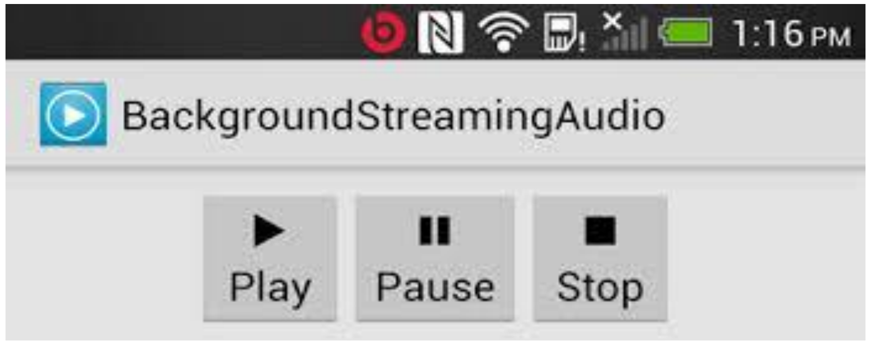
Services

Why Android Services

- Provide background functionality
- Can run even when app is closed
- Can start on boot or other events
- Can be accessible by multiple apps

Some examples of Service:

- to implement location listener,
- sound module, generating various voices
- in app content updates,
- API, provide services to other apps
- in app billing
- Communication with webservices



Music player application example

Requirements:

- this application always need to play the music even though application is not visible or partial visible.
- These requirements can be achieved via using a thread inside the activity class.
- But, the thread life is associated with the activity life cycle, so whenever activity is recreated the thread will also goes off.
- This way a thread can't provide a mechanism to continue the operation.
- Service does not destroy when you rotate the device from portrait mode to landscape mode like activity does.
- A service still runs in the background while user is not interacting with the application i.e application is not visible to the user.

Services

- A Service is an application component that can perform long-running operations in the background; it does not provide a UI.
  - Services are declared in the Manifest
  - Services can be exposed to other processes
  - Services do not need to be connected with an Activity
  - Services are Task with no UI.

- A Service provides a robust environment for background tasks …

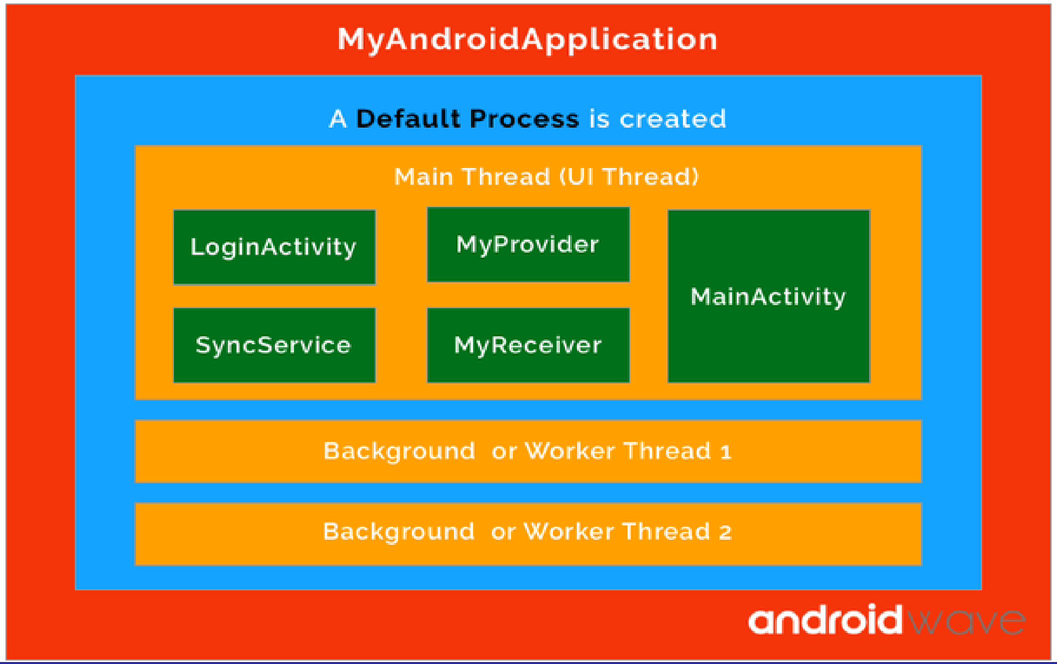
Android Services vs. Threads

Android Services

- Cannot access the UI
- Runs in main thread of host app process
- Can contain multiple threads

Threads

- Cannot access the UI
- Not accessible to other apps
- Terminated with app
- Used to improve responsiveness



Service vs Thread in Android

Any difference between Services and Thread and where you should use them?

- **Diff1**: Thread will sleep if your device sleeps. Whereas, Service can perform operation even if the device goes to sleep.
- Example playing music using both approaches.
  - **Thread Approach**: the music will only play if your app is active or screen display is on.
  - **Service Approach**: the music can still play even if you minimized your app or screen is off.

Service vs Thread in Android

Any difference between Services and Thread and where you should use them?

- **Diff2**: If you need to perform work outside your **main thread**, but only while the user is interacting with your application, then you should create a new thread and not a service.

- **Remember** that if you do use a service, it still runs in your application's main thread by default, so you should still create a new thread within the service if it performs intensive or blocking operations.

Three different types of services

- Background services
  - Is a service that runs only when the app is running
  - Is terminated when the app is terminated
- Foreground services
  - Stays alive even when the app is terminated
- Bound services
  - Runs only when the component it is bound to is still alive

Background Services

Create a Local Service in background

- To use a Service it is necessary to carry out two operations:
- create a **class** that extends Service or its subclass

```
import android.app.Service

class MyService : Service() {
  override fun onBind(intent: Intent?): IBinder? {…}
  …
}
```

- register the service in the **manifest** with the <service> node

```
<application
  …
  <activity android:name=".MainActivity">
  ...
  </activity>
  ...
</application>
```

```
<service
  android:name=".MyService"
  android:enabled="true"
  android:exported="false">
</service>
```

Create a Local Service in background

- You must override some callback methods of Service
- The most important callback methods that you should override:
  - **onStartCommand()**
    The system invokes this method by calling startService()
  - **onBind()**
    The system invokes this method by calling bindService()
  - **onCreate()**
    The system invokes it when the service is initially <u>created</u>
  - **onDestroy()**
    The system invokes it when the service is <u>no longer used</u>

Registration in the Manifest

```
<service android:enabled=["true" | "false"]
         android:exported=["true" | "false"]
         android:icon="drawable resource"
         android:isolatedProcess=["true" | "false"]
         android:label="string resource"
         android:name="string"
         android:permission="string"
         android:process="string" >
    . . .
</service>
```

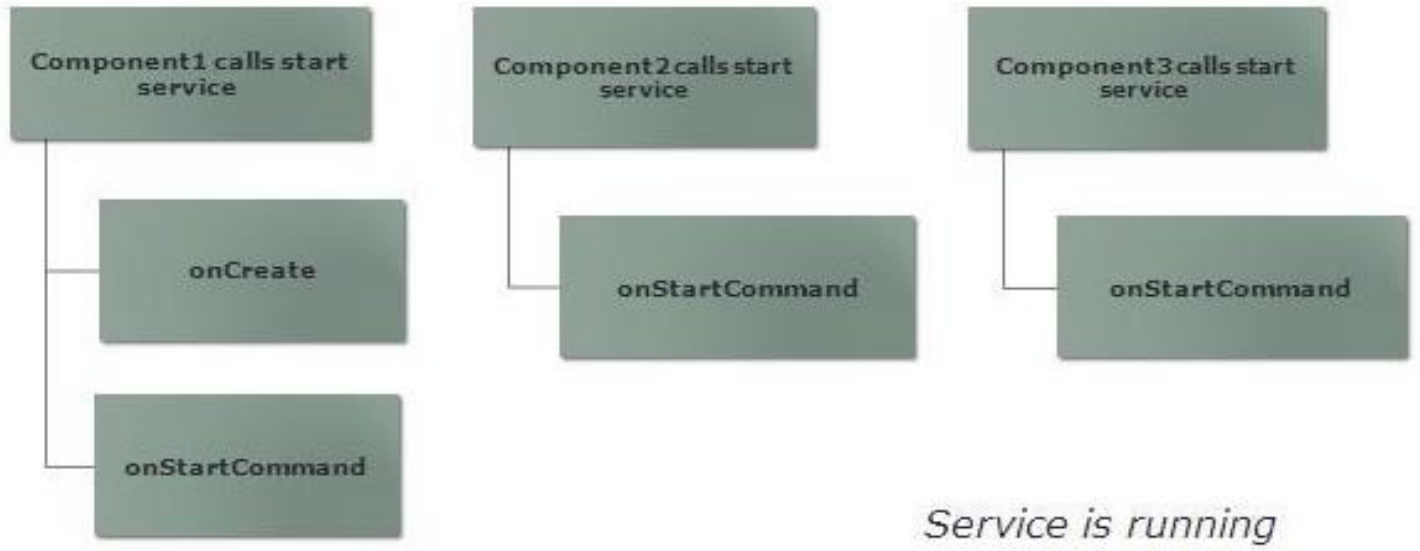
- android:enabled
  - Whether or not the service can be instantiated by the system; "true" if it can be, and "false" if not. The default value is "true".
- android:exported
  - Whether or not components of other applications can invoke the service or interact with it; "true" if they can, and "false" if not.

Start a Service

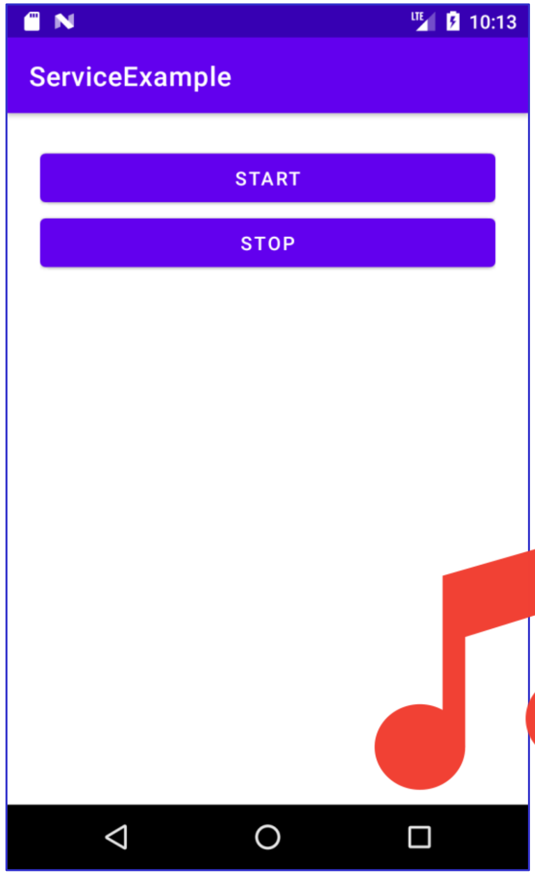
- A Service is *started* when an application component starts it by calling `startService(Intent)`.

  - Once started, a *Service runs in background indefinetely*.

- *Termination* of a Service:
  - `selfStop()`: self-termination of the service
  - `stopService(Intent)`: terminated by others
  - killed by the system


Component1 calls start service
onCreate
onStartCommand
Component2 calls start service
onStartCommand
Component3 calls start service
onStartCommand
Service is running

Service running music



AndroidManifest.xml

Services class is simple as it starts an audio (mp3) file and the start button invokes the service.

Step1: create a Service Controller (Activity)

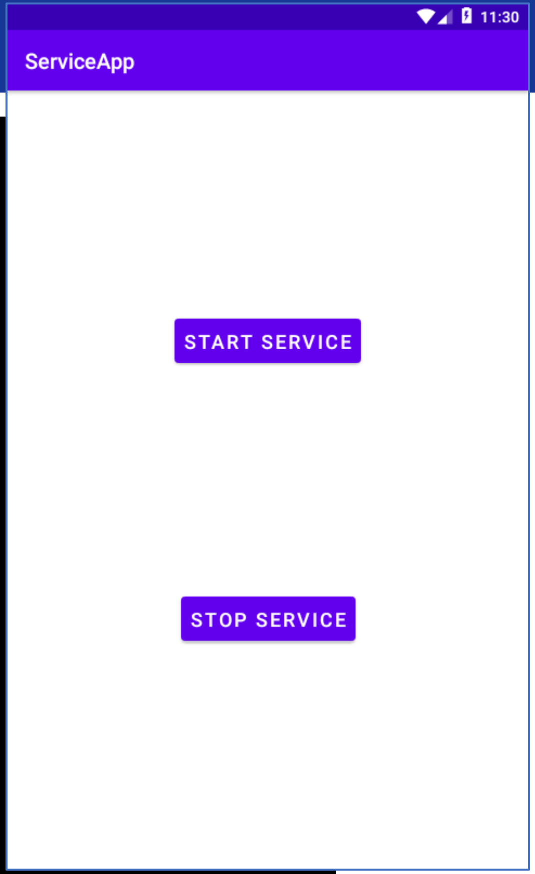
```kotlin
class MainActivity : AppCompatActivity() {
   override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      setContentView(R.layout.activity_main)
      buttonStart.setOnClickListener {
         val intentBg = Intent(this, MusicBackgroundService::class.java)
         startService(intentBg)
      }
      buttonStop.setOnClickListener {
         val intentBg = Intent(this, MusicBackgroundService::class.java)
         stopService(intentBg)
      }
   }
}
```

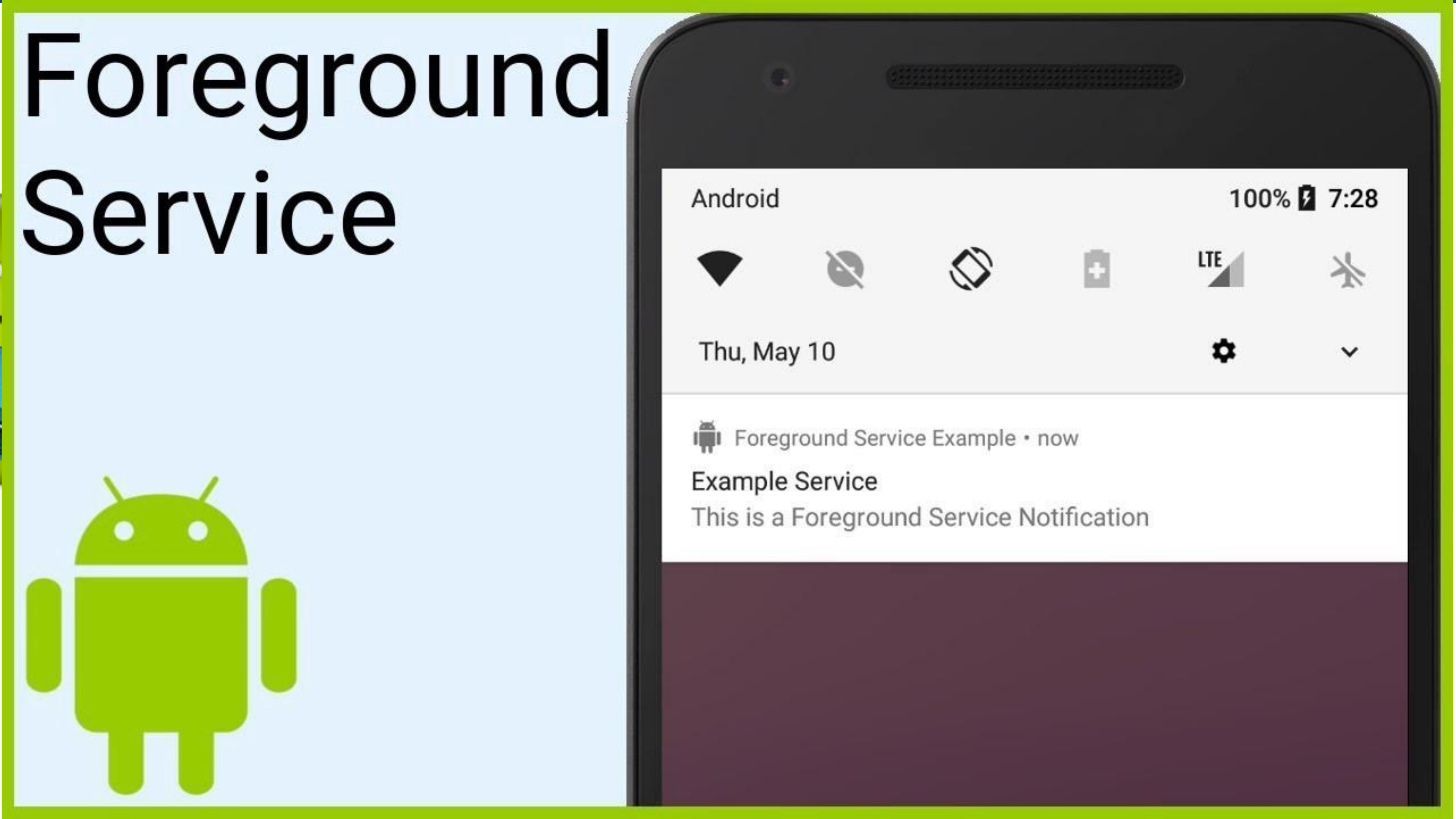
Step2: create a service class

```
class MusicBackgroundService : Service() {
    private lateinit var player: MediaPlayer
    override fun onCreate() {
        super.onCreate()
        player = MediaPlayer.create(this, R.raw.queen_we_are_the_champions)
        Log.i("MusicBackgroundService", "Service created")
        player.isLooping = false
    }
    override fun onStartCommand(intent: Intent?, flags: Int, startId: Int): Int {
        player.start()
        Log.i("MusicBackgroundService", "Music starts")
        return super.onStartCommand(intent, flags, startId)
    }
    override fun onBind(p0: Intent?): IBinder? {
        return null
    }
    override fun onDestroy() {
        super.onDestroy()
        Log.i("MusicBackgroundService", "Music stops")
        player.stop()
    }
}
```

Step3: modify AndroidManifest.xml

```
<application
android:allowBackup="true"
android:icon="@mipmap/ic_launcher"
android:label="@string/app_name"
android:roundIcon="@mipmap/ic_launcher_round"
android:supportsRtl="true"
android:theme="@style/Theme.ServiceApp">

<service android:name=".MusicBackgroundService"/>

<activity
android:name=".MainActivity"
android:exported="true">
<intent-filter>
<action android:name="android.intent.action.MAIN" />
<category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
</application>
```





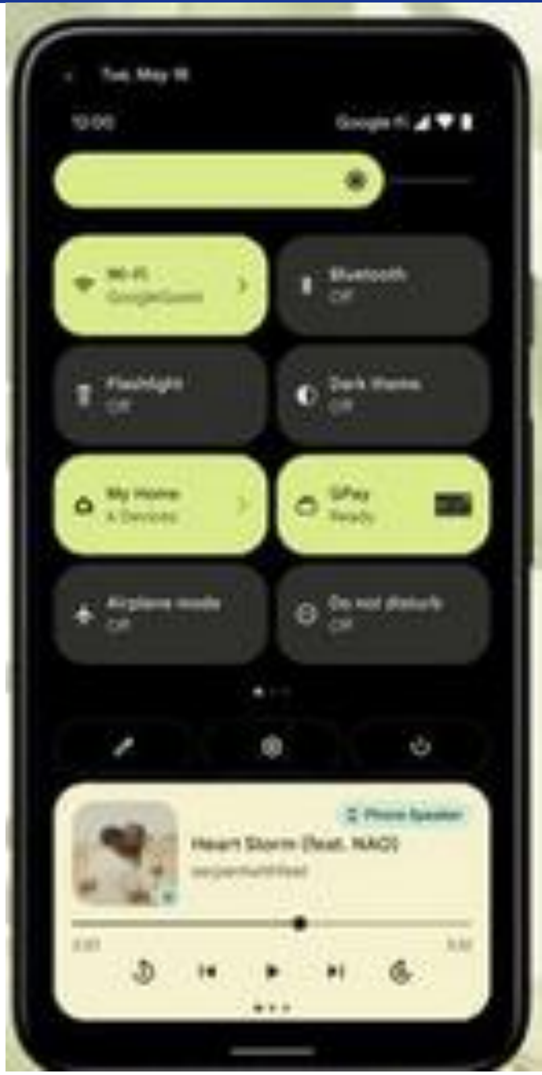
Foreground Service Example • now

Example Service

This is a Foreground Service Notification

Foreground services

- A foreground service performs some operation that is noticeable to the user.
- For example, an audio app would use a foreground service to play an audio track.
- Foreground services must display a Notification.
- Foreground services continue running even when the user isn't interacting with the app.



Foreground Services

- A Foreground Service is a service that is continuously active in the Status Bar, and thus it is not a good candidate to be killed in case of low memory.
- To create a Foreground Service:
  - Create a Notification object
  - Call `startForeground(id, notification)` from `onStartCommand()`
  - Call `stopService()/stopForeground()` to stop the Service.

Step1: create a Service Controller (Activity)

```kotlin
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        buttonStart.setOnClickListener {
            val intentFg = Intent(this, MusicForegroundService::class.java)
            startService(intentFg)
        }
        buttonStop.setOnClickListener {
            val intentFg = Intent(this, MusicForegroundService::class.java)
            stopService(intentFg)
        }
    }
}
```

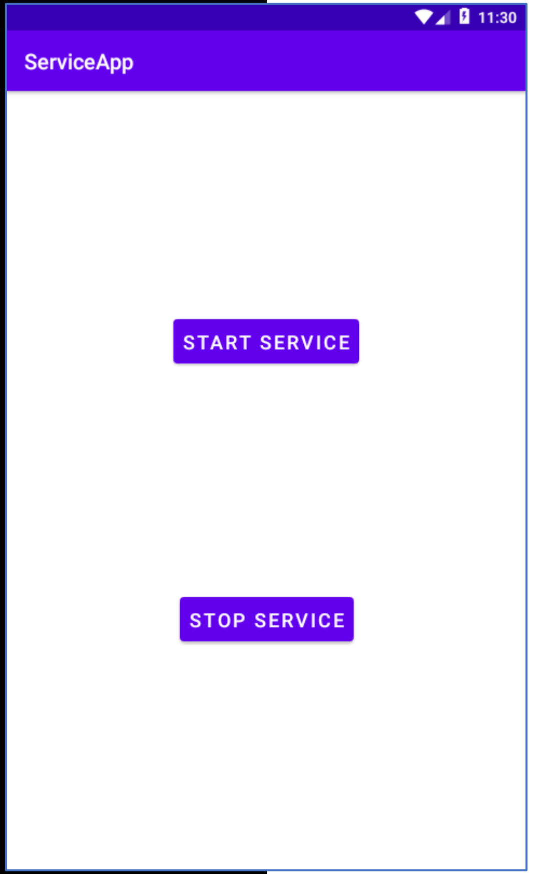
Step2: create a service class

```kotlin
class MusicForegroundService : Service() {
   private lateinit var player: MediaPlayer
   private val CHANNEL_ID = "ForegroundService Kotlin"

   override fun onStartCommand(intent: Intent?, flags: Int, startId: Int): Int {
      player.start()
      Log.i("MusicForegroundService", "Music starts")
      createNotificationChannel()
      val notificationIntent = Intent(this, MainActivity::class.java)
      val pendingIntent = PendingIntent.getActivity(this, 0, notificationIntent, 0)
      val notification = NotificationCompat.Builder(this, CHANNEL_ID)
         .setContentTitle("Foreground Service Kotlin Example")
         .setContentText("You are listening queen_we_are_the_champions...")
         .setSmallIcon(R.drawable.ic_notifications)
         .setContentIntent(pendingIntent)
         .build()
      startForeground(1, notification)
      return super.onStartCommand(intent, flags, startId)
   }
   override fun onBind(p0: Intent?): IBinder? { return null }
   override fun onCreate() {...}
   override fun onDestroy() {...}
}
```

```kotlin
private fun createNotificationChannel() {
   if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
   val serviceChannel = NotificationChannel(CHANNEL_ID,
                        "Foreground Service Channel",
                        NotificationManager.IMPORTANCE_DEFAULT)
   val manager = getSystemService(NotificationManager::class.java)
   manager!!.createNotificationChannel(serviceChannel)
}
```

Step3: modify AndroidManifest.xml

```
<uses-permission android:name="android.permission.FOREGROUND_SERVICE"/>

<application
android:allowBackup="true"
android:icon="@mipmap/ic_launcher"
android:label="@string/app_name"
android:roundIcon="@mipmap/ic_launcher_round"
android:supportsRtl="true"
android:theme="@style/Theme.ServiceApp">

<service android:name=".MusicForegroundService"/>

<activity
android:name=".MainActivity"
android:exported="true">
<intent-filter>
<action android:name="android.intent.action.MAIN" />
<category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
</application>
```

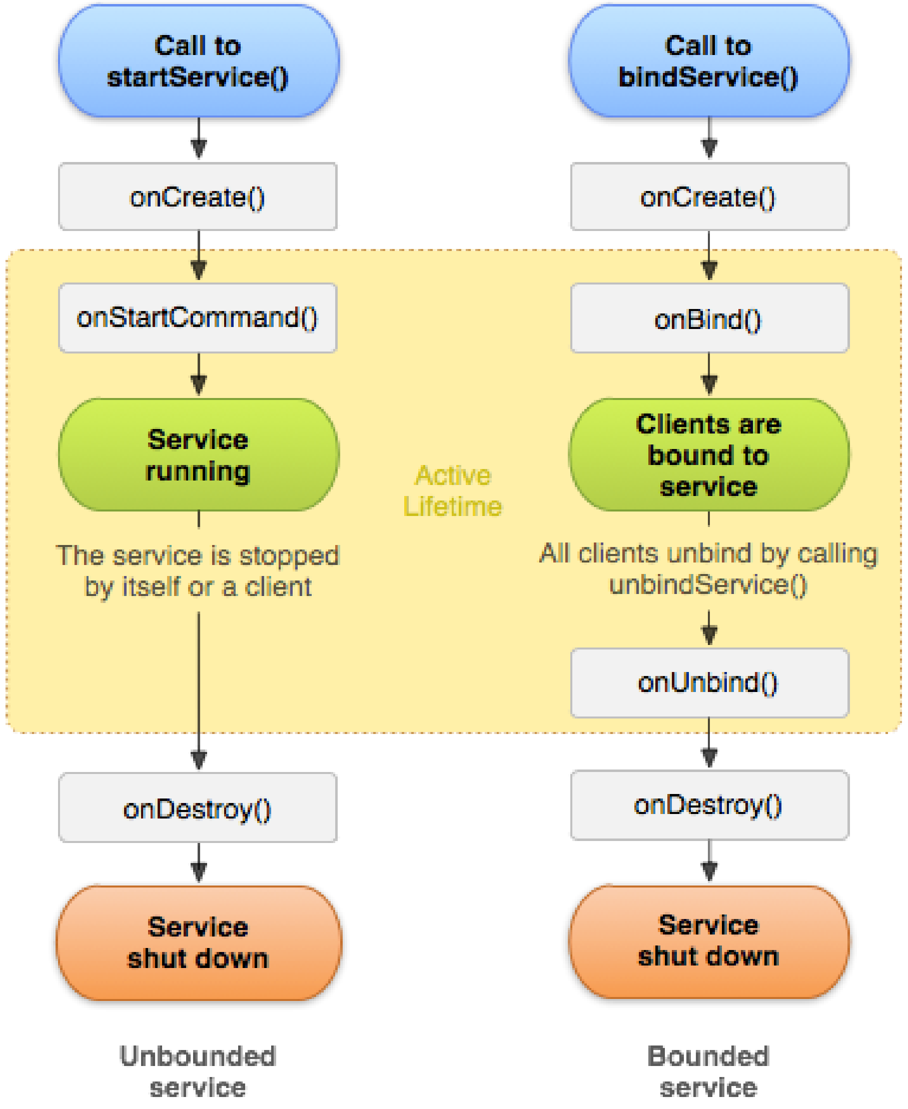


Bound Services

Bounded and Unbounded Services

Unbounded Service

started when an application component, such as an activity, starts it by calling **startService()**. Once started, a service can run in the background indefinitely, even if the component that started it is destroyed.



Bounded Service

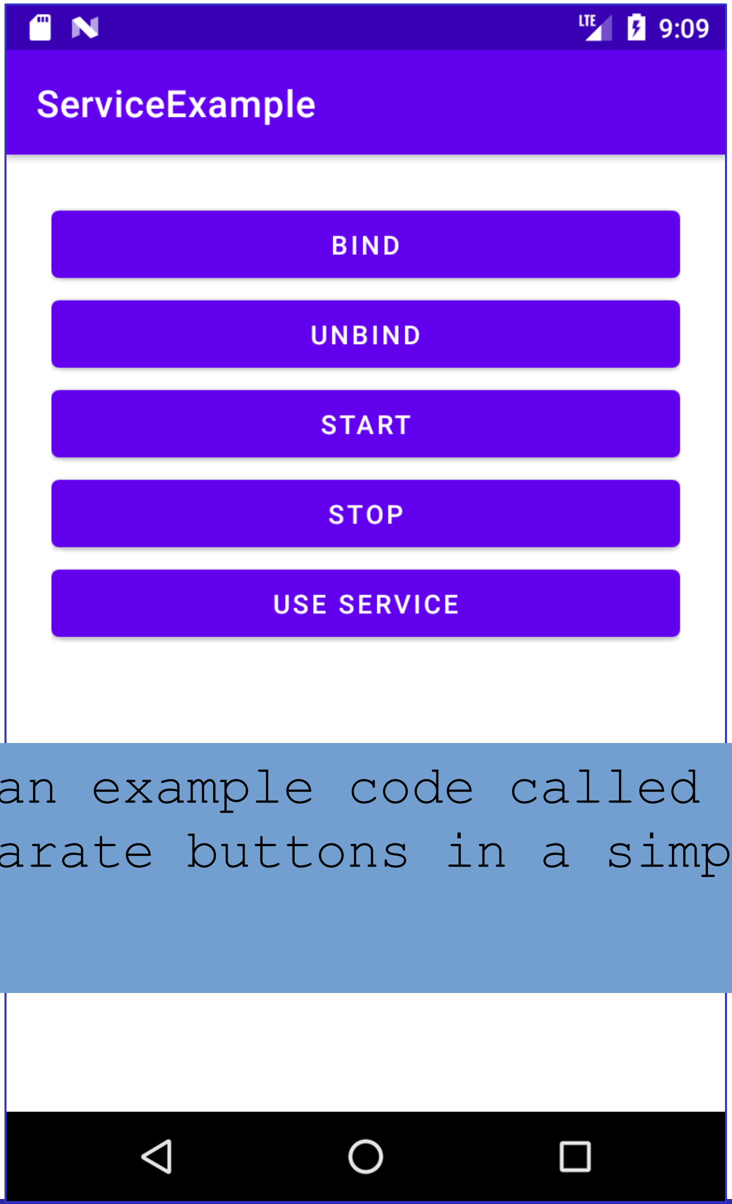
when an application component binds to it by calling **bindService()**.
A bound service offers a client-server interface that allows components to interact with the service, **send requests, get results**, and even do so across processes with interprocess communication (IPC).
When the last client unbinds from the service, the system destroys the service.

IBinder

- When creating a Service, an `IBinder` must be created to provide an Interface that clients can use to interact with the Service.
  - Extending the `Binder` class (local Services only)
    - Extend the `Binder` class and return it from `onBind()`
    - Only for a Service used by the same application
- Using the **Android Interface Definition Language** (AIDL)
- Allow to access a Service from different applications.

Interaction between Activity and Service

```
fun onBindBtnClick(v: View){
   val intent = Intent(this, LocalService::class.java)
   bindService(intent, servcConn, BIND_AUTO_CREATE)
   mBound = true
}
fun onUnBindBtnClick(v: View){
   if (mBound) {
      unbindService(servcConn)
      mBound = false
   }
}
fun onStopBtnClick(v: View){
   val intent = Intent(this, LocalService::class.java)
   stopService(intent)
}
fun onStartBtnClick(v: View){
   val intent = Intent(this, LocalService::class.java)
   startService(intent)
}
```



Here is an example code called from separate buttons in a simple test app

From a simple Activity and Service

bind-unbind

```
bindService() caused:
    onCreate()
    onBind()
unbindService() caused:
    onUnbind()
    onDestroy()
```

start-bind-unbind-stop

```
startService() caused:
    onCreate()
    onStartCommand()
bindService() caused:
    onBind()
unbindService() caused:
    onUnbind()
stopService() caused:
    onDestroy()
```

start-bind-stop-unbind

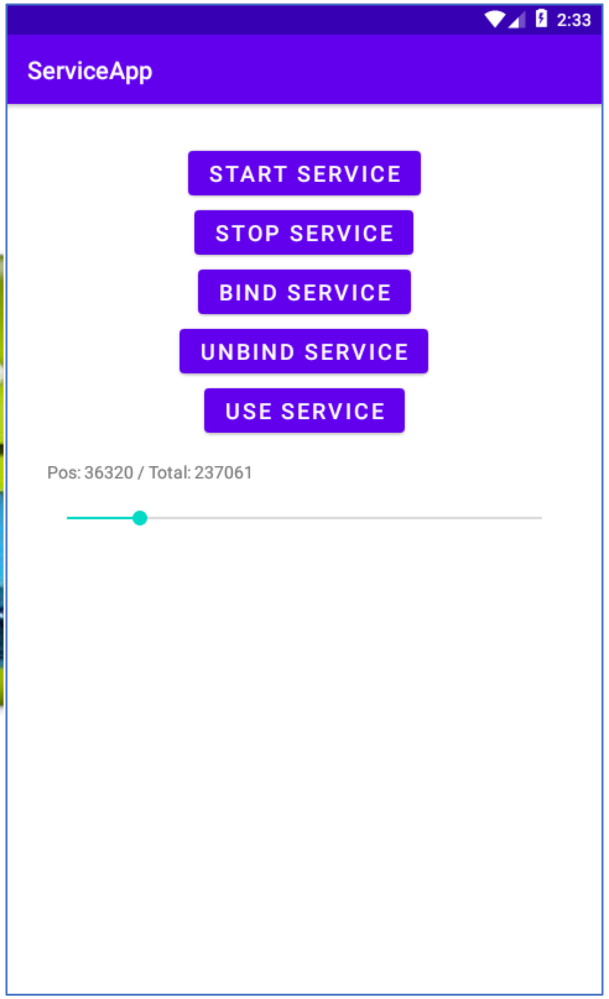
```
startService() caused:
    onCreate()
    onStartCommand()
bindService() caused:
    onBind()
stopService() caused:
    -- nothing
unbindService() caused:
    onUnbind()
    onDestroy()
```

bind-start-stop-unbind

```
bindService() caused:
    onCreate()
    onBind()
startService() caused:
    onStartCommand()
stopService() caused:
    -- nothing
unbindService() caused:
    onUnbind()
    onDestroy()
```

bind-start-unbind-stop

```
bindService() caused:
    onCreate()
    onBind()
startService() caused:
    onStartCommand()
unbindService() caused:
    onUnbind()
stopService() caused:
    onDestroy()
```

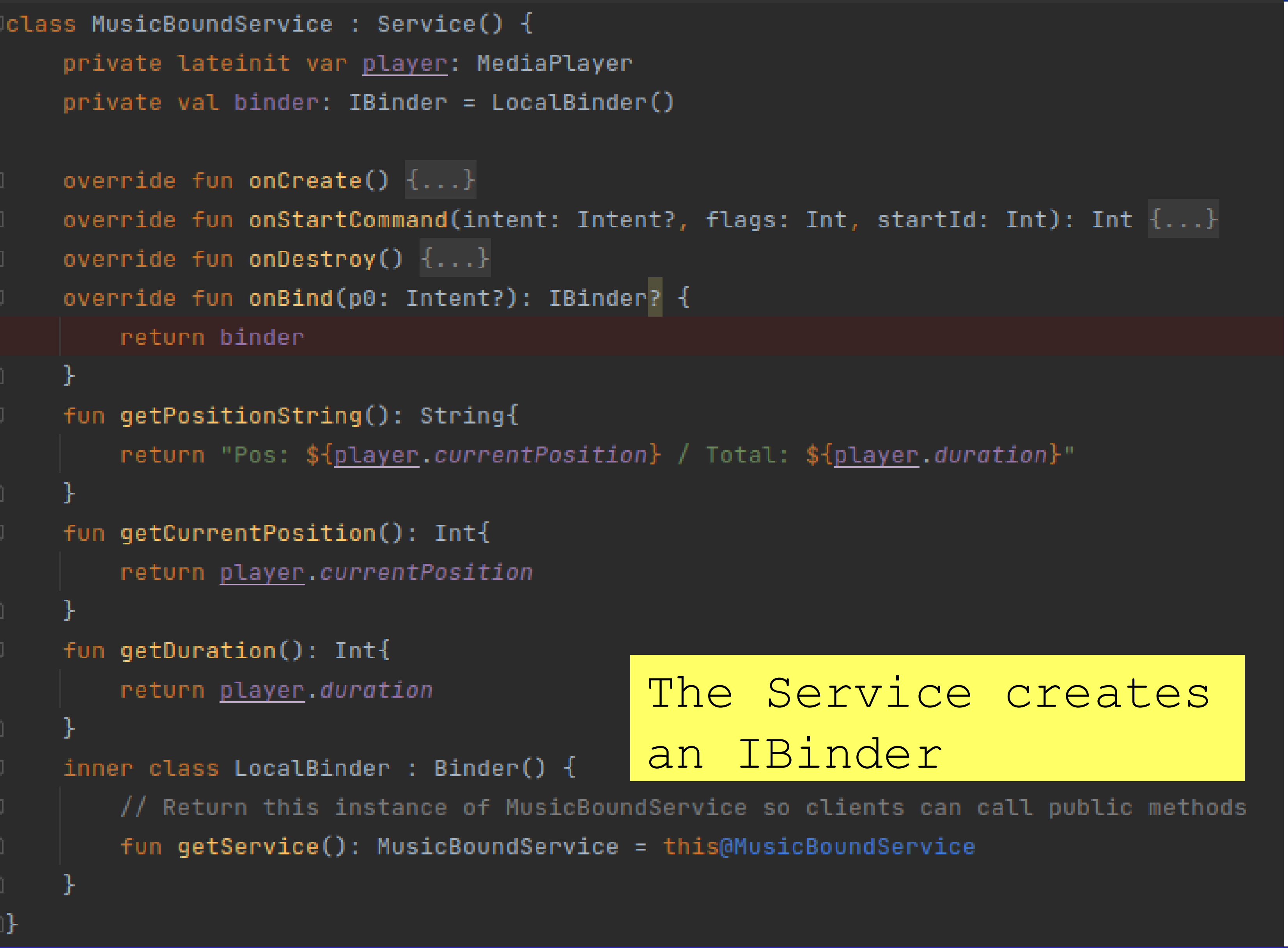


• Random Number Service

AndroidManifest.xml

The Service class read position of MediaPlayer

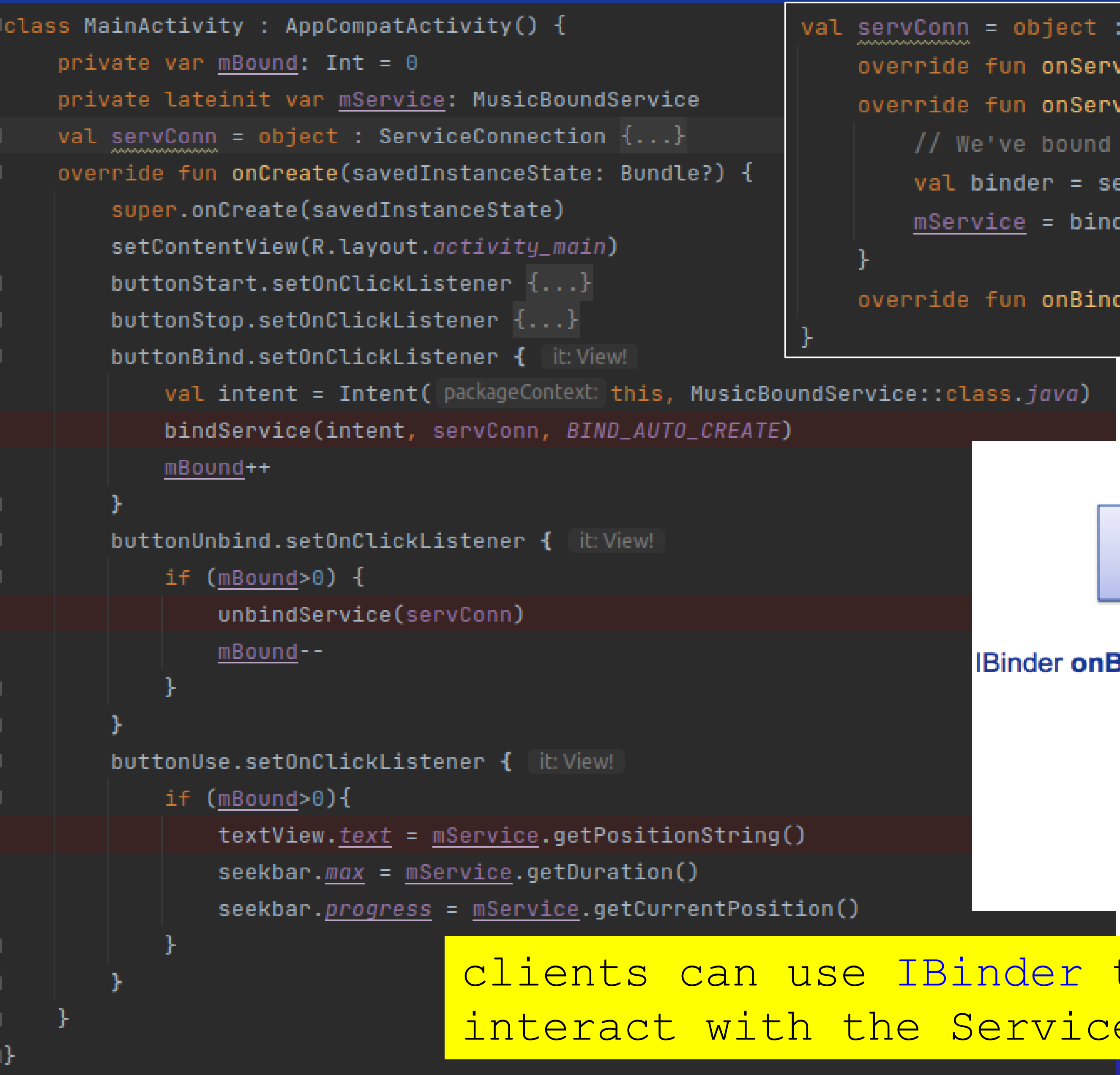
Bound services



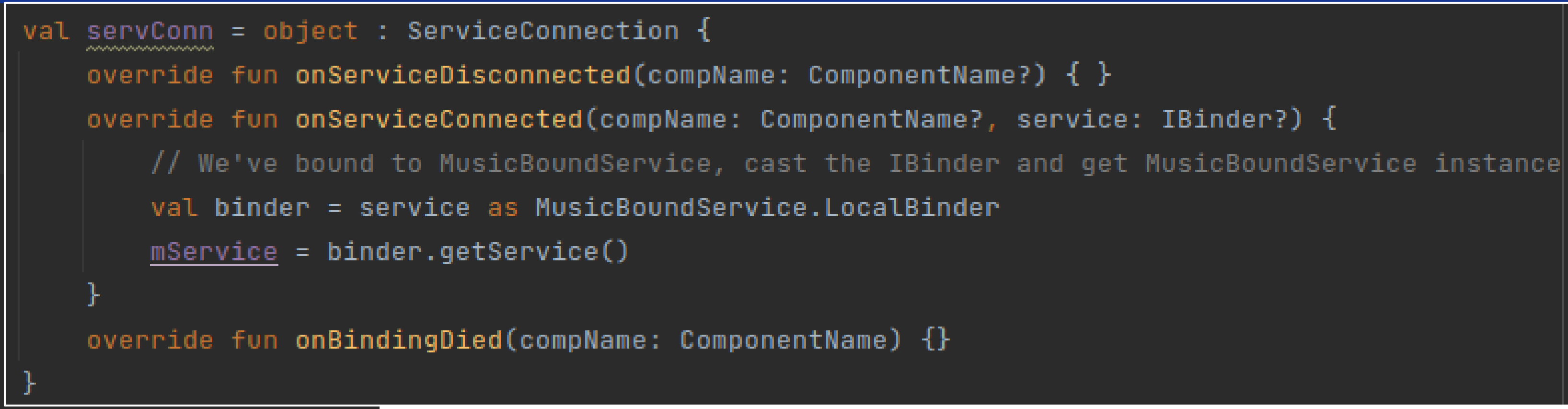
```kotlin
class MusicBoundService : Service() {
    private lateinit var player: MediaPlayer
    private val binder: IBinder = LocalBinder()

    override fun onCreate() {...}
    override fun onStartCommand(intent: Intent?, flags: Int, startId: Int): Int {...}
    override fun onDestroy() {...}
    override fun onBind(p0: Intent?): IBinder? {
        return binder
    }
    fun getPositionString(): String{
        return "Pos: ${player.currentPosition} / Total: ${player.duration}"
    }
    fun getCurrentPosition(): Int{
        return player.currentPosition
    }
    fun getDuration(): Int{
        return player.duration
    }
    inner class LocalBinder : Binder() {
        // Return this instance of MusicBoundService so clients can call public methods
        fun getService(): MusicBoundService = this@MusicBoundService
    }
}
```

The Service creates an IBinder

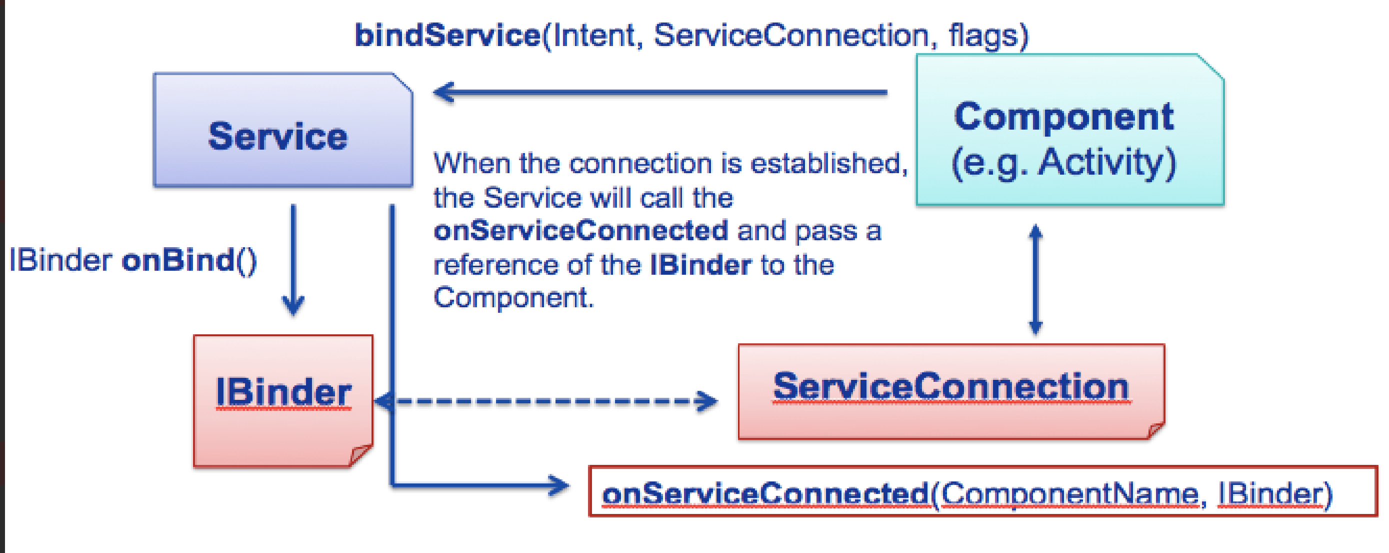
Bound services: connection



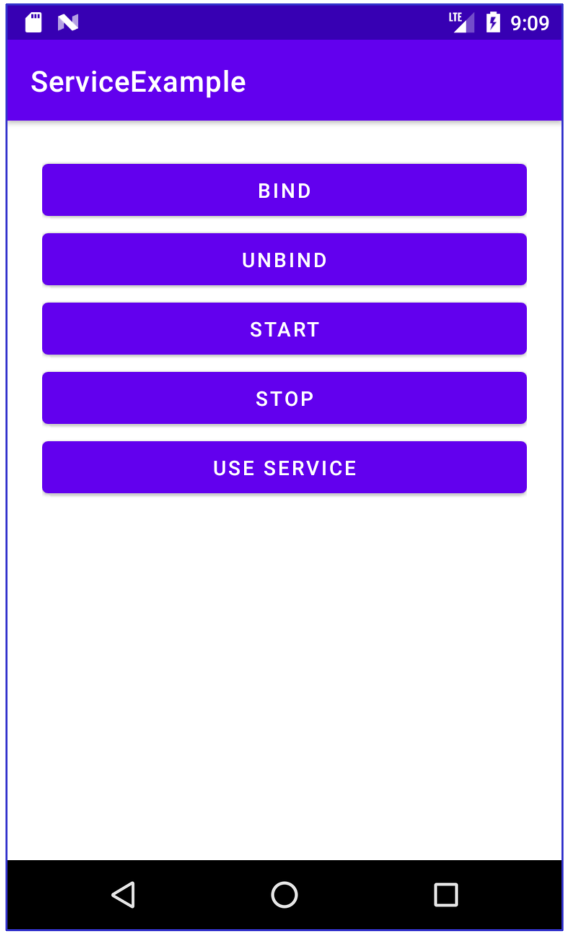
```
class MainActivity : AppCompatActivity() {
    private var mBound: Int = 0
    private lateinit var mService: MusicBoundService
    val servConn = object : ServiceConnection {...}
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        buttonStart.setOnClickListener {...}
        buttonStop.setOnClickListener {...}
        buttonBind.setOnClickListener {  it: View!
            val intent = Intent( packageContext: this, MusicBoundService::class.java)
            bindService(intent, servConn, BIND_AUTO_CREATE)
            mBound++
        }
        buttonUnbind.setOnClickListener {  it: View!
            if (mBound>0) {
                unbindService(servConn)
                mBound--
            }
        }
        buttonUse.setOnClickListener {  it: View!
            if (mBound>0){
                textView.text = mService.getPositionString()
                seekbar.max = mService.getDuration()
                seekbar.progress = mService.getCurrentPosition()
            }
        }
    }
}
```



```
val servConn = object : ServiceConnection {
    override fun onServiceDisconnected(compName: ComponentName?) { }
    override fun onServiceConnected(compName: ComponentName?, service: IBinder?) {
        // We've bound to MusicBoundService, cast the IBinder and get MusicBoundService instance
        val binder = service as MusicBoundService.LocalBinder
        mService = binder.getService()
    }
    override fun onBindingDied(compName: ComponentName) {}
}
```


bindService(Intent, ServiceConnection, flags)
Service
Component (e.g. Activity)
When the connection is established, the Service will call the **onServiceConnected** and pass a reference of the **IBinder** to the Component.
IBinder **onBind**()
IBinder
ServiceConnection
onServiceConnected(ComponentName, IBinder)

clients can use IBinder to interact with the Service



• Random Number Service

```
AndroidManifest.xml

The Service class generates random numbers.
```

Bound services

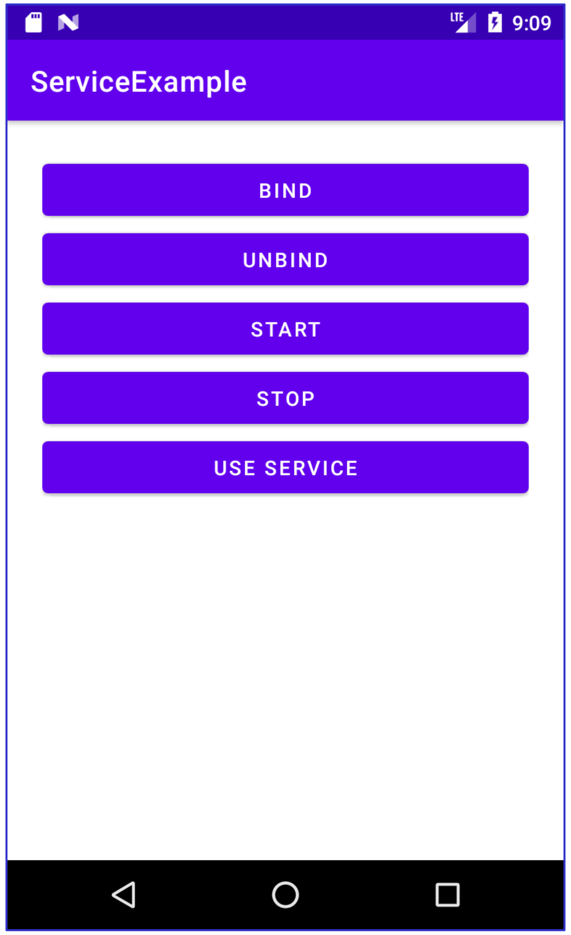
```
class LocalService: Service() {
private val binder: IBinder = MyBinder(this)
private val generator: Random = Random()

override fun onBind(intent: Intent?): IBinder? {
   return binder
}
override fun onUnbind(intent: Intent?): Boolean {
   return super.onUnbind(intent)
}
override fun onCreate() {
    super.onCreate()
}
override fun onDestroy() {
   super.onDestroy()
}
override fun onStartCommand(intent: Intent?, flags: Int, startId: Int): Int {
   return super.onStartCommand(intent, flags, startId)
}
/** method for clients */
val randomNumber: Int
   get() = generator.nextInt(100)
}
```

The Service creates an IBinder

```
class MyBinder(val servc:LocalService) : Binder() {
    fun getService():LocalService {
       return servc
   }
}
```

Bound services: connection

```
class MainActivity : AppCompatActivity() {
private var mBound: Boolean = false
private lateinit var mService: LocalService

val servcConn = object : ServiceConnection {
   override fun onServiceDisconnected(compName: ComponentName?) { }
   override fun onServiceConnected(compName: ComponentName?, binder: IBinder?) {
      mService = (binder as MyBinder).getService()   }
   override fun onBindingDied(compName: ComponentName) {}
}
override fun onCreate(savedInstanceState: Bundle?) {...}
fun onBindBtnClick(v: View){
   val intent = Intent(this, LocalService::class.java)
   bindService(intent, servcConn, BIND_AUTO_CREATE)
   mBound = true
}
fun onUnBindBtnClick(v: View){
   if (mBound) {
      unbindService(servcConn)
      mBound = false
   }}
fun onUseServiceClick(v: View){
   if (mBound) {
      val num: Int = mService.randomNumber
      Toast.makeText(this, "number: $num", Toast.LENGTH_SHORT).show()
   }}}
```



clients can use IBinder to interact with the Service