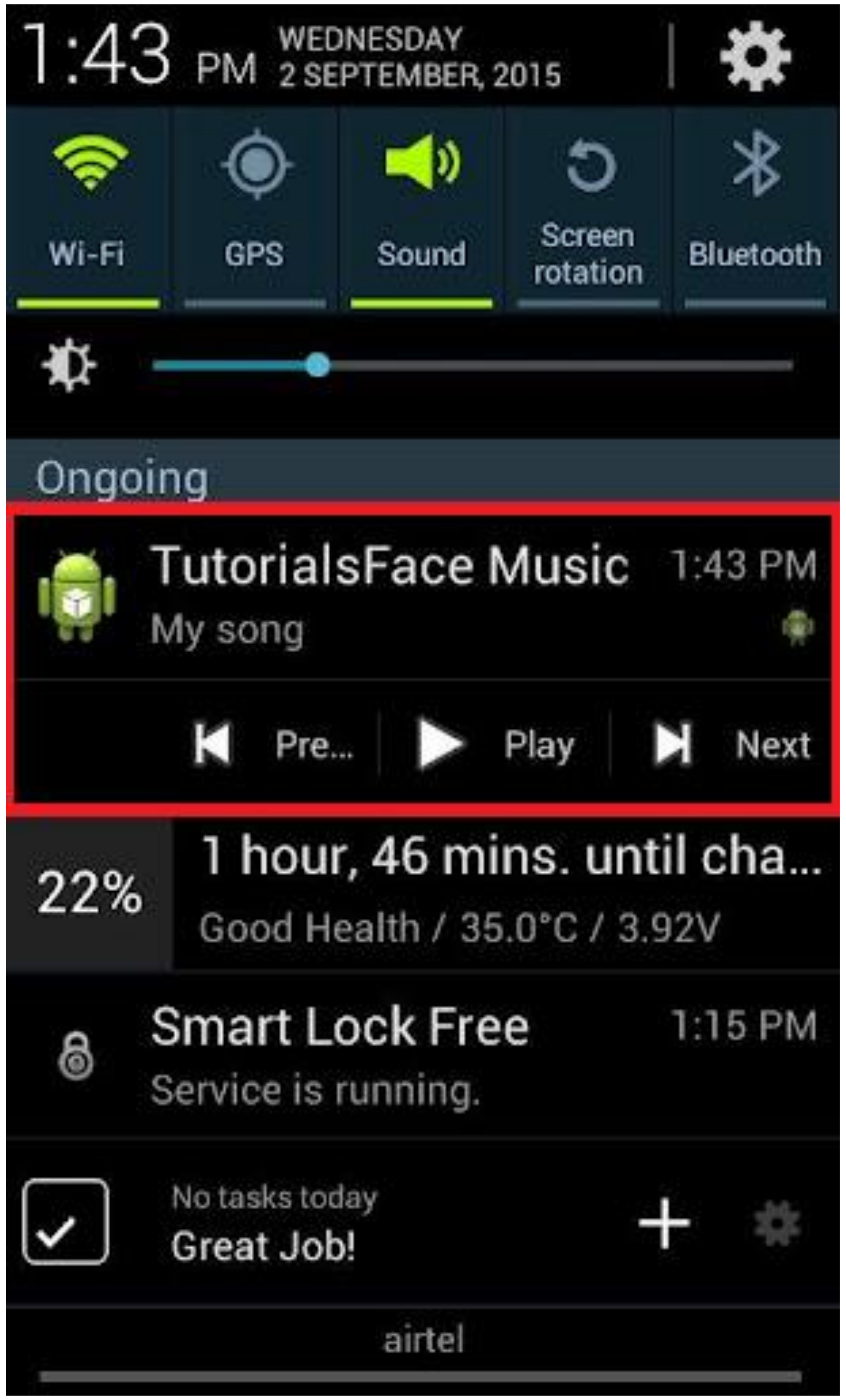
Department of Computer Science & Engineering

Android Services



Foreground Services

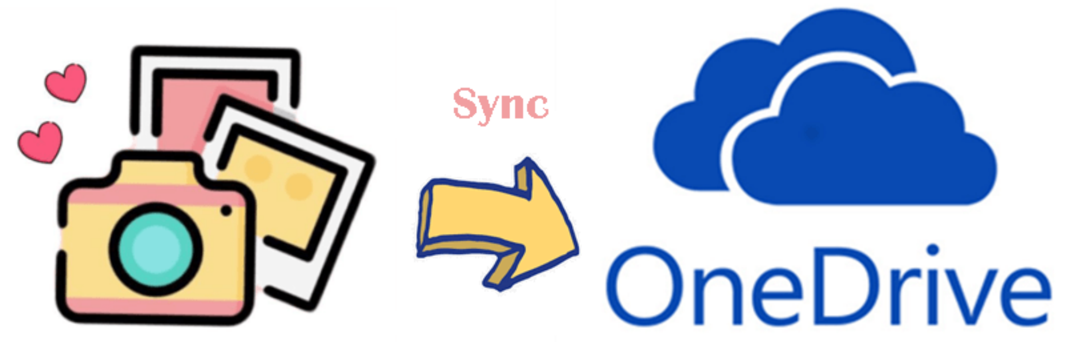
- Foreground services are those services whose **ongoing tasks** are visible to the users
  - The users can interact with them at ease and track what's happening via **Intent**
- These services continue to run even when users are using other applications
- Examples – Music Player and Downloading



Background Services

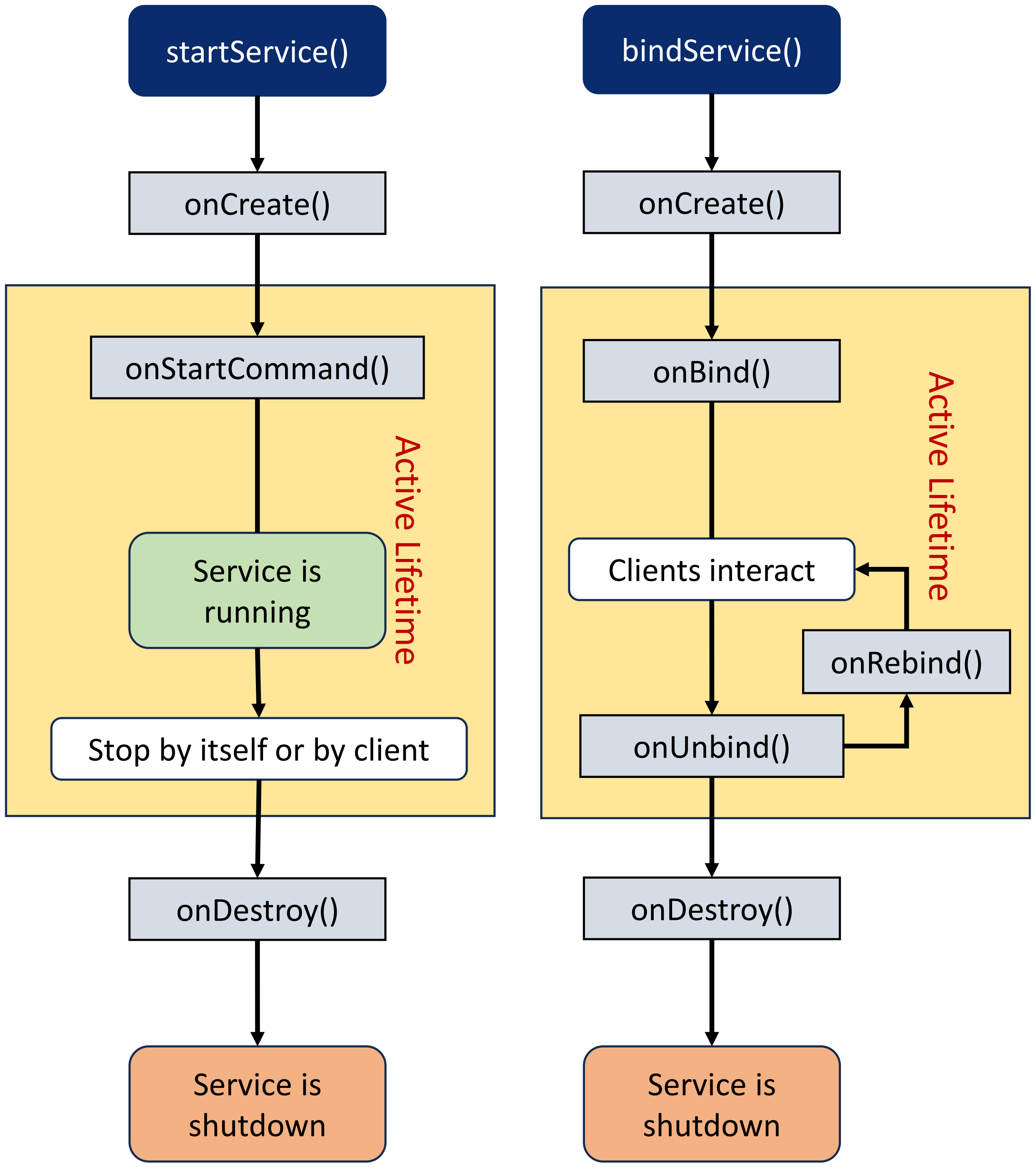
- These services run in the background, such that the user can’t see or access them
- These are the tasks that don’t need the user to know them
- Examples – Syncing and Storing data

SYNC
Local Folder
Google Drive Folder



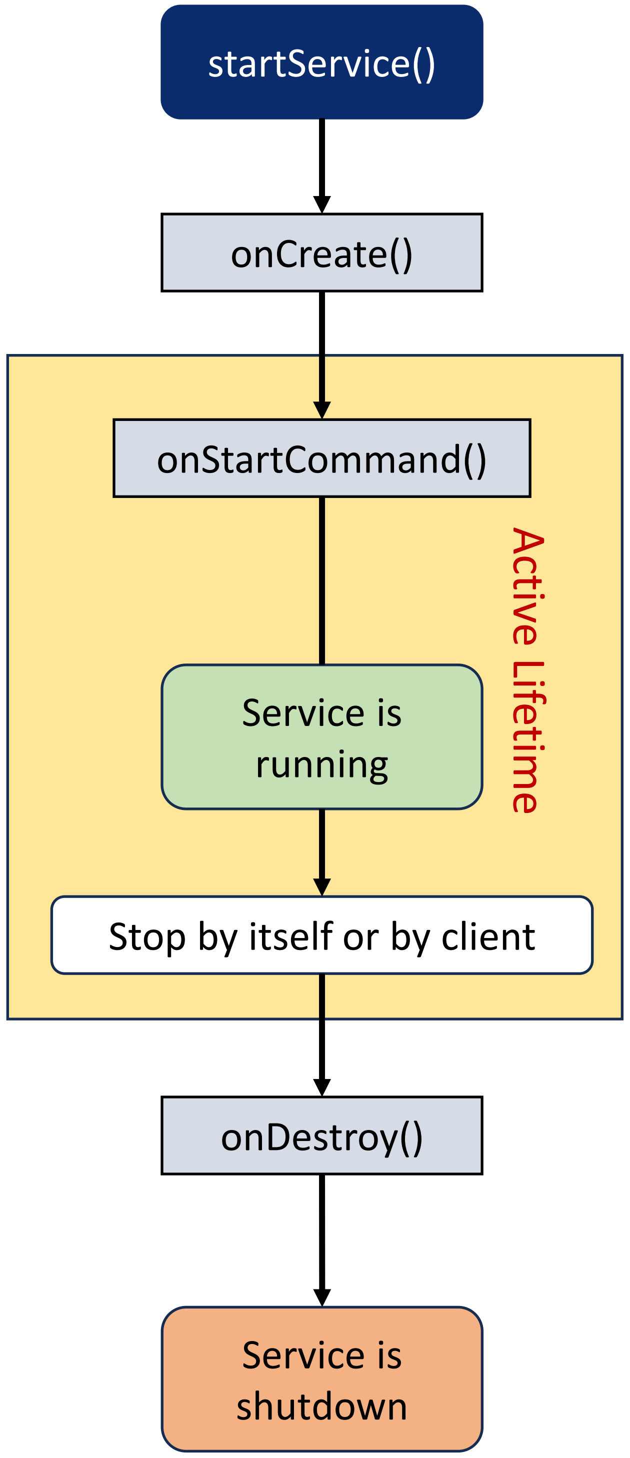
Lifecycle of Android Services

- Android services life-cycle can have two forms of services and they follow two paths, that are:
  - Started Service
    - Also known as Unbound
  - Bound Service

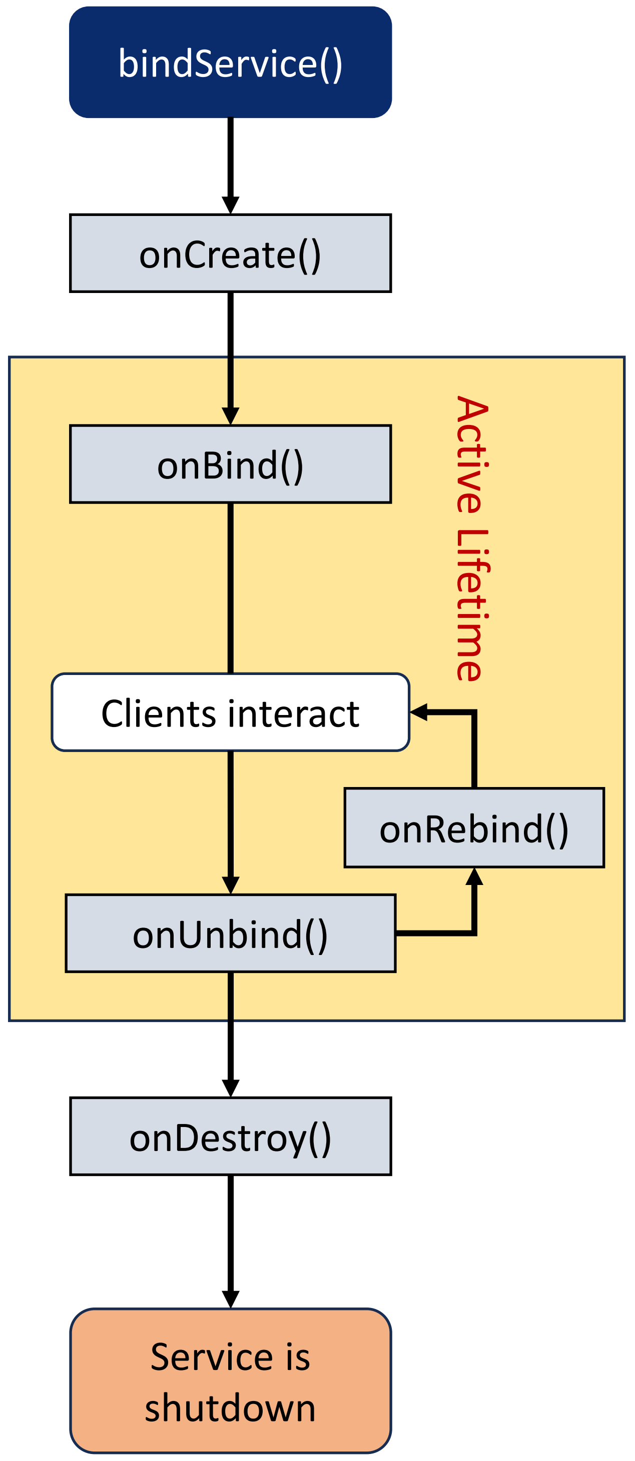


Started/Unbound Service

- Started only when an application component calls startService()
- Performs a single operation and doesn't return any result to the caller
- Once starts, runs in the background even if the component that created it destroys
- Can be stopped only in one of the two cases:
  - By using the stopService() method.
  - By stopping itself using the stopSelf() method



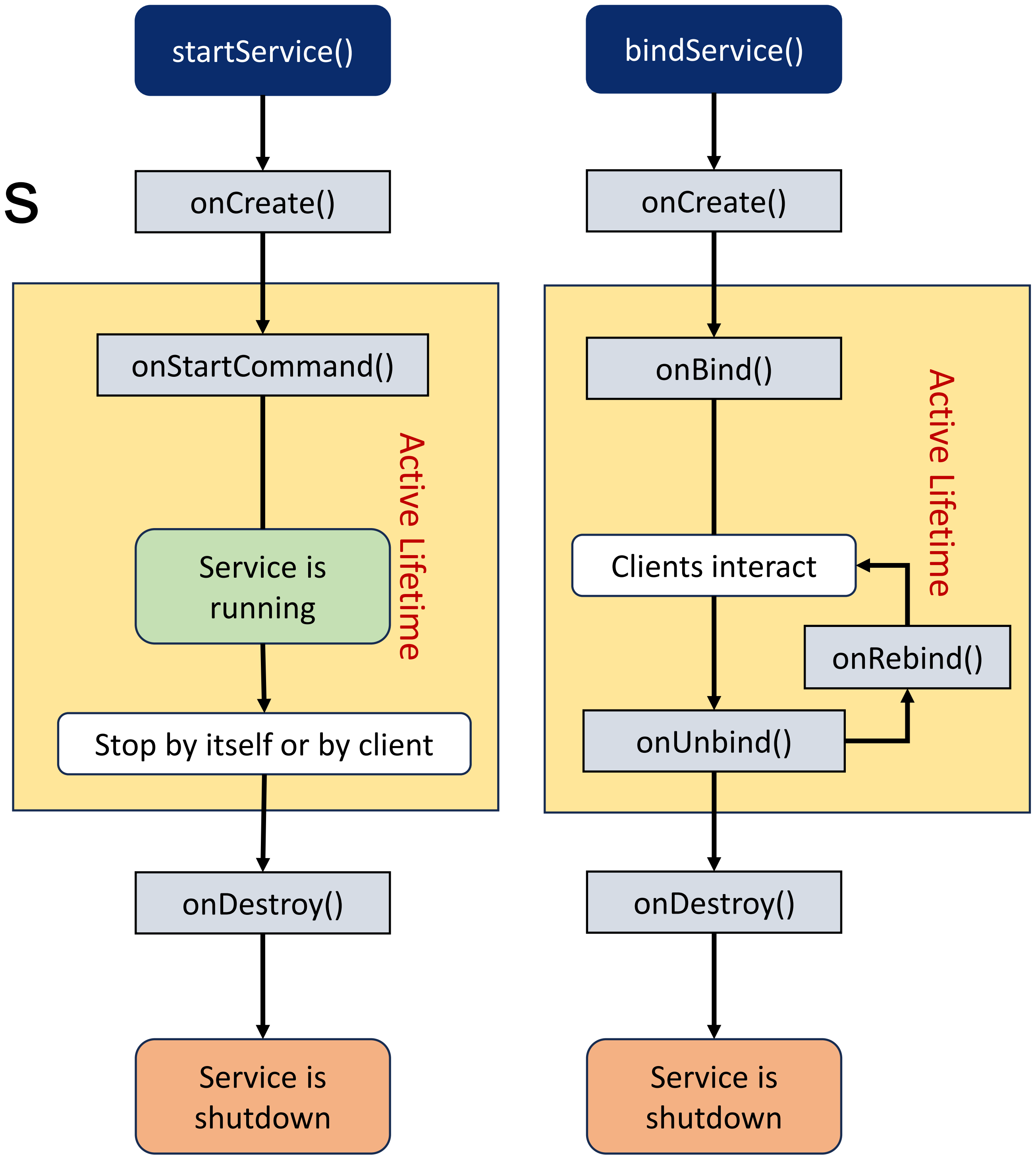
Bound Service



- A service is bound only if an application component binds to it using bindService()
- Gives a client-server relation that lets the components interact with the service
- Components can send requests to services and get results
- Runs in the background as long as another application is bound to it
- Or can be unbound according to our requirement by using the unbindService() method

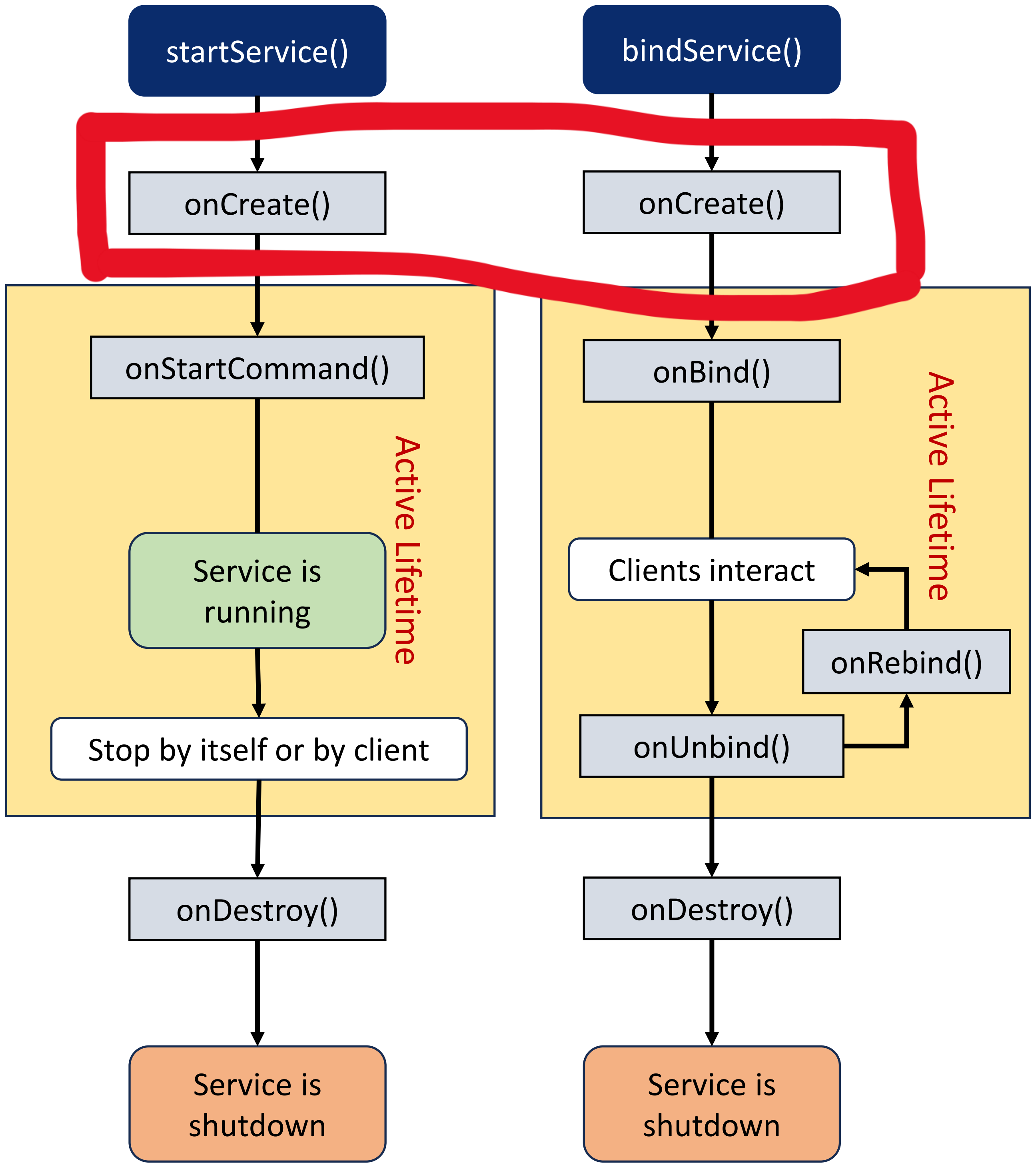
Methods of Android Services

- ❖ The service base class defines certain callback methods to perform operations on applications
- ❖ The following are a few important methods of Android services :
  - onCreate()
  - onStartCommand()
  - onBind()
  - onUnbind()
  - onRebind()
  - onDestroy()



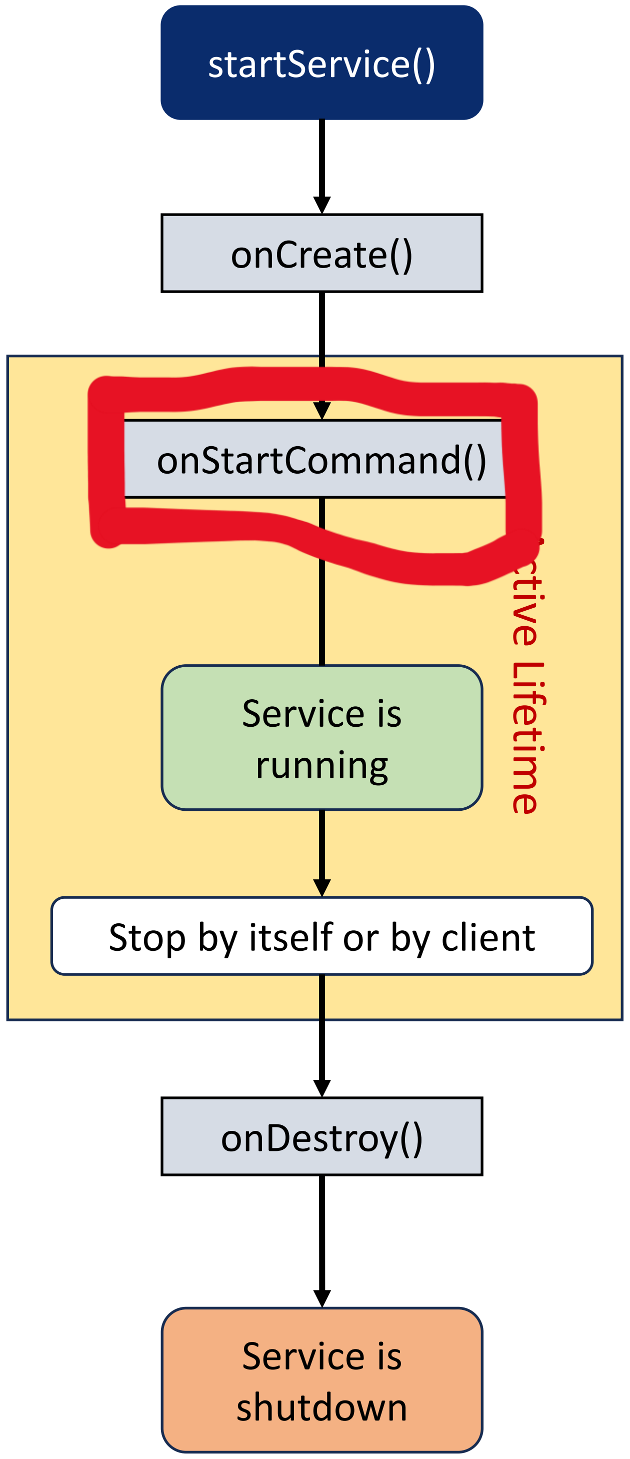
onCreate()

- First method that the system calls when a new component starts the service
- We need this method for a one-time set-up

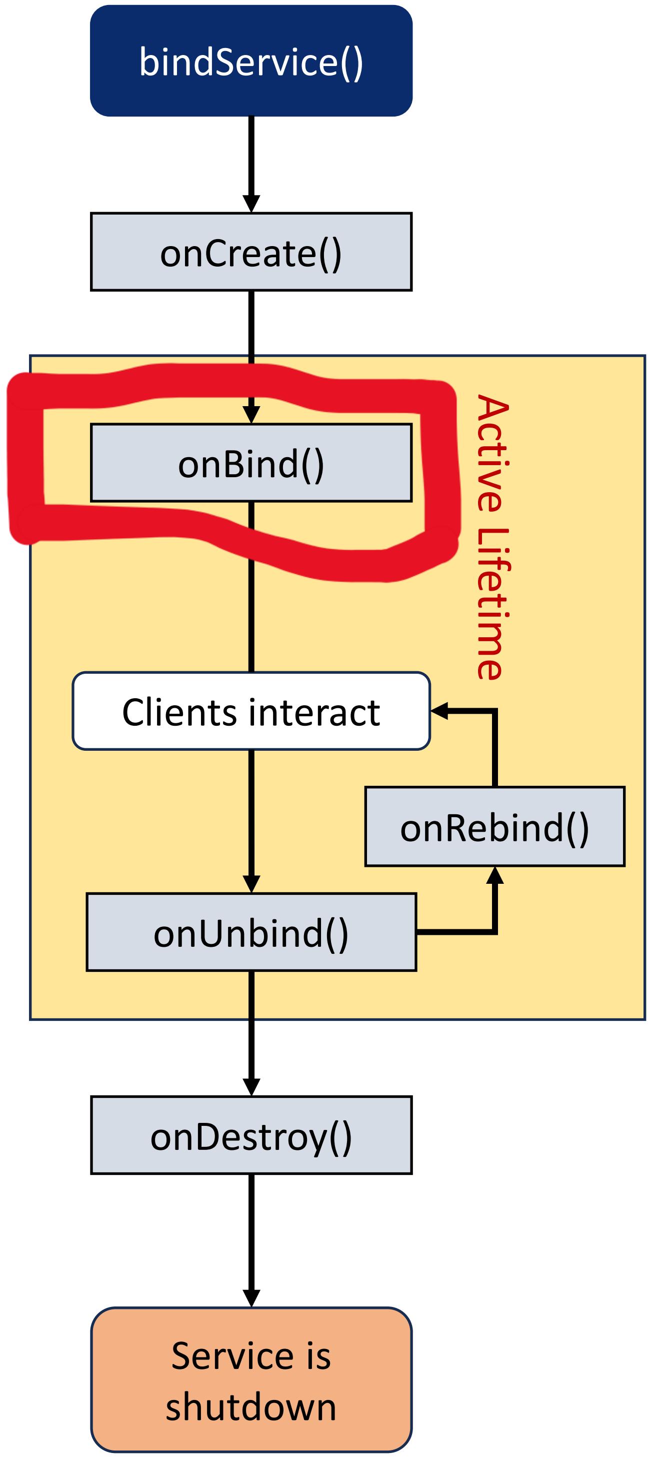


onStartCommand()

- The system calls this method whenever a component, i.e. an activity, requests 'start' to a service, using startService()
- Once we use this method it's our duty to stop the service using stopService() or stopSelf()
- The return value define the behaviour of service
  - return START_STICKY
    - Restart by system if killed
    - Why/How does the system kill a service?
  - return START_NOT_STICKY
    - Doesn't restart if killed
  - return START_REDELIVER_INTENT
    - Restart if system crashed

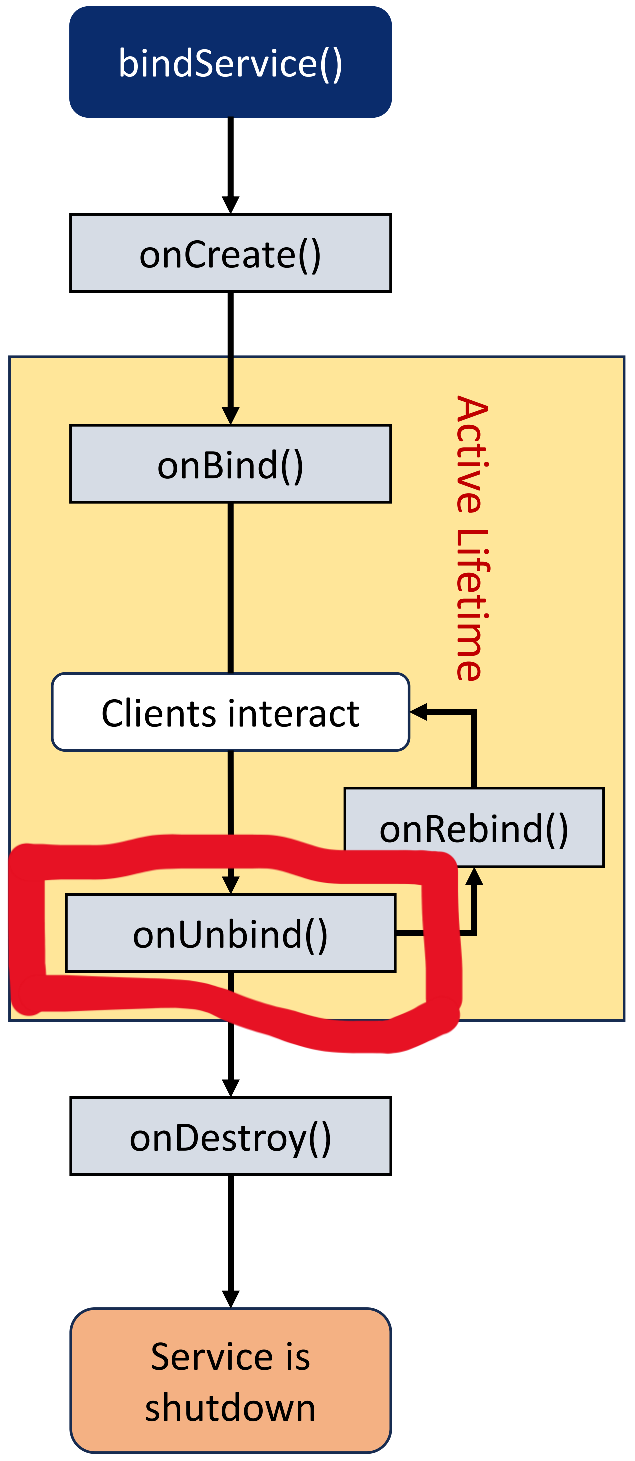


onBind()



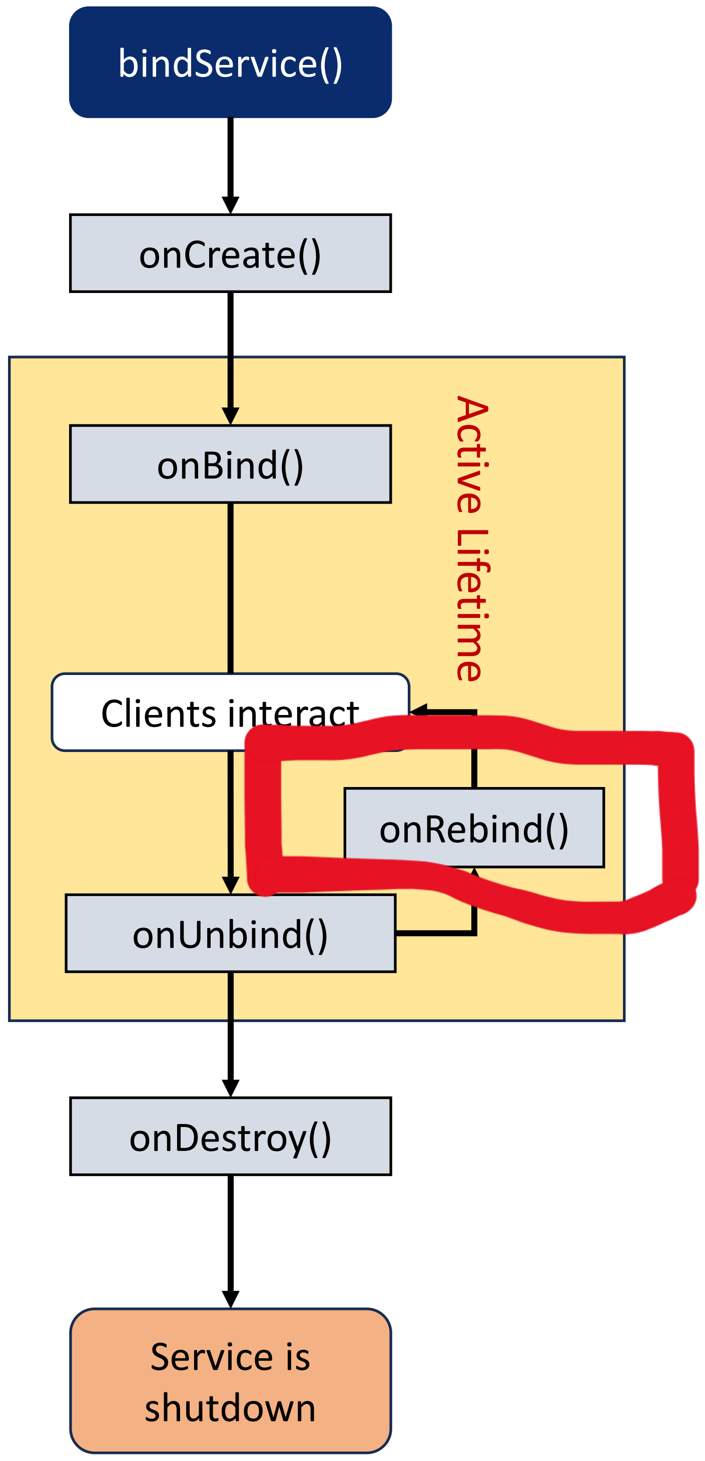
- ❖ Invoked when a component wants to bind with the service by calling bindService()
- ❖ In this, we must provide an interface for clients to communicate with the service
  - For inter-process communication, we can use the IBinder object
- ❖ It is a must to implement this method if bindService() is invoked
  - If in case binding is not required, we should return null as implementation is mandatory.

onUnbind()



- The system invokes this when all the clients disconnect from the interface published by the service

onRebind()

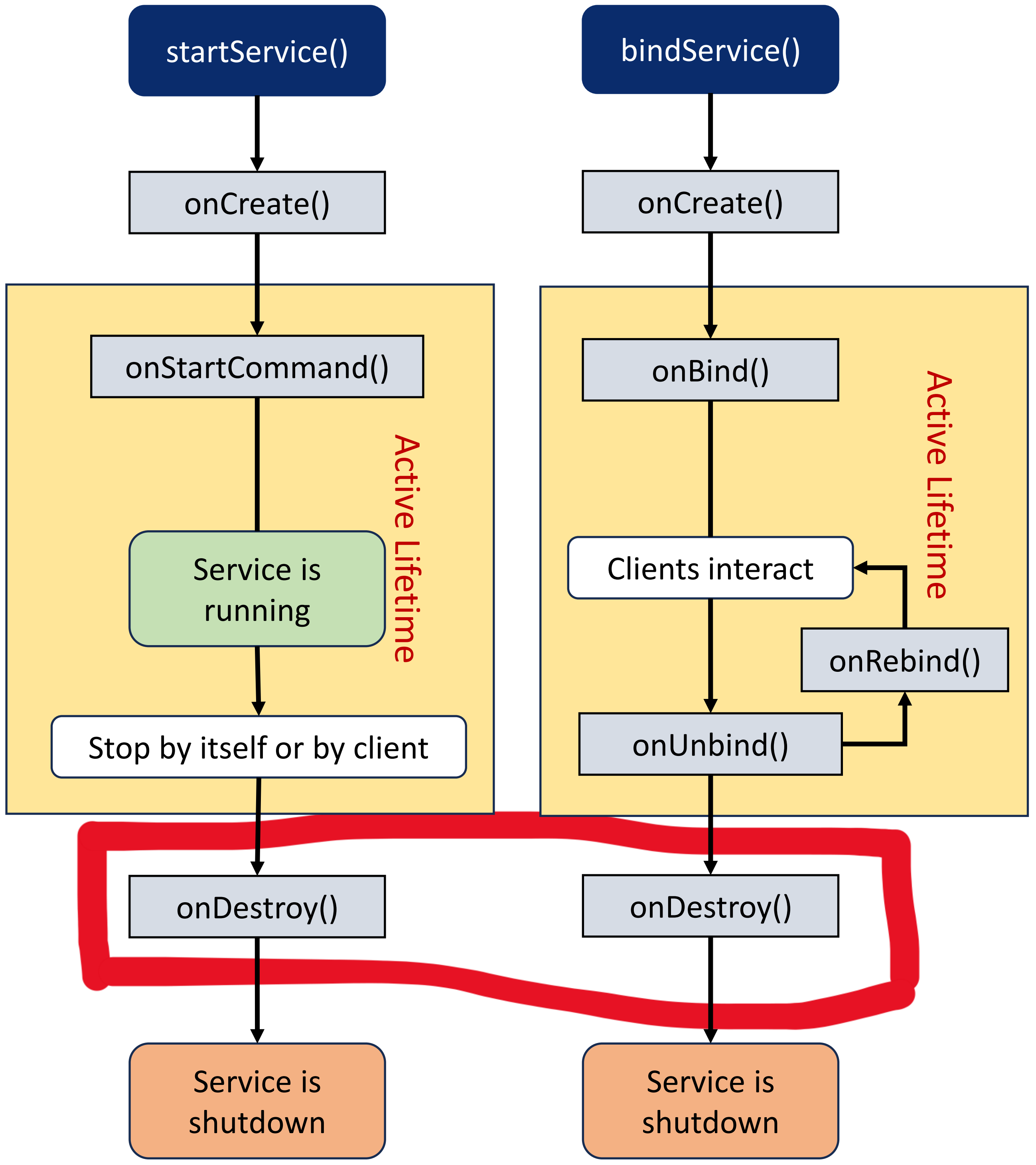


- The system calls this method when new clients connect to the service as long as bound service is running

- The system calls it after the onBind() method

onDestroy()

- The final clean up call for the system
- The system invokes it just before the service destroys
- It cleans up resources like threads, receivers, registered listeners, etc.



Example: Started/Unbound Service

```java
public class UnboundService extends Service {
    @Override
    public void onCreare() {
        // one time execution
        super.onCreate();
        // ... here add code for initialization or others
    }
    @Override
    public int onStartCommand(Intent i, int flags, int startId) {
        // This will execute every time when startService() called by the client
        // service is active now
        // ... add code for any processing here
        return START_STICKY; // other choices: START_NOT_STICKY, START_REDELIVER_INTENT
    }
    @Override
    public IBinder onBind(Intent i) {
        // This will never be invoked if startService() called by the client
        throw new UnsupportedOperationException("This service cannot be bound");
    }
    @Override
    public void onDestroy(){
        // ... code to do something before destroying the service process
        super.onDestroy();
    }
}
```

Example: Bound Service

```java
public class MyBoundService extends Service {

    // Binder given to clients
    private final IBinder binder = new LocalBinder();

    // Class used for the client Binder
    public class LocalBinder extends Binder {
        public MyBoundService getService() {
            return MyBoundService.this;
        }
    }

    @Override
    public void onCreate() {
        super.onCreate();
        // Called when the service is first created (only
once during its lifetime)
        System.out.println("Service: onCreate called");
    }

    @Override
    public IBinder onBind(Intent intent) {
        // Called when a client (Activity) binds to the
service by calling bindService()
        System.out.println("Service: onBind called");
        return binder;
    }

    // Public method the clients can call
    public String getFromService() {
        return "something from service";
    }
```

```java
    @Override
    public boolean onUnbind(Intent intent) {
        // Called when all clients have disconnected
from a particular interface published by the service
        // If we return true, onRebind() will be
called when a new client binds
        System.out.println("Service: onUnbind
called");
        return true;
    }

    @Override
    public void onRebind(Intent intent) {
        // Called when a client rebinds to the service
after it had been unbound (and we returned true in
onUnbind)
        System.out.println("Service: onRebind
called");
        super.onRebind(intent);
    }

    @Override
    public void onDestroy() {
        // Called when the service is no longer used
and is being destroyed
        System.out.println("Service: onDestroy
called");
        super.onDestroy();
    } // end of onDestroy()


} // end of service class
```

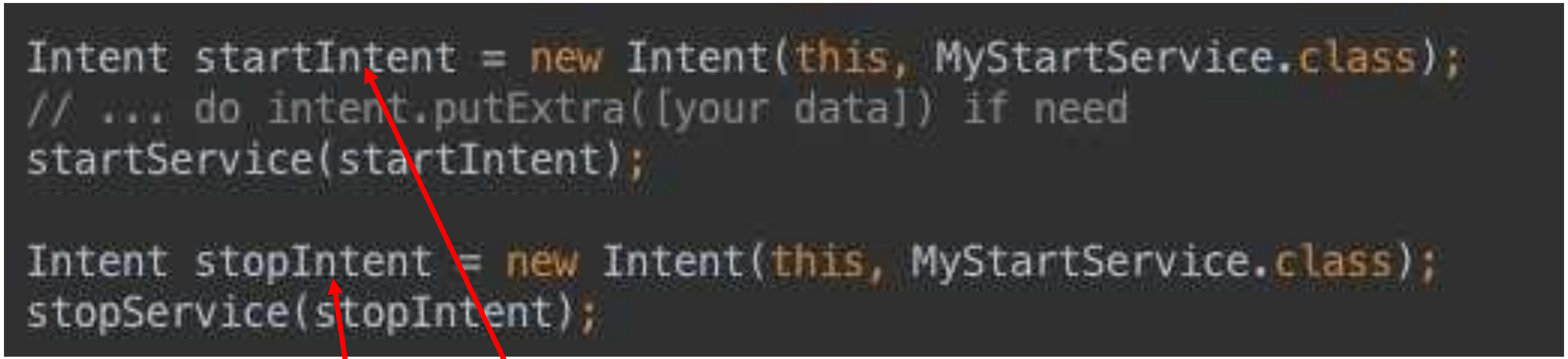
Example: Bound Service

```
public class MainActivity extends Activity {
    private MyBoundService myService;
    private boolean isBound = false;
    private TextView textView;
    private Button btnBind, btnUnbind, btnGetFromService;

    // Defines callbacks for service binding,
      // passed to bindService()
    private ServiceConnection connection = new
ServiceConnection() {

      public void onServiceConnected(ComponentName name,
IBinder service) {
         // Called when the connection with the service has
been established
         System.out.println("onServiceConnected");
         MyBoundService.LocalBinder binder =
(MyBoundService.LocalBinder) service;

          // Get service instance
         myService = binder.getService();
         isBound = true;
     }

     public void onServiceDisconnected(ComponentName name) {
       // Called when the connection with the service has
been unexpectedly disconnected – crashed or killed
       System.out.println("onServiceDisconnected");
       isBound = false;
     }
  };
```

```
  protected void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.activity_main);

     // Set up button click listeners
     btnBind.setOnClickListener(v->{
        // Called when user presses "Bind Service"
         // Bind to the service
         Intent intent = new Intent(this,
MyBoundService.class);
    bindService(intent, connection,
Context.BIND_AUTO_CREATE);
     });
     btnUnbind.setOnClickListener(v->{
       // Called when user presses "Unbind Service"
       unbindMyService();
     });
     btnGetFromService.setOnClickListener(v->{
       // Called when user presses "Get Time"
       if (isBound) {
          String str = myService.getFromService();
          textView.setText(str);
       }
     });
  }

  private void unbindMyService() {
     // Unbind from the service
     if (isBound) {
        unbindService(connection); isBound = false;
     }
  }

  protected void onDestroy() {
    super.onDestroy();
    // Good practice: unbind service to avoid leaks
    unbindMyService();
  }
} // end of activity class
```

	Bound Service	Started Service
How you start	bindService(intent, conn, flags)	startService(intent)
Needs ServiceConnection?	✅ Yes	❌ No
Direct method call?	✅ Yes (through the Binder)	❌ No (need other IPC methods)
Service lifespan	Exists only while bound	Lives until stopped manually or system kills it

Singleton Class

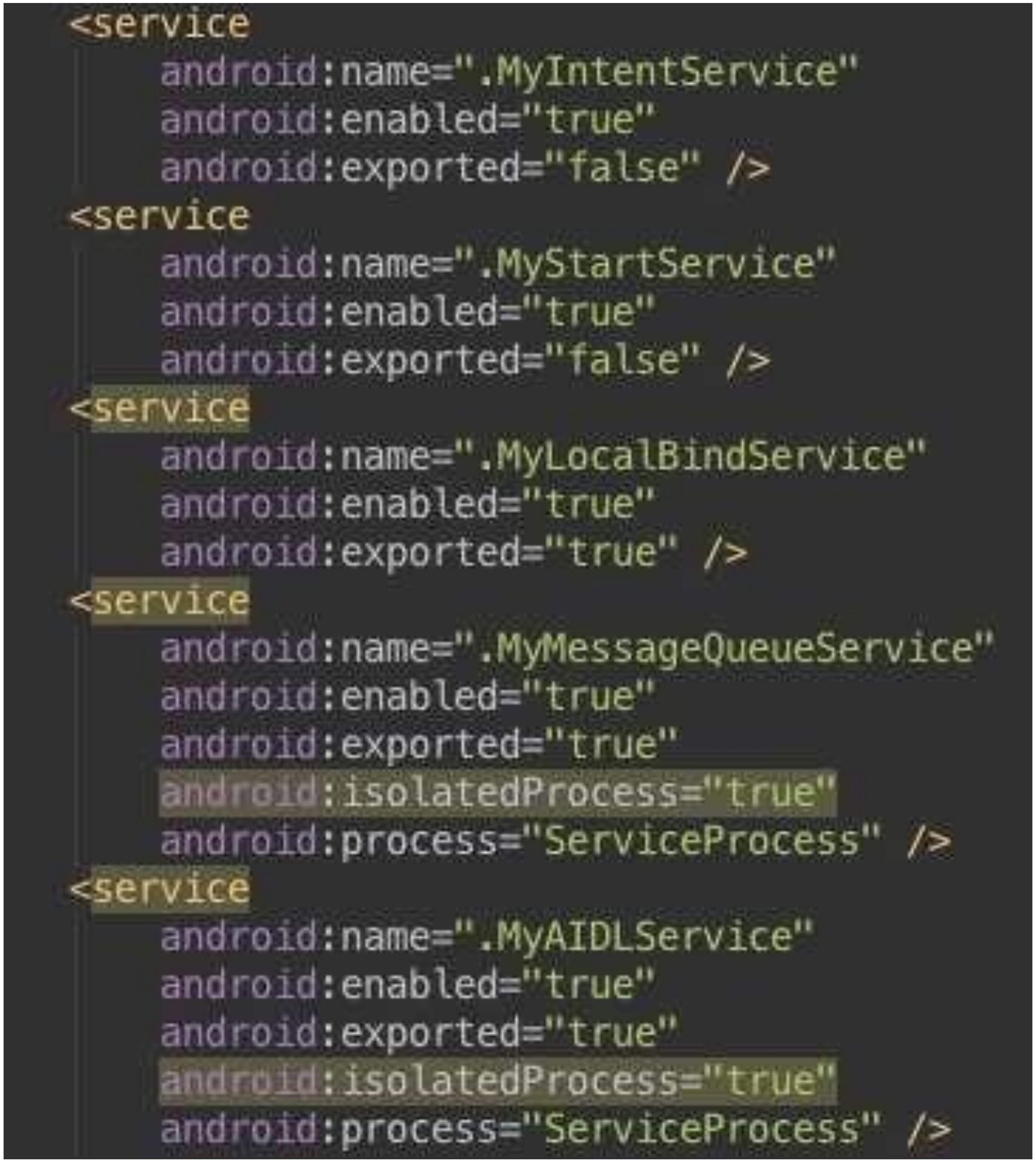


```
Intent startIntent = new Intent(this, MyStartService.class);
// ... do intent.putExtra([your data]) if need
startService(startIntent);

Intent stopIntent = new Intent(this, MyStartService.class);
stopService(stopIntent);
```

Refer to same service

Declaration of Service in AndroidManifest

- android:name="[package/service]"
- android:enabled="[true|false]"
- android:exported="[true|false]"
- android:isolatedProcess="[true|false]"
- android:process="[name/of/process]"



```
<service
    android:name=".MyIntentService"
    android:enabled="true"
    android:exported="false" />
<service
    android:name=".MyStartService"
    android:enabled="true"
    android:exported="false" />
<service
    android:name=".MyLocalBindService"
    android:enabled="true"
    android:exported="true" />
<service
    android:name=".MyMessageQueueService"
    android:enabled="true"
    android:exported="true"
    android:isolatedProcess="true"
    android:process="ServiceProcess" />
<service
    android:name=".MyAIDLService"
    android:enabled="true"
    android:exported="true"
    android:isolatedProcess="true"
    android:process="ServiceProcess" />
```

Isolated Service

- android:isolatedProcess=“true”
- android:process=“[process_name]”
- Even you give two isolated processes the same process name, they will NOT run in same process.

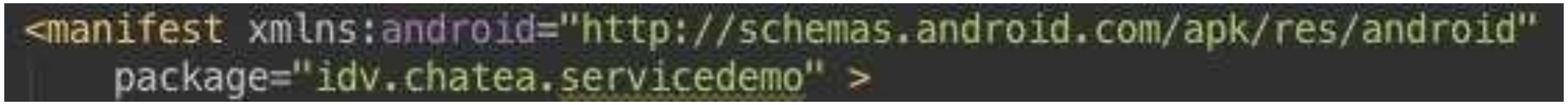
  Thus, you get at least 3 process: app, service1, and service2

  The process names of service1 and service2 are same

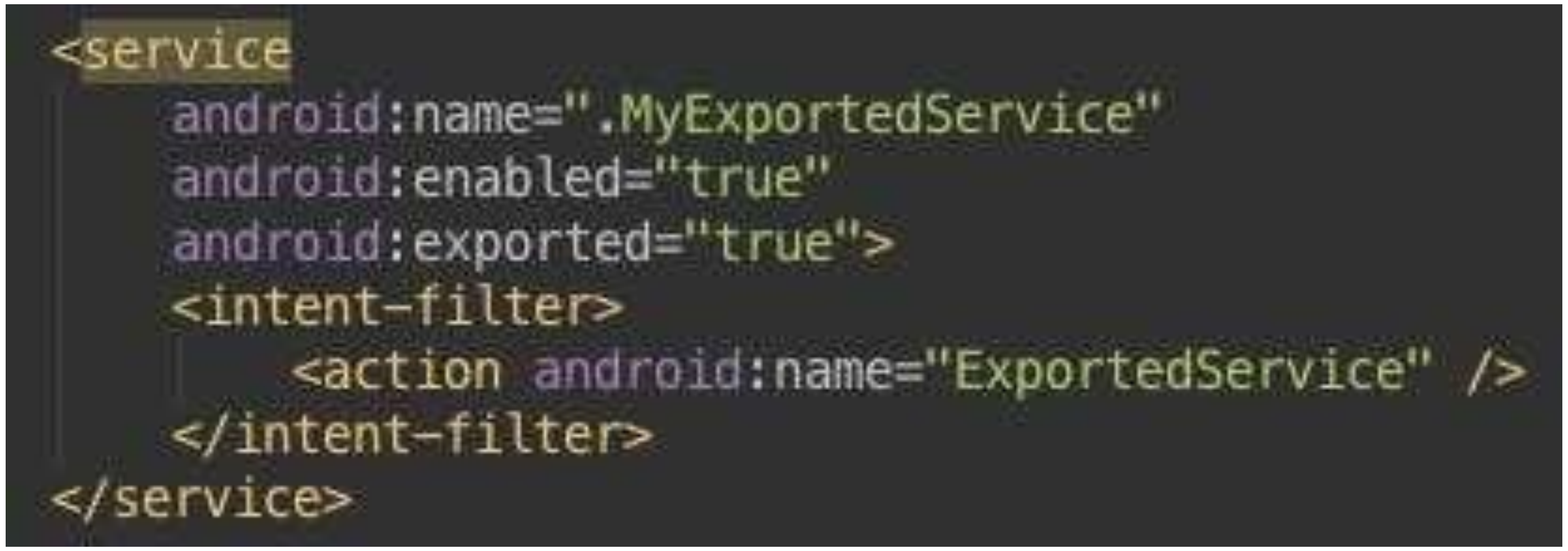
Exported Service

- Exported Service makes your app be used from other application's service

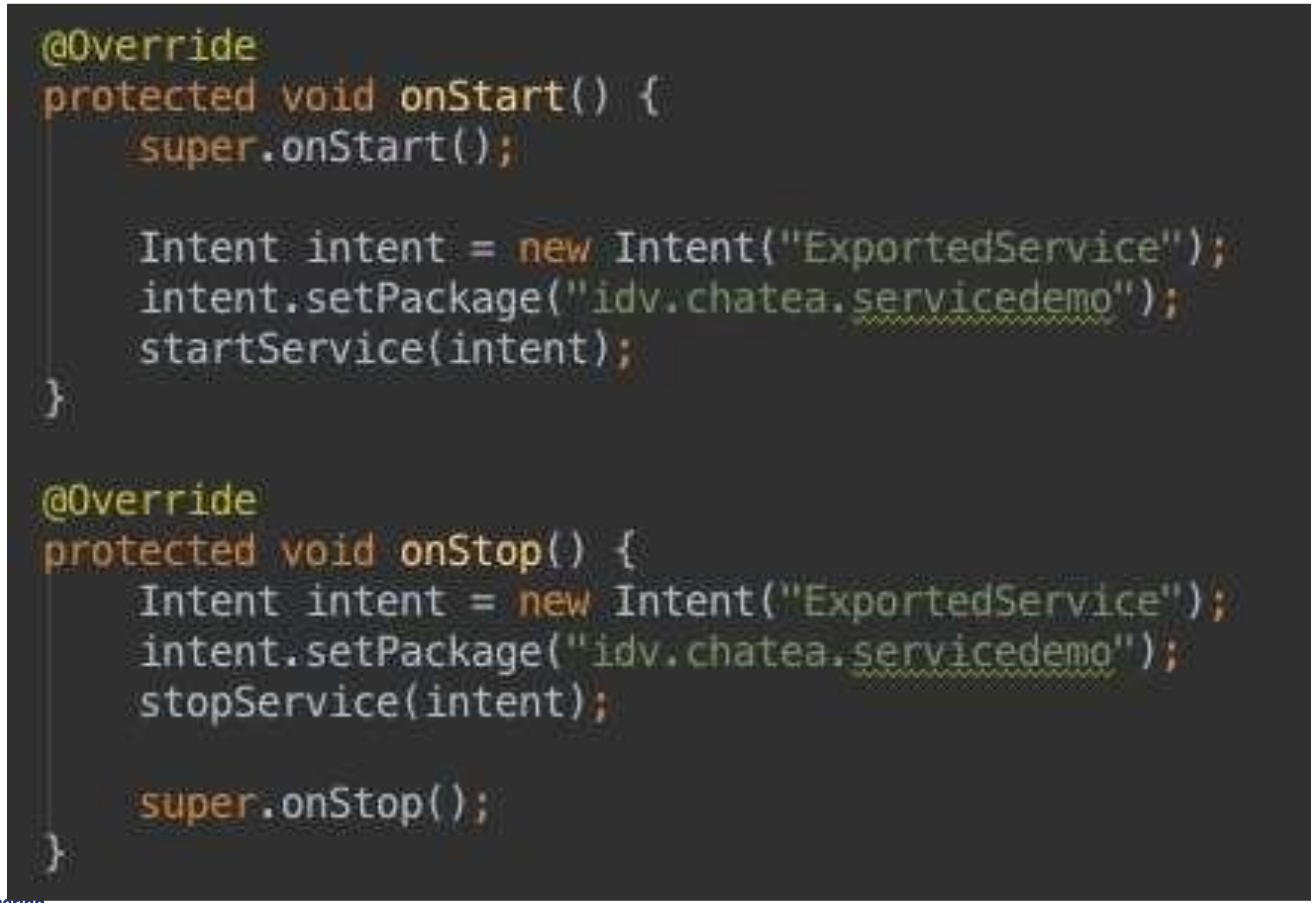
- The exported service is run in the process of its application, NOT in the process of caller



```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="idv.chatea.servicedemo" >
```



```
<service
    android:name=".MyExportedService"
    android:enabled="true"
    android:exported="true">
    <intent-filter>
        <action android:name="ExportedService" />
    </intent-filter>
</service>
```

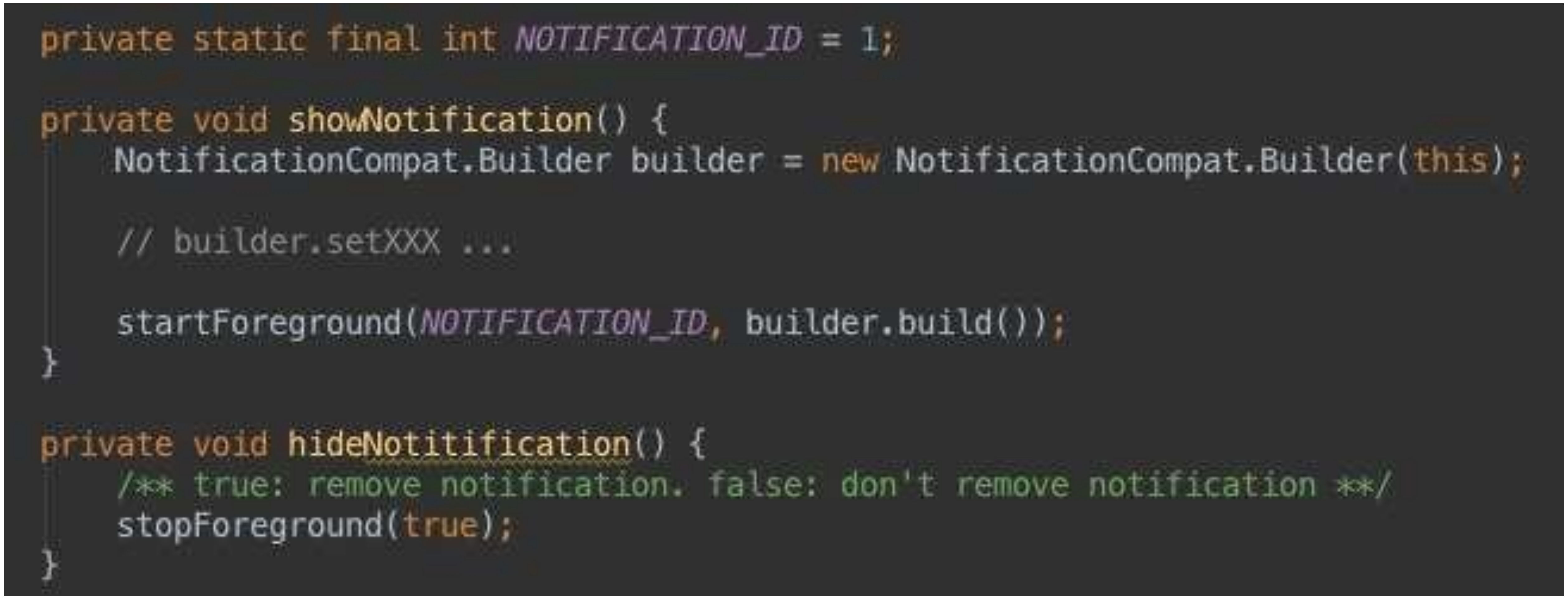


```
@Override
protected void onStart() {
    super.onStart();

    Intent intent = new Intent("ExportedService");
    intent.setPackage("idv.chatea.servicedemo");
    startService(intent);
}

@Override
protected void onStop() {
    Intent intent = new Intent("ExportedService");
    intent.setPackage("idv.chatea.servicedemo");
    stopService(intent);

    super.onStop();
}
```

Notification

- Notification is part of Service
- Use NotificationCompat.Builder
- startForeground(int notificationId, Notification)
  # the notificationId must NOT be 0
- startForeground with same id will replace the previous notification which has same id.

Example: Notification



```java
private static final int NOTIFICATION_ID = 1;

private void showNotification() {
    NotificationCompat.Builder builder = new NotificationCompat.Builder(this);

    // builder.setXXX ...

    startForeground(NOTIFICATION_ID, builder.build());
}

private void hideNotitification() {
    /** true: remove notification. false: don't remove notification **/
    stopForeground(true);
}
```

Thanks!