

● 서비스: 백그라운드 작업을 위한 컴포넌트로 화면과 상관없이 장시간 동안 처리해야 하는 업무를 구현할 때 사용한다. 하지만 시간이 오래 걸리는 업무라도 앱의 화면이 스마트폰에 떠 있는 동안만 진행되어야 한다면, 서비스가 아닌 액티비티로 작성하는 게 좋다. 즉, 서비스는 다른 앱이 사용자 화면을 점유하고 있더라도 앱의 프로세스가 살아서 계속 무언가 작업을 수행해야 할 때 사용하는 컴포넌트이다.

▶ 작성 방법: Service라는 클래스를 상속 받아서 작성한다.

```
public class PlayService extends Service {
    //...
}
```

이렇게 작성한 서비스도 컴포넌트 클래스이므로 AndroidManifest.xml 파일에 등록해야 한다. 액티비티나 브로드캐스트 리시버와 마찬가지로 암시적 인텐트로 수행되게 하려면 <intent-filter>를 서브 태그로 등록해 주어야 한다.

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

▶ 실행/종료 방법: 서비스를 실행하고 종료하기 위한 두 가지 함수가 있다. 서비스를 실행하면서 인텐트의 Extra 데이터를 전달할 수 있다. 그런데 서비스는 액티비티나 브로드캐스트 리시버와는 다르게 서비스를 종료하는 인텐트 발생도 제공한다. (서비스를 구동시킨 곳에서 종료 인텐트를 날리는 것이 아니다)

1. startService(), stopService()

```
Intent intent = new Intent(this, PlayService.class);
startService(intent);
```

```
Intent intent = new Intent(this, PlayService.class);
stopService(intent);
```

2. bindService(), unbindService(): 서비스를 bindService() 함수로 실행하려면, ServiceConnection 인터페이스를 구현한 객체를 준비해야 한다. ServiceConnection 인터페이스의 추상 함수인 onServiceConnected()는 bindService() 함수로 인해 서비스가 구동하는 시점에 호출되며, onServiceDisconnected() 함수는 unbindService() 함수로 인해 서비스가 종료된 시점에 호출된다.

```

ServiceConnection connection = new ServiceConnection() {
    @Override
    public void onServiceConnected(ComponentName, IBinder service) {
    }

    @Override
    public void onServiceDisconnected(ComponentName name) {
    }
};

```

bindService() 함수로 서비스를 구동하면서 두 번째 매개변수로 ServiceConnection을 구현한 객체를 알려주어야 한다. bindService() 함수로 실행된 서비스는 unbindService() 함수로 종료할 수 있다.

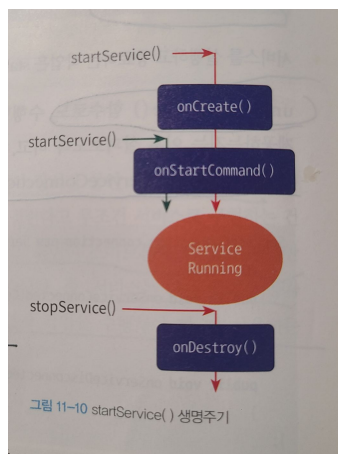
```

Intent intent = new Intent(this, BindService.class);
bindService(bIntent, connection, Context.BIND_AUTO_CREATE);
// ....
unbindService(connection);

```

▶ 서비스 생명주기: 서비스는 구동 방법이 startService()와 bindService() 두 가지이므로 어느 함수로 실행하는지에 따라 생명주기가 다르다.

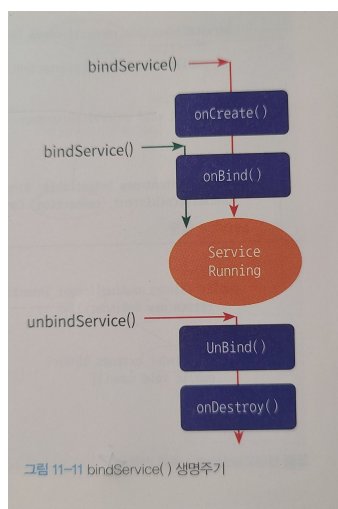
1. startService() 함수로 서비스를 실행하고, stopService() 함수로 종료할 때의 서비스 생명주기



startService() 함수가 서비스를 실행하면 onCreate() → onStartCommand() 함수가 자동으로 호출되면서 Service Running 상태가 되고, stopService() 함수가 서비스를 종료할 때는 onDestroy() 함수가 자동으로 호출되면서 종료된다.

서비스는 액티비티와 다르게 싱글톤으로 동작하는 컴포넌트다. 이미 실행되어 Running 상태인 서비스를 다시 startService() 함수로 실행되면 객체는 다시 생성되지 않는다. onCreate() 함수도 호출되지 않고, onStartCommand() 함수는 다시 호출된다. 즉, onCreate()는 최초 한 번만 호출되는 함수이고, onStartCommand()는 반복 호출할 수 있는 함수이다.

2. bindService() 함수로 서비스를 실행하고, unbindService() 함수로 종료할 때의 서비스 생명주기



bindService() 함수가 서비스를 시작하면 onCreate() → onBind() 함수가 호출되면서 Running 상태가 되고, unbindService() 함수로 서비스를 종료하면서 unbind() → onDestroy() 함수가 호출된다. 서비스는 싱글톤이며 이미 구성된 서비스를 다시 bindService()로 실행하면 객체가 매번 생성되지 않고 onBind() 함수만 반복 호출된다.

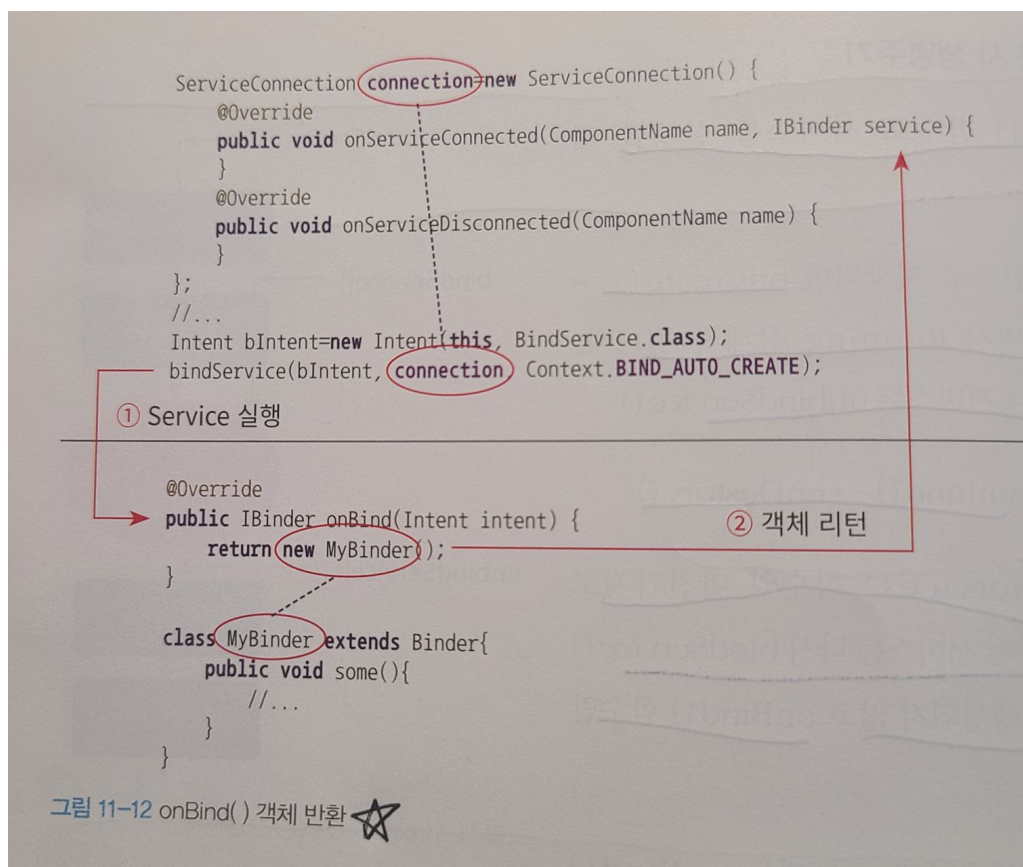
bindService() 함수로 인해 호출되는 onBind() 함수는 다른 생명주기 함수와 다르게 반환값이 있다.

onBind() 함수에서 반환할 객체는 Binder를 상속받아 개발자가 만든 클래스의 객체이다. Binder를 상속받아야 한다는 규칙 외에 클래스 내부의 함수 구현 규칙은 없고, 개발자가 임의로 함수를 정의하면 된다. 이 클래스의 객체를 생성해서 onBind() 함수에서 반환하면 서비스가 bindService() 함수로 실행한 곳에 전달된다. 함수 이름처럼 서비스가

실행되면서 onBind() 함수에서 반환한 객체가 서비스를 실행한 곳에 바인딩 되는 방법인 것이다. onBind() 함수에서 반환한 객체는 bindService() 함수의 매개변수로 지정한 ServiceConnection 구현 객체의 onServiceConnected() 함수의 매개변수로 전달된다. bindService() 함수로 서비스를 실행하면 IBinder를 구현한 객체가 바인딩 된다.

```
@Override
public IBinder onBind(Intent intent) {
    return new MyBinder();
}

class MyBinder extends Binder {
    public void some() {
        // ...
    }
}
```



bindService() 함수로 실행된 서비스는 자신에게 바인드된 컴포넌트가 하나도 없으면 자동으로 종료된다. 즉, 하나의 액티비티가 bindService() 함수로 서비스를 실행한 것이라면, 해당 액티비티가 종료되는 시점이나, unbindService() 함수로 종료를 명시하는 시점에 서비스도 함께 종료된다. 물론 여러 개의 액티비티가 bindService() 함수로 서비스를 이용하고 있다면, 하나의 액티비티가 종료되거나 unbindService() 함수를 호출하더라도 종료되지 않고 계속 움직이게 된다.

▶ Messenger 바인딩: bindService() 함수로 서비스를 실행하면 onBind() 함수에서 서비스로 구동시킨 쪽에 객체를 넘겨주어야 하는데, 이때 이 객체를 개발자가 직접 준비한 클래스의 객체가 아니라, Messenger 클래스를 이용해서 Messenger 객체를 반환시켜 바인딩하는 방법도 있다.

우선 Service에 Handler를 상속받은 클래스를 준비한다. 서비스를 실행한 곳(외부, 액티비티)에서 데이터를 넘기면, 이 클래스의 handleMessage() 함수가 자동 콜돼서 데이터를 받아서 처리한다.

```

public class PlayService extends Service {

    class MessengerHandler extends Handler {
        @Override
        // Message msg가 외부에서 넘긴 데이터
        public void handleMessage(@NonNull Message msg) {
            // 만약 1번 데이터이면
            if (msg.what == 1) {
                // 실제 데이터(Object 타입)을 필요한 데이터 타입으로 캐스팅해서 띄우기
                Toast.makeText(getApplicationContext(), "what==1 : " + (String)msg.obj,
            } else if (msg.what == 2) {
                Toast.makeText(getApplicationContext(), "what==2 : " + (String)msg.obj,
            } else {
                super.handleMessage(msg)
            }
        }
    }

    @Override
    public IBinder onBind(Intent intent) {
        Messenger messenger = new Messenger(new MessageHandler());
        // 메신저의 생성자로 데이터를 받아 처리할 Handler를 구현한 객체를 넘겨주기
        return messenger.getBinder(); // 서비스를 실행시킨 곳에 Messenger 객체 바인딩 (주기)
    }
}

```

서비스를 실행한 곳에서 데이터를 전달하면, handleMessage() 함수가 자동으로 호출된다. 이때 외부에서 전달한 데이터는 Message 타입이며, Message의 int 타입인 what 변수로 어떤 성격의 데이터인지를 구분하고 (1번 데이터인지, 2번 데이터인지...) Object 타입의 obj 변수로 전달된 데이터(실제 데이터)를 획득한다.

Messenger을 이용하기 위해, onBind() 함수의 반환값으로 Messenger 객체를 생성하면서 생성자 매개변수로 Handler을 구현한 객체를 지정한다. 그리고 Messenger 객체를 onBind() 함수의 결과값으로 반환한다. 그러면 서비스를 실행한 곳에 Messenger 객체가 전달된다.

```

public class MainActivity extends AppCompatActivity {
    Messenger messenger;

    ServiceConnection connection = new ServiceConnection() {
        // IBinder 객체는 서비스에서 넘어온 객체 (onBind()가 리턴한 Messenger 객체)
        @Override
        public void onServiceConnected(ComponentName componentName, IBinder service) {
            // 서비스에서 넘어온 객체를 매개변수로 메신저 생성, 초기화
            messenger = new Messenger(iBinder);
        }

        @Override
        public void onServiceDisconnected(ComponentName componentName) {

        }
    };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ActivityMainBinding binding = ActivityMainBinding.inflate(getLayoutInflater());
        setContentView(binding.getRoot());
        // ...
        Intent intent = new Intent(this, PlayService.class);
        bindService(intent, connection, Context.BIND_AUTO_CREATE);

        binding.button.setOnClickListener(view -> {
            try {
                Message message = new Message();
                message.what = 1;
                message.obj = "hello";
                messenger.send(message);
            } catch (Exception e) {
                e.printStackTrace();
            }
        });
    }
}

```

서비스를 이용하는 액티비티 코드다. bindService() 함수로 서비스를 실행하고, 서비스에서 넘어온 객체는 onServiceConnected() 함수의 매개변수로 받는다. 이렇게 서비스로부터 넘어온 객체를 Messenger의 생성자 매개변수에 지정만 해주면 된다. 그리고 이렇게 서비스를 실행한 후 서비스에 데이터를 전달하고 싶을 때는 Messenger의 send() 함수를 호출한다. send() 함수의 매개변수는 전달하는 데이터를 추상화한 Message 객체이다. 액티비티에서 send() 함수를 호출하는 순간, 서비스의 handleMessage() 함수가 자동으로 호출된다.

- MainActivity: bindService() 함수로 서비스 실행(구동)
- Service: onBind() 함수로 인해 Messenger 객체(IBinder) 반환 및 MainActivity에 전달
- MainActivity: onServiceConnected() 함수의 매개변수로 서비스에서 넘어온 객체 받기. 그 객체(IBinder)를 이용해서 Messenger 객체 생성 및 초기화.
- MainActivity: messenger.send(message)로 서비스에 데이터 전달.
- Service: Handler를 구현한 클래스의 handleMessage() 함수 자동 Call. 받은 데이터를 처리. onBind() 함수에서 Messenger 객체를 생성하면서 생성자 매개변수로 Handler를 구현한 객체를 지정하고, Messenger 객체 (IBinder)를 결과값으로 반환하여 서비스를 실행한 MainActivity에 Messenger 객체 전달.
- MainActivity: onServiceConnected() 함수의 매개변수로 서비스에서 넘어온 Messenger 객체 받기. 그 객체 (IBinder)를 Messenger의 생성자 매개변수에 지정하여 Messenger 객체 생성 및 초기화.

▶ 패키지 공개 상태: 앱에서 외부 앱의 패키지 정보에 접근하려면 AndroidManifest.xml 파일에 접근하고자 하는 외부 앱의 패키지명을 명시적으로 선언해야 한다.

외부 앱의 service를 앱의 액티비티에서 bindService()로 연동할 경우(이용할 경우), 패키지 정보를 지정해 주어야 한다. 외부 앱을 연동하려면 패키지 정보에 접근하기 위해 패키지 공개 상태(package visibility)라는 개념이 필요하다. 패키지 공개 상태란 AndroidManifest.xml 파일에 접근하고자 하는 외부 앱의 패키지 정보를 명시적으로 선언하는 것을 의미한다. 대표적으로 다음의 함수를 이용해서 패키지 정보에 접근하고, 이 경우 AndroidManifest.xml 파일에 외부 앱의 패키지명을 등록(패키지 공개 상태를 설정)하는 것이 필요하다.

- ☞ PackageManager.getPackageInfo()
- ☞ PackageManager.queryIntentActivities()
- ☞ Intent.resolveActivity()
- ☞ PackageManager.getInstalledPackages()
- ☞ PackageManager.getInstalledApplications()
- ☞ bindService()

```
// AndroidManifest.xml

<manifest >
    <queries>
        <package android:name="com.example.test4_outter" />
    </queries>
</manifest>
```

<queries> 태그 하위에 <package> 태그를 이용하여 접근하고자 하는 앱의 패키지명을 명시적으로 선언한 후 사용한다. 하나 이상일 경우 <package> 태그로 계속해서 추가하면 된다. 만약 여러 앱의 패키지 정보에 접근해야 한다면, 위의 코드처럼 작성하지 않고 모든 외부 앱의 패키지 정보에 접근 허용을 해달라는 퍼미션을 선언해도 되지만, 권장사항은 <package> 태그를 이용하는 방식이다.

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

패키지 공개 상태 설정 없이 다음과 같은 코드를 실행하면 안드로이드 버전 11 이상부터는 android.content.pm.PackageManager\$NameNotFoundException: com.example.test4_outter 에러가 발생한다.

```
try {
    PackageInfo packageInfo = getPackageManager().getPackageInfo("com.example.test4_outter", 0);
    String versionName = packageInfo.versionName;
    Log.d("kkang", versionName);
} catch (Exception e) {
    e.printStackTrace();
}
```

● AIDL(Android Interface Definition Language): 안드로이드에서 프로세스 간의 통신을 목적으로 제공하는 기능이고, bindService()를 사용하는 기법이다. 외부 앱의 데이터 액세스는 콘텐츠 프로바이더 기법이고, AIDL은 두 앱 간의 프로세스가 각자 실행된 상태에서 프로세스 간의 통신을 통해 데이터를 주고받거나, 한 앱의 프로세스가

bindService를 통해 다른 앱의 프로세스의 원격 업무 로직을 이용하는 기법이다. AIDL은 프로세스 간의 통신이지만, 개발자가 직접 코드를 작성하지 않고 외부 앱의 함수 호출만으로 되게 하는 기법이다. bindService() 함수로 서비스 컴포넌트를 실행하여 객체를 공유하고, 공유된 객체의 함수를 호출하면서 프로세스 간의 통신을 수행한다.

원천 로직 업무를 구현해놓고 있는 한 앱의 프로세스를 Server Process로 칭하고, 그 앱의 원격 로직 업무를 이용하기 위한 다른 앱의 프로세스를 Client Process로 부르기로 한다.

1) 서버 프로세스/클라이언트 프로세스: aidl이 확장자인 파일을 만든다. 이 파일은 확장자만 aidl일뿐 내부는 완벽하게 자바의 인터페이스이다. 추상 함수만 선언한 파일로, 이 함수(들)은 서버 프로세스 쪽에서 호출하게 되는 함수이다. 그리고 이 aidl 파일은 클라이언트 프로세스 쪽에도 공유되어야 한다. 또한 만든 aidl 파일을 자바 코드에서 인식하게 하려면 Make Module 작업을 해주어야 한다. 모듈명을 클릭한 채로 Build > Make Module 메뉴를 눌러 Make Module 작업을 진행해야 한다.

```
interface IPlayService {  
    void start();  
    void stop();  
}
```

2) 서버 프로세스: 서비스를 만든다. 이 서비스의 onBind() 함수에서 AIDL에서 선언되어 있는 인터페이스를 구현하여 업무 로직을 가지는 객체를 준비한다. bindService() 함수로 실행되는 서비스이므로 onBind() 함수가 자동으로 호출되며, 이 함수에서 aidl 통신을 대행해주는 Stub 클래스를 자동으로 만들어 클라이언트 프로세스 쪽에 반환한다. bindService() 함수로 onBind() 함수가 호출되며 aidl 파일을 구현한 클래스 객체를 반환하지만, 실제로 넘어가는 것은 실제 업무 로직을 가지고 있는 객체가 아니라, 프로세스 업무를 대행해줄 수 있는 Stub 객체가 넘어가는 것이다. 이 Stub 객체는 우리가 직접 만드는 것이 아니다. Stub 객체 자체는 업무 로직을 가지지 않고 프로세스 간의 통신을 대행한다.

```
@Override  
public IBinder onBind(Intent intent) {  
    return new IPlayService.Stub() {  
  
        @Override  
        public void start() throws RemoteException {  
            // ...  
        }  
  
        @Override  
        public void stop() throws RemoteException {  
            // ...  
        }  
    }  
}
```

3) 클라이언트 프로세스: 서버 프로세스에서 준비한 서비스를 인텐트로 실행한다. bindService() 함수로 인텐트를 보내시킬 때, 꼭 대상이 되는 앱의 패키지명을 setPackage() 함수로 설정해야 한다. (바인드 서비스로 외부 앱을 이용할 때는 반드시 외부 앱의 패키지 명까지 주어야 한다). 또한 서비스를 이용하는 앱의 매니페스트 파일에는 패키지 공개 상태 개념을 위해 <package> 태그가 선언되어 있어야 한다.

```
Intent intent = new Intent("com.example.ACTION_PLAY");
intent.setPackage("com.example.test7_20._aidl");
bindService(intent, connection, Context.BIND_AUTO_CREATE);
```

```
// AndroidManifest.xml

<manifest >
    <queries>
        <package android:name="com.example.test7_20._aidl" />
    </queries>
</manifest>
```

4) 클라이언트 프로세스: 바인드 객체 획득 및 이용을 위해 ServiceConnection 인터페이스를 구현한 객체를 준비한다. bindService() 함수로 서비스가 실행되면 반환되는 Stub 객체를 onServiceConnected() 함수에서 받아 이용하게 된다. onServiceConnected() 함수 내에 IPlayService.Stub.asInterface(service); 구문이 서비스의 onBind() 함수에서 반환한 객체이다. 이렇게 객체를 획득한 다음 코드 어딘가에서 pService.함수이름으로 함수를 호출하면, 서비스 쪽 프로세스의 함수가 실행된다.

```
ServiceConnection connection = new ServiceConnection() {
    @Override
    public void onServiceConnected(ComponentName name, IBinder service) {
        pService = IPlayService.Stub.asInterface(service);
        // ...
    }

    @Override
    public void onServiceDisconnected(ComponentName name) {
        pService = null;
    }
}
```

1. playmusic module

package name: com.android.playmusic

```
// aidl 파일 만들기: 모듈명 우클릭 > New > Folder > AIDL Folder > New > AIDL > AIDL File
// 방금 만든 aidl 파일을 자바 코드에서 인식하려면 Make Module 작업을 해주어야 한다.
// 모듈명을 클릭한 채로 Build > Make Module 메뉴를 눌러 Make Module 작업을 진행해야 한다.
// MusicService.aidl
```

```
package com.android.playmusic;

interface MusicService {
    void startMusic();
    void stopMusic();
}
```



```

// 서비스 만들기: New > Service > Service
// PlayMusicService.java

package com.android.playmusic;

import android.app.Service;
import android.content.Intent;
import android.media.MediaPlayer;
import android.os.IBinder;
import android.os.RemoteException;

public class PlayMusicService extends Service {
    MediaPlayer player;

    @Override
    public void onDestroy() {
        player.release();
        super.onDestroy();
    }

    @Override
    public IBinder onBind(Intent intent) {
        // aidl 파일을 구현한 객체를 만들고,
        // 이 객체로 외부 프로세스에서 통신할 Stub를 만들어서 반환
        return new MusicService.Stub() {
            @Override
            public void startMusic() throws RemoteException {
                player = MediaPlayer.create(PlayMusicService.this, R.raw.music);
                try {
                    player.start();
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }

            @Override
            public void stopMusic() throws RemoteException {
                if (player.isPlaying()) {
                    player.stop();
                }
            }
        };
    }
}

```

```
// 서비스 외부에서 암시적 인텐트로 실행할 수 있게 <intent-filter> 추가
// AndroidManifest.xml

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.android.playmusic">

    <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.PracticeAIDL">

        <service
            android:name=".PlayMusicService"
            android:enabled="true"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.ACTION_PLAY" />
            </intent-filter>
        </service>

        <activity
            android:name=".MainActivity"
            android:exported="true">
        </activity>
    </application>

</manifest>
```

2. PracticeAIDL 모듈

```
// aidl 파일 만들기: 모듈명 우클릭 > New > Folder > AIDL Folder > New > AIDL > AIDL File
// 이때 주의할 점은, aidl 파일이 playmusic module의 aidl 파일이 있는 패키지명과 같게 지정해야
// 패키지명: com.android.playmusic.으로 패키지를 만든 다음에, playmusic module의 aidl 파일을
// 복사해서 붙여넣기 한다. 그리고 Build > Make Module 한다.
```

```
package com.android.playmusic;

interface MusicService {
    void startMusic();
    void stopMusic();
}
```

```
// AndroidManifest.xml
// 외부 앱의 패키지 정보에 접근하기 위해 패키지 정보 추가
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.android.practiceaidl">
    <queries>
        <package android:name="com.android.playmusic" />
    </queries>
// ...
</application>

</manifest>
```

```
// MainActivity.java
package com.android.practiceaidl;

import androidx.appcompat.app.AppCompatActivity;

import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.Bundle;
import android.os.IBinder;

import com.android.playmusic.MusicService;
import com.android.practiceaidl.databinding.ActivityMainBinding;

public class MainActivity extends AppCompatActivity {

    MusicService pService;
    ActivityMainBinding binding;

    ServiceConnection connection = new ServiceConnection() {
        @Override
        public void onServiceConnected(ComponentName name, IBinder service) {
            pService = MusicService.Stub.asInterface(service);
            // play 이미지 버튼 활성화
            binding.play.setEnabled(true);
        }

        @Override
        public void onServiceDisconnected(ComponentName name) {
            pService = null;
        }
    };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        binding = ActivityMainBinding.inflate(getLayoutInflater());
        setContentView(binding.getRoot());

        // 서비스 구동
        Intent intent = new Intent("android.intent.ACTION_PLAY");
        intent.setPackage("com.android.playmusic");
        bindService(intent, connection, Context.BIND_AUTO_CREATE);
    }
}
```

```

binding.play.setOnClickListener(view -> {
    // 서비스가 구동되었으면
    if (pService != null) {
        try {
            pService.startMusic();
            // play 버튼은 비활성화하고, stop 버튼은 활성화하고
            binding.play.setEnabled(false);
            binding.stop.setEnabled(true);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
});

binding.stop.setOnClickListener(view -> {
    if (pService != null) {
        try {
            pService.stopMusic();
            binding.play.setEnabled(true);
            binding.stop.setEnabled(false);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
});

}

@Override
protected void onDestroy() {
    super.onDestroy();
    unbindService(connection);
}
}

```

```
// activity_main.xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#146d64">

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/background"
        android:layout_alignParentTop="true" />

    <ImageView
        android:id="@+id/play"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/ic_play"
        android:layout_alignParentBottom="true"
        android:layout_marginBottom="36dp"
        android:layout_marginLeft="24dp"
        android:clickable="true" />

    <TextView
        android:id="@+id/lab2_title"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Play Music"
        android:textColor="@android:color/white"
        android:textStyle="bold"
        android:textSize="30dp"
        android:layout_alignParentBottom="true"
        android:layout_alignTop="@id/play"
        android:layout_marginLeft="16dp"
        android:layout_centerHorizontal="true"
        android:layout_centerInParent="true"
        android:layout_centerVertical="true"
    />

    <ImageView
        android:id="@+id/stop"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/ic_stop"
        android:layout_alignParentBottom="true"
        android:layout_alignParentRight="true"
        android:layout_marginBottom="36dp"
        android:layout_marginRight="24dp"
        android:clickable="true" />

</RelativeLayout>
```

Resources: 강쌤의 안드로이드 프로그래밍