

손전등 앱 만들기

▶ 구성요소

- ▶ CameraManager : 플래시를 켜는 기능을 제공하는 클래스
- ▶ Service : 화면이 없고 백그라운드에서 실행되는 컴포넌트
- ▶ AppWidget : 런처에 배치하여 빠르게 앱 기능을 쓸 수 있게 하는 컴포넌트

▶ 라이브러리 설정

- ▶ Anko 라이브러리 : 인텐트, 다이얼 로그, 로그 등을 효율적으로 구현하게 해주는 라이브러리

손전등 앱 만들기

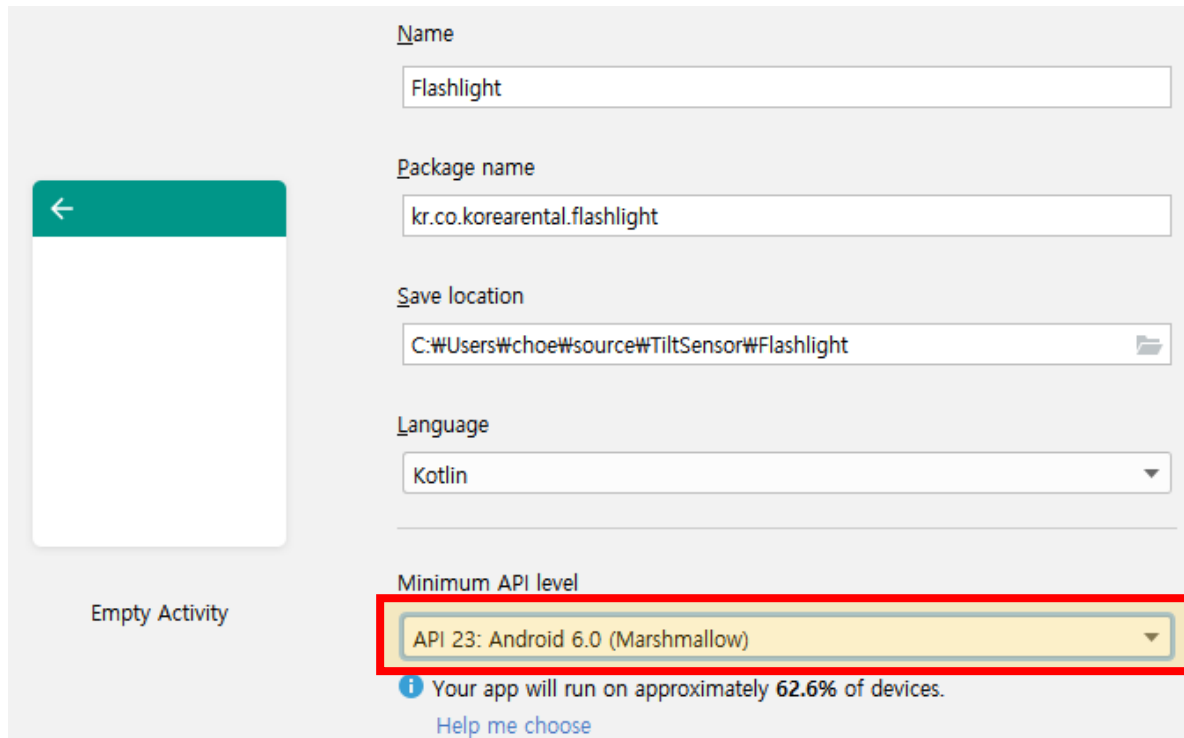
▶ 구현 순서

- ▶ 손전등 기능 구현
- ▶ 액티비티에서 손전등을 제어하도록 구현
- ▶ 서비스에서 손전등 기능을 사용할 수 있도록 구현
- ▶ 앱 위젯에서 손전등 기능을 사용하도록 구현

손전등 앱 만들기

▶ 프로젝트 생성

- ▶ 프로젝트 명 : Flashlight
- ▶ minSdkVersion : 23(안드로이드 6.0 Marshmallow)
- ▶ 기본 액티비티 : Empty Activity
- ▶ 프로젝트 생성한 후 아이콘 변경 : kr_flashlight.png



The screenshot shows the 'New Project' dialog in Android Studio. On the left, there is a preview of an 'Empty Activity' with a teal header bar containing a back arrow. The right side contains the following fields:

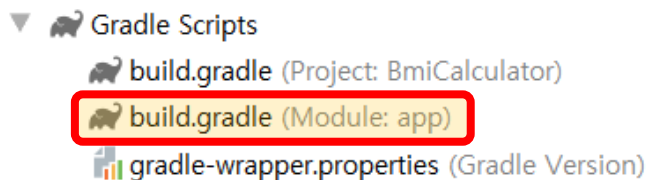
- Name:** Flashlight
- Package name:** kr.co.korearental.flashlight
- Save location:** C:\Users\choe\source\TiltSensor\Flashlight
- Language:** Kotlin
- Minimum API level:** API 23: Android 6.0 (Marshmallow) (highlighted with a red box)

Below the API level dropdown, there is an information icon and the text: 'Your app will run on approximately 62.6% of devices.' with a link 'Help me choose'.

손전등 앱 만들기

▶ Anko 라이브러리 추가

- ▶ 프로젝트 창에서 모듈 수준의 build.gradle 파일에 anko 라이브러리 추가



- ▶ dependencies 항목에 anko 라이브러리를 추가

▷ implementation "org.jetbrains.anko:anko-commons:\$anko_version"

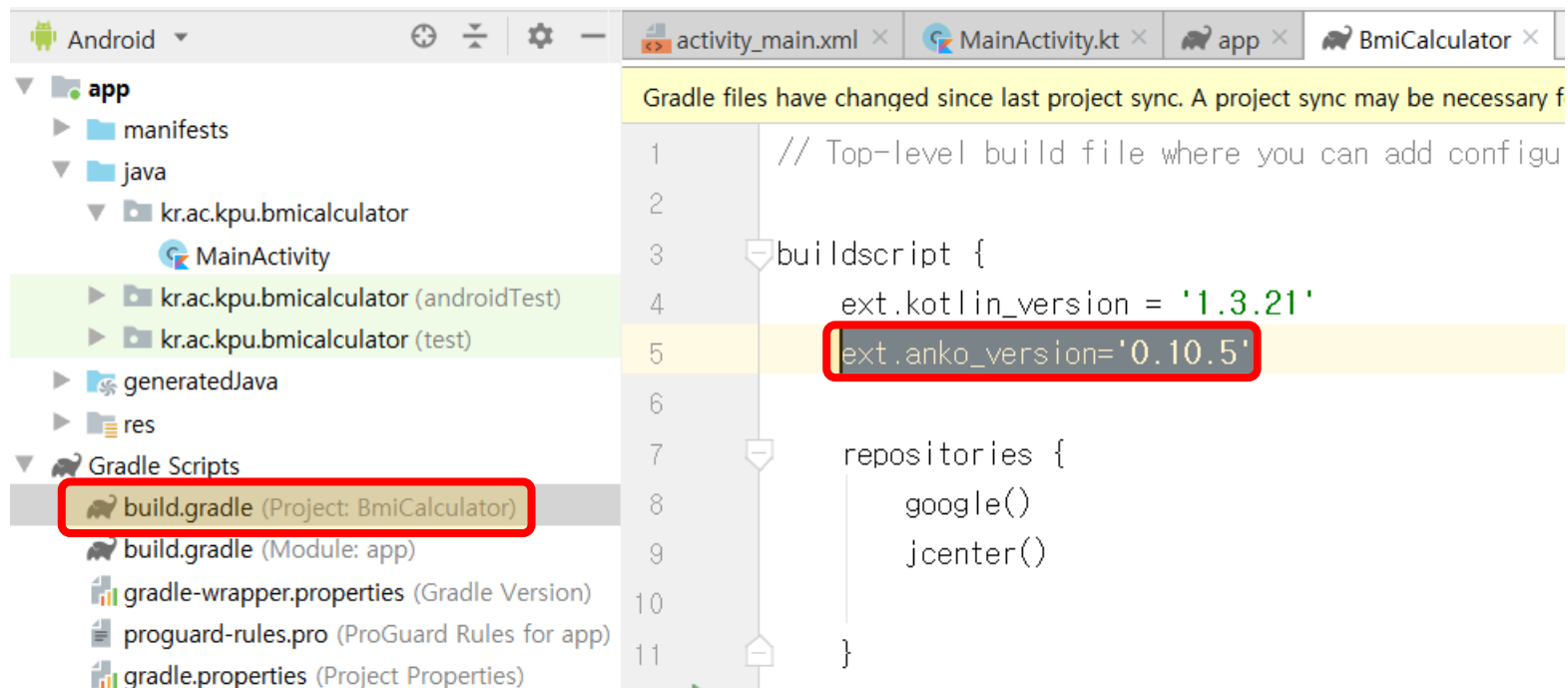
```
25  dependencies {  
26      implementation "org.jetbrains.anko:anko-commons:$anko_version"  
27  }  
28  implementation fileTree(dir: 'libs', include: ['*.jar'])  
29  implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk7:$kotlin_version"  
30  implementation 'com.android.support:appcompat-v7:28.0.0'  
31  implementation 'com.android.support.constraint:constraint-layout:1.1.3'  
32  testImplementation 'junit:junit:4.12'  
33  androidTestImplementation 'com.android.support.test:runner:1.0.2'  
34  androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.2'
```

손전등 앱 만들기

▶ Anko 라이브러리 추가

▶ 프로젝트 수준의 build.gradle Anko 라이브러리 버전을 지정

▷ ext.anko_version='0.10.5'



The screenshot shows the Android Studio interface. On the left, the 'Project' view displays the project structure for 'BmiCalculator'. The 'Gradle Scripts' folder is expanded, and 'build.gradle (Project: BmiCalculator)' is selected and highlighted with a red box. On the right, the 'build.gradle' file is open in the editor. A yellow banner at the top of the editor states: 'Gradle files have changed since last project sync. A project sync may be necessary f'. The code in the editor is as follows:

```
1 // Top-level build file where you can add configura
2
3 buildscript {
4     ext.kotlin_version = '1.3.21'
5     ext.anko_version = '0.10.5'
6
7     repositories {
8         google()
9         jcenter()
10
11     }
```

The line `ext.anko_version = '0.10.5'` on line 5 is highlighted with a red box.

손전등 앱 만들기

▶ 플래시 구현 환경

- ▶ 플래시를 다루는 방법은 안드로이드 6.0(sdk23)이상에서 제공하는 방법이 가장 간단함
- ▶ 공식적으로 5.0에서도 플래시를 켤 수 있지만 코드가 매우 복잡함
- ▶ 5.0 미만에서는 제조사마다 다른 방법을 사용해야 함

▶ 플래시 관련 안드로이드 API

방법1 (Camera API 이용하는 방법)

- 장점: 안드로이드 구버전도 지원 가능, 구현 용이
- 단점: 카메라 권한 필요, deprecated API, 느린 속도

방법2 (Camera2 API 이용하는 방법)

- 장점: 최신 Camera API
- 단점: 카메라 권한 필요, 구현 복잡, 안드로이드 5.0 (API Level 21) 이상부터 가능

방법3 (Flashlight API 이용하는 방법)

- 장점: 카메라 권한 불필요, 구현 매우 용이, 빠른 속도
- 단점: 안드로이드 6.0 (API Level 23) 이상부터 가능

손전등 앱 만들기

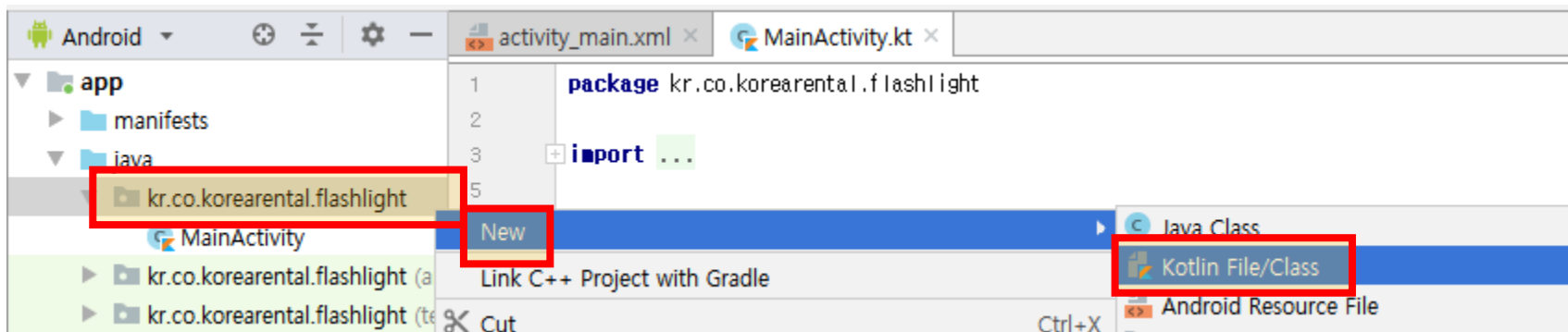
▶ 손전등 기능 구현

▶ 손전등의 핵심은 카메라 플래시를 동작시키는 것이므로 CameraManager 클래스를 사용

▶ 손전등 기능을 Torch 클래스에 작성

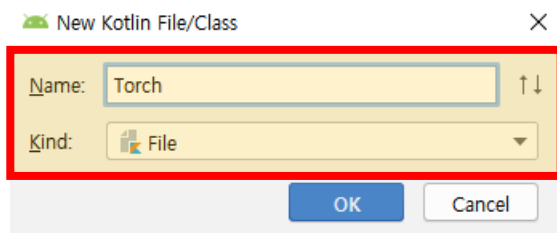
▶ 손전등의 기능을 별도의 Torch 클래스로 분리하여 작성

▶ 패키지명에서 마우스 우 클릭 - New - Kotlin File/Class



▶ Name : Torch

▶ Kind : Class



손전등 앱 만들기

▶ Torch 클래스 구현

- ▶ context의 getSystemService() 메서드는 안드로이드 시스템에서 제공하는 각종 서비스를 관리하는 매니저 클래스 생성
 - ▷ 인자로 Context클래스에 정의된 서비스를 정의한 상수를 지정
 - ▷ 이 메서드는 Object형을 반환하므로 as연산자를 사용하여 CameraManager형으로 형 변환
- ▶ getCameraId()메서드는 카메라의 ID를 얻는 메서드
 - ▷ 카메라가 없다면 ID가 null이기 때문에 cameraId 프로퍼티를 null허용으로 선언

```
class Torch(context: Context) {  
    플래시를 제어하기 위하여 CameraManager 객체가 필요하고  
    이를 얻으려면 Context 객체가 필요 / 생성자로 Context 받음  
    private var cameraId: String? = null 카메라를 켜고 끌때 카메라 ID가 필요  
    private val cameraManager = context.getSystemService(Context.CAMERA_SERVICE) as CameraManager  
    카메라를 제어할 수 있도록 CameraManager를 가져옴  
  
    init {  
        | cameraId = getCameraId() 카메라 ID를 가져오는 메서드 / 카메라 ID는 기기에 내장된 카메라 마다 고유한 ID를 보유  
    }  
}
```


손전등 앱 만들기

▶ Torch 클래스 구현(계속)

- ▶ Torch 클래스는 플래시를 켜는 flashOn() 메서드와 플래시를 끄는 flashOff() 메서드를 제공

```
fun flashOn() {  
    cameraManager.setTorchMode(cameraId, enabled: true)  
}
```

```
fun flashOff() {  
    cameraManager.setTorchMode(cameraId, enabled: false)  
}
```

손전등 앱 만들기

▶ Torch 클래스 구현(계속)

▶ getCameraId()메서드는 카메라의 ID를 얻는 메서드

▶ 카메라가 없다면 ID가 null이기 때문에 반환 값은 String?로 지정

```
private fun getCameraId(): String? {  
    val cameraIds = cameraManager.cameraIdList  
    for (id in cameraIds) {  
        val info = cameraManager.getCameraCharacteristics(id)  
        val flashAvailable = info.get(CameraCharacteristics.FLASH_INFO_AVAILABLE)  
        val lensFacing = info.get(CameraCharacteristics.LENS_FACING)  
        if (flashAvailable != null  
            && flashAvailable  
            && lensFacing != null  
            && lensFacing == CameraCharacteristics.LENS_FACING_BACK) {  
            return id  
        }  
    }  
    return null  
}
```

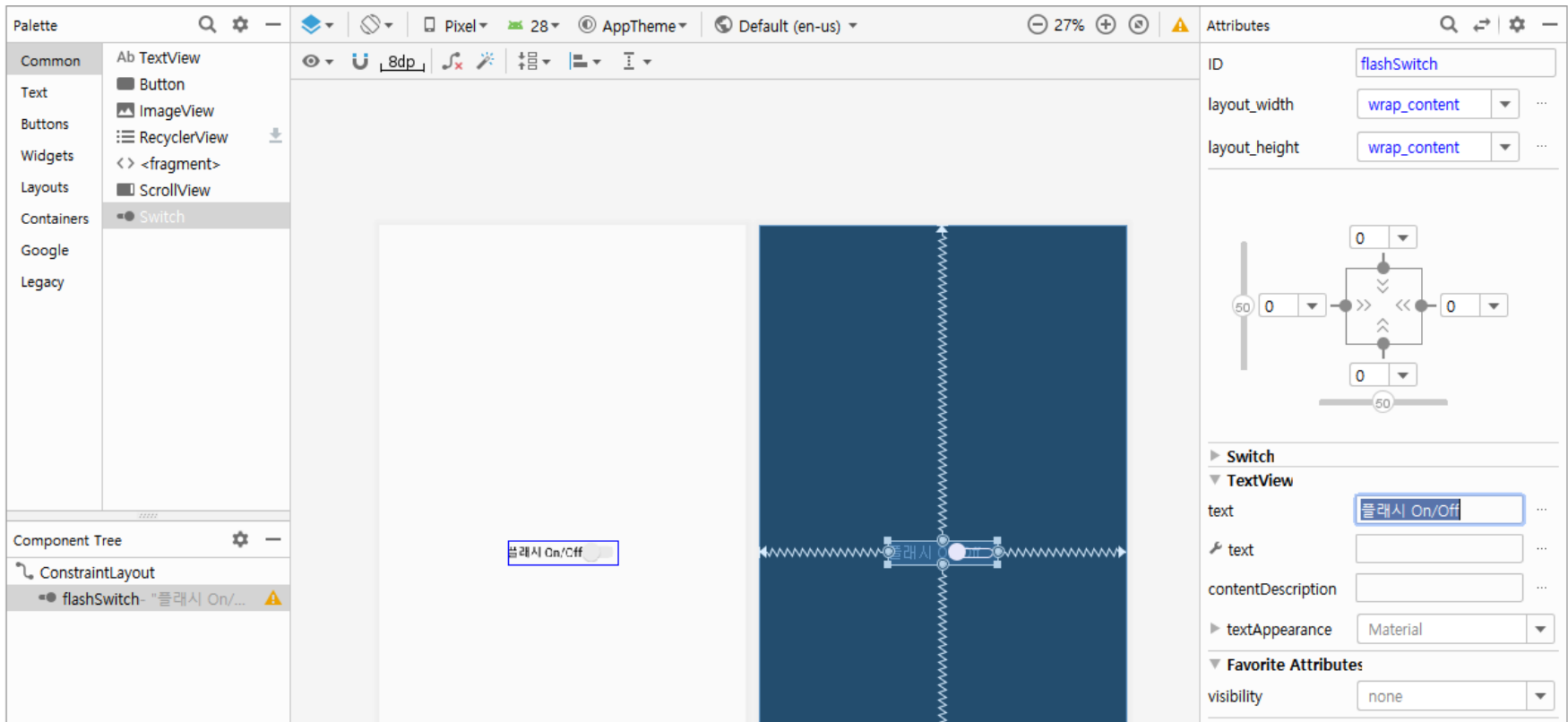
CameraManager는 기기가 가지고 있는 모든 카메라에 정보 목록을 제공

이 목록을 순화하면서 각 ID별로 세부 정보를 가지는 객체를 얻음
이 객체로부터 플래시 가능 여부와 카메라의 렌즈 방향을 알 수 있음

플래시가 사용 가능하고 카메라가 기기의 뒷면을 향하는 카메라의 ID를 찾았다면 ID 값을 반환, 찾지 못했다면 null을 반환

손전등 앱 만들기

- ▶ 액티비티에서 손전등 기능을 사용하도록 화면 작성
 - ▶ 기본 텍스트 뷰를 삭제하고 스위치만 추가
 - ▶ Autoconnect Mode 활성화 - 제약을 자동으로 생성



손전등 앱 만들기

▶ 스위치(Switch)



- ▶ 켜거나 끄는 두 가지 상태 값만 가지는 버튼
- ▶ `setOnCheckedChangeListener`를 구현하면 상태가 변경되었을 때의 처리를 수행
- ▶ `buttonView` 인자는 상태가 변경된 Switch 객체 자신
- ▶ `isChecked` 인자는 On/Off 상태를 boolean으로 전달
 - ▶ 이 정보로 On/Off에 따라 처리 가능

```
flashSwitch.setOnCheckedChangeListener { buttonView, isChecked ->
    if (isChecked) {
        //On일 때 동작
    } else {
        //Off일 때 동작
    }
}
```

손전등 앱 만들기

▶ MainActivity.kt 코드 작성

▶ 앞의 설명을 참고하여 플래시를 제어하는 코드 작성

```
override fun onCreate(savedInstanceState: Bundle?) {
```

```
    super.onCreate(savedInstanceState)
```

```
    setContentView(R.layout.activity_main)
```

```
    val torch = Torch(context, this) 작성한 Torch 클래스를 인스턴스화
```

```
    flashSwitch.setOnCheckedChangeListener { _, isChecked ->
```

```
        if (isChecked) {
```

```
            torch.flashOn()
```

스위치가 켜지면 flashOn() 메서드를 호출하여 플래시를 켜고
스위치가 꺼지면 flashOff() 메서드 호출하여 플래시 끄

```
        } else {
```

```
            torch.flashOff()
```

```
        }
```

```
    }
```

```
}
```

▶ 실행 확인!

손전등 앱 만들기

▶ 람다식

- ▶ 함수를 간단히 표현하는 방법이지만 디버깅이 어렵고 코드의 가독성이 떨어지므로 주의해서 사용
- ▶ 두 수를 전달 받아 더한 값을 반환하는 함수

```
fun add(x : Int, y : Int){  
    return x + y  
}
```

- ▶ 코틀린에서는 간단한 메소드를 반환형, 블록{}, return 생략하고 식처럼 함수를 만들 수 있음

```
fun add(x : Int, y : Int) = x + y
```

- ▶ 람다식은 다음과 같이 항상 중괄호로 둘러싸여 있음
- ▶ 인수 목록을 나열하고 -> 이후에 본문이 위치
- ▶ 람다식을 변수에 저장할 수 있고 이러한 변수는 일반 함수처럼 사용할 수 있음

```
var add = {x : Int, y : Int -> x + y}  
println(add(2,5)) //7
```

손전등 앱 만들기

▶ SAM 변환

▶ Single Abstract Method

- ▶ 자바로 작성된 메서드가 하나인 인터페이스를 구현하는 대신 함수를 작성하는 것
- ▶ 안드로이드에서는 버튼의 클릭 이벤트를 구현할 때 `onClick()` 추상 메소드 하나만을 갖는 `View.OnClickListener` 인터페이스를 구현
- ▶ 아래 예제는 위의 내용을 익명클래스로 작성한 코드

```
button.setOnClickListener(object : View.OnClickListener{  
    override fun onClick(v:View?){  
        //클릭 시 처리할 내용  
    }  
})
```

▶ 위와 같이 구현하는 메소드가 하나인 경우 람다식으로 변경가능

```
button.setOnClickListener({v:View? ->  
    //클릭 시 처리할 내용  
})
```

손전등 앱 만들기

▶ SAM 변환

```
button.setOnClickListener({v:View? ->
    //클릭 시 처리할 내용
})
```

▶ 메서드 호출 시 전달되는 인수가 람다식인 경우 괄호 밖으로 뺄 수 있음

```
button.setOnClickListener(){v:View? ->
    //클릭 시 처리할 내용
}
```

▶ 람다가 어떤 메서드의 유일한 인수인 경우에는 메서드의 괄호를 생략

```
button.setOnClickListener {v:View? ->
    //클릭 시 처리할 내용
}
```

▶ 컴파일러가 자료형의 추론이 가능한 경우에는 자료형을 생략

```
button.setOnClickListener {v ->
    //클릭 시 처리할 내용
}
```


손전등 앱 만들기

▶ SAM 변환

- ▶ 만약에 메소드 내부에서 인자를 사용하지 않는다면 _ 로 표기하여 잘못 사용하는 실수를 방지

```
button.setOnClickListener {_ ->
    //클릭 시 처리할 내용
}
```

- ▶ 또한 람다식에서 인수가 하나인 경우에는 인자를 생략하고 람다 블록내에서 인수를 it으로 접근 가능하며 아래 코드에서 it은 View? 타입의 v를 의미

```
button.setOnClickListener {
    it.visibility = View.GONE
}
```

손전등 앱 만들기

▶ 소스코드 분석

```
flashSwitch.setOnCheckedChangeListener { _, isChecked ->
    if (isChecked) {
        startService(intentFor<TorchService>().setAction("on"))
    } else {
        startService(intentFor<TorchService>().setAction("off"))
    }
}
```

▶ 위 코드는 아래 코드의 형식으로 표현됨

```
button.setOnClickListener {v ->
    //클릭 시 처리할 내용
}
```

손전등 앱 만들기

▶ 소스코드 분석

- ▶ `setOnCheckedChangeListener` 메소드는 `CompoundButton.OnCheckedChangeListener` 인터페이스의 객체를 인자로 받음

`setOnCheckedChangeListener`

Added in API level 1

```
public void setOnCheckedChangeListener (CompoundButton.OnCheckedChangeListener listener)
```



Register a callback to be invoked when the checked state of this button changes.

Parameters

`listener`

`CompoundButton.OnCheckedChangeListener`: the callback to call on checked state change
This value may be `null`.

손전등 앱 만들기

▶ 소스코드 분석

▶ `OnCheckedChangeListener`는 `onCheckedChanged`라는 추상메서드 하나를 갖고 있으며 해당 메서드는 두개의 인자를 받음

▶ `compoundButton` `buttonView`, `boolean` `isChecked`

Android Developers > Docs > 참조

☆☆☆☆☆

CompoundButton.OnCheckedChangeListener

Added in API level 1

```
public static interface CompoundButton.OnCheckedChangeListener
    android.widget.CompoundButton.OnCheckedChangeListener
```

Interface definition for a callback to be invoked when the checked state of a compound button changed.

Summary

Public methods

| | |
|----------------------------|--|
| <code>abstract void</code> | <code>onCheckedChanged(CompoundButton buttonView, boolean isChecked)</code> Called when the checked state of a compound button has changed. |
|----------------------------|--|

손전등 앱 만들기

▶ 소스코드 분석

▶ OnCheckedChangeListener는 SAM으로 변환된 람다식으로 처리

```
flashSwitch.setOnCheckedChangeListener { _, isChecked ->
    if (isChecked) {
        startService(intentFor<TorchService>().setAction("on"))
    } else {
        startService(intentFor<TorchService>().setAction("off"))
    }
}
```

손전등 앱 만들기

▶ 안드로이드 서비스 개요

- ▶ 서비스란 안드로이드의 4대 컴포넌트 중 하나
- ▶ 화면이 없고 백그라운드에서 수행하는 작업을 작성하는 컴포넌트
- ▶ 바운드된 서비스와 바운드되지 않은 서비스로 구분

▶ 서비스의 생명주기

- ▶ 액티비티와 같이 생명주기용 콜백 메소드가 존재
- ▶ 서비스를 시작하면 onCreate() 메서드가 호출
- ▶ 이후에 onStartCommand() 메서드가 호출
 - ▶ 여기서 서비스의 동작을 코드를 작성
- ▶ 서비스가 종료되면 onDestroy() 메서드가 호출됨

손전등 앱 만들기

▶ 서비스 생명주기 메소드

▶ 서비스는 바운드된 서비스와 바운드 되지 않은 서비스로 구분 -

▶ onCreate()

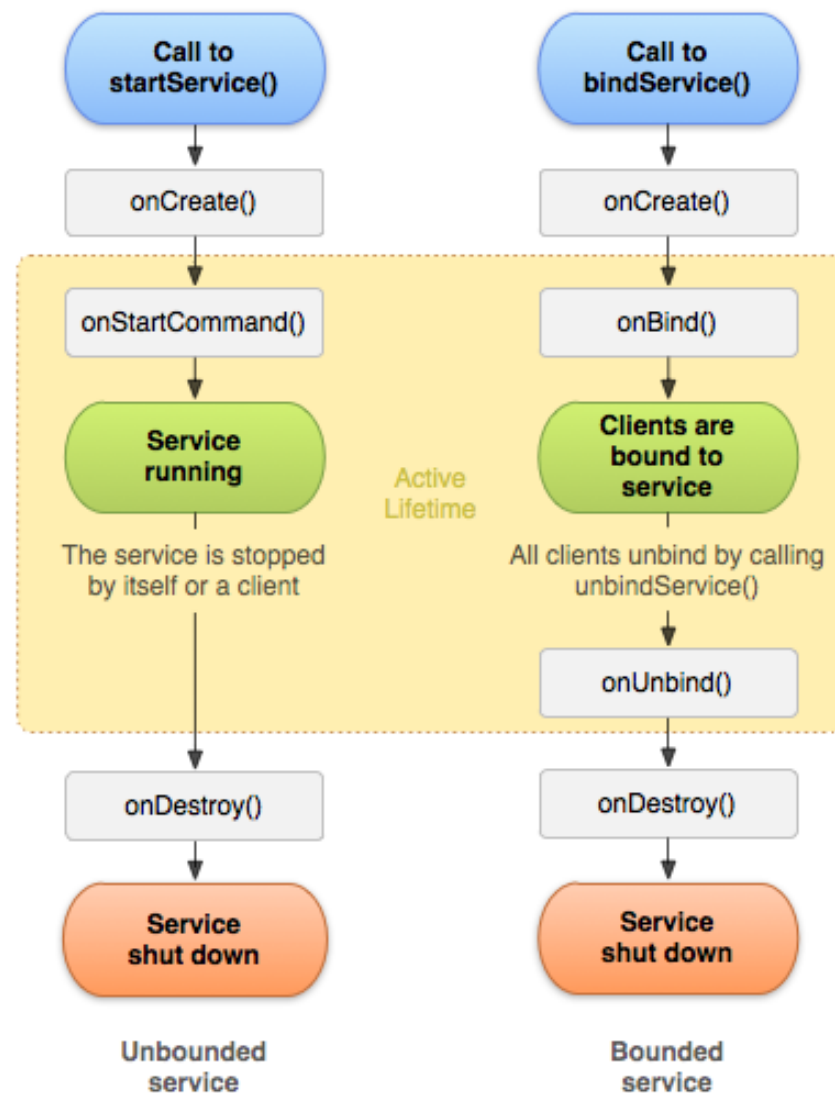
▶ 서비스가 생성될 때 호출되는 콜백 메서드, 초기화 등을 수행

▶ onStartCommand()

▶ 서비스가 액티비티와 같은 다른 컴포넌트로 부터 startService() 메서드로 호출되면 불리는 콜백 메서드, 실행할 작업을 여기에 작성

▶ onDestroy()

▶ 서비스 내부에서 stopSelf()를 호출하거나 외부에서 stopService()로 서비스를 종료하면 호출됨



손전등 앱 만들기

▶ 손전등 프로젝트에서 서비스 활용

- ▶ 플래시를 켜는 기능에 화면이 꼭 필요하지는 않기 때문에 액티비티는 단순히 플래시를 켜고 끄는 인터페이스만 제공하고 서비스에서 플래시를 제어

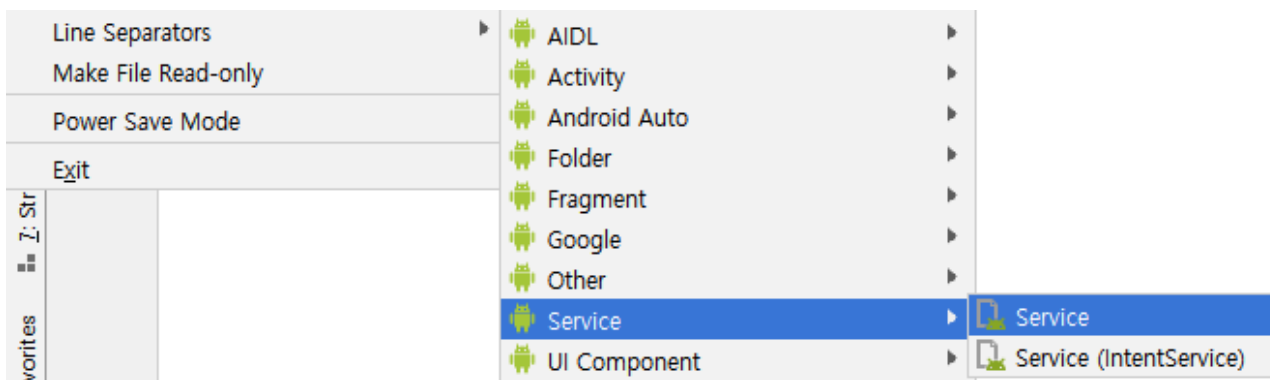
 - ▶ 액티비티는 서비스를 호출하는 기능

- ▶ 추후에 작성하는 위젯도 서비스를 호출해서 플래시 제어

▶ 손전등 기능을 구현할 서비스 클래스 생성

- ▶ file - new - service - service

- ▶ 액티비티에서 구현된 기능을 서비스에서 처리하기 위하여

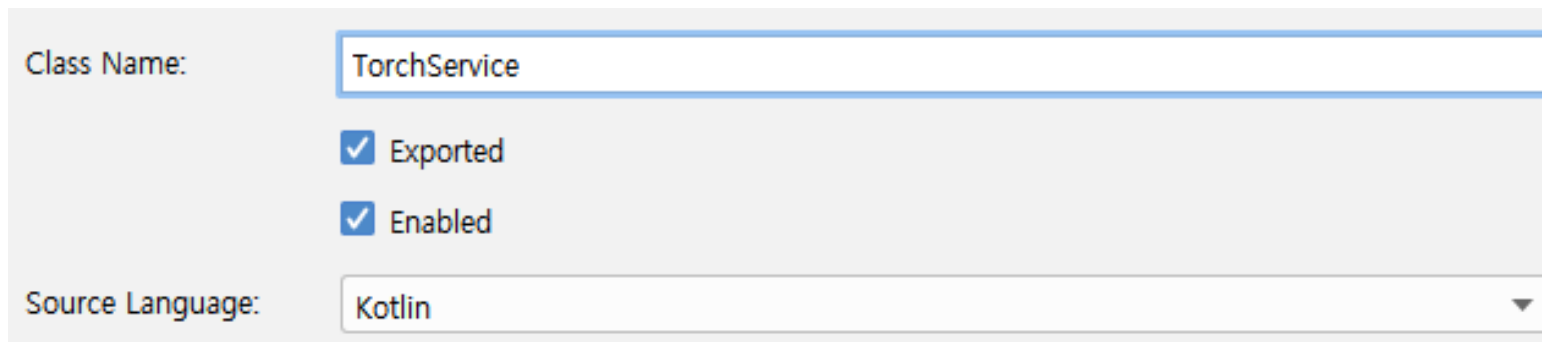


손전등 앱 만들기

▶ 서비스 클래스 생성

▶ 클래스 명 : TorchService

▶ 다른 항목은 기본으로 설정 후 Finish클릭



Class Name: TorchService

☒ Exported

☒ Enabled

Source Language: Kotlin

▶ TorchService 파일에서 서비스를 호출하면 실행되는 onStartCommand()를 오버라이드

▶ 에디터 중간에 onStartCommand의 몇 글자만 입력해도 자동 완성 기능이 제공되므로 enter키를 눌러서 오버라이드 메서드 완성



```
onSta  
override fun onStartCommand(intent: Intent?, flags: Int, startId: Int):
```

손전등 앱 만들기

▶ TorchService 클래스 작성

```
class TorchService : Service() { Service 클래스를 상속받음
    private val torch: Torch by lazy {
        Torch(context, this) TorchService 서비스가 Torch 클래스를 사용하여 플래시를 제어
    }
    외부에서 startService() 메서드로 TorchService 서비스를 호출하면
    onStartCommand() 콜백 메서드 호출

    override fun onStartCommand(intent: Intent?, flags: Int, startId: Int): Int {
        when (intent?.action) { 보통 인텐트에 action 값을 설정하여 호출하는데
            // 앱에서 실행할 경우 "on " 과 "off" 문자열을 액션으로 받았을 때
            "on" -> { When문을 사용하여 각각 플래시를 켜고 끄는 동작을 하도록 코드 작성
                torch.flashOn()
            }
            "off" -> {
                torch.flashOff()
            }
        }
        서비스는 메모리 부족 등의 이유로 강제 종료 될 수 있음
        onStartCommand()메서드는 반환 값에 따라 시스템이 강제로 종료한 후에
        시스템 자원이 회복되어 다시 서비스를 시작할 수 있을 때 어떻게 할지를 결정

        return super.onStartCommand(intent, flags, startId)
    }

    override fun onBind(intent: Intent): IBinder {
        TODO( reason: "Return the communication channel to the service.")
    }
}
```

손전등 앱 만들기

▶ onStartCommand() 반환 값 종류

▶ START_STICKY

- ▶ 인텐트로 null을 넘겨받고 다시 시작
- ▶ 명령을 실행하지는 않지만 무기한으로 대기 중이며 작업을 기다리고 있는 미디어 플레이어와 비슷함

▶ START_NOT_STICKY

- ▶ 다시 시작하지 않음

▶ START_REDELIVER_INTENT

- ▶ 마지막 인텐트로 다시 시작함
- ▶ 능동적으로 수행중인 파일 다운로드와 같은 서비스에 적합

▶ 대부분의 경우에 Super클래스의 onStartCommand() 메서드를 호출하면 내부적으로 START_STICKY를 반환

▶ 예제에서는 서비스가 강제종료 될 가능성이 적기 때문에 어떤 것을 선택해도 관계없음

손전등 앱 만들기

▶ 액티비티에서 서비스를 사용해 손전등 켜기 코드

▶ 일반적인 사용 방법

```
val intent = Intent(this, TorchService::Class.java)
intent.action = "on"
startService(intent)
```

▶ Anko 라이브러리 사용

```
startService(intentFor<TorchService>().setAction("on"))
```

▶ MainActivity.kt 파일의 onCreate()내부에 아래 코드 추가

```
flashSwitch.setOnCheckedChangeListener { _, isChecked ->
    if (isChecked) {
        startService(intentFor<TorchService>().setAction("on"))
    } else {
        startService(intentFor<TorchService>().setAction("off"))
    }
}
```

▶ 실행하기!

▶ 액티비티에서 플래시를 켜는 구조에서 서비스를 사용하여 제어하는 구조로 변경

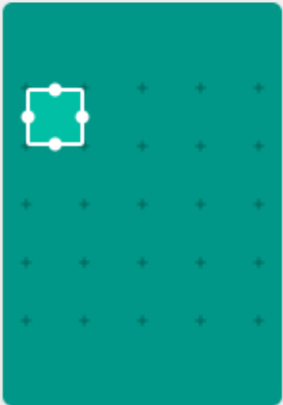
손전등 앱 만들기

▶ 앱 위젯

- ▶ 앱 위젯은 런처에 배치하여 빠르게 앱 기능을 제공하는 컴포넌트
- ▶ 손전등 위젯을 클릭하면 앱 실행없이 플래시가 켜지고 꺼짐

▶ 앱 위젯 추가

- ▶ 메뉴에서 File - New - Widget - App Widget을 클릭
- ▶ 클래스명 : TorchAppWidget



Creates a new App Widget

Class Name:

Placement:

Resizable (API 12+):

Minimum Width (cells):

Minimum Height (cells):

☐ Configuration Screen

Source Language:

손전등 앱 만들기

▶ 앱 위젯 속성

Creates a new App Widget

Class Name:

Placement:

Resizable (API 12+):

Minimum Width (cells):

Minimum Height (cells):

☐ Configuration Screen

Source Language:

▶ Placement

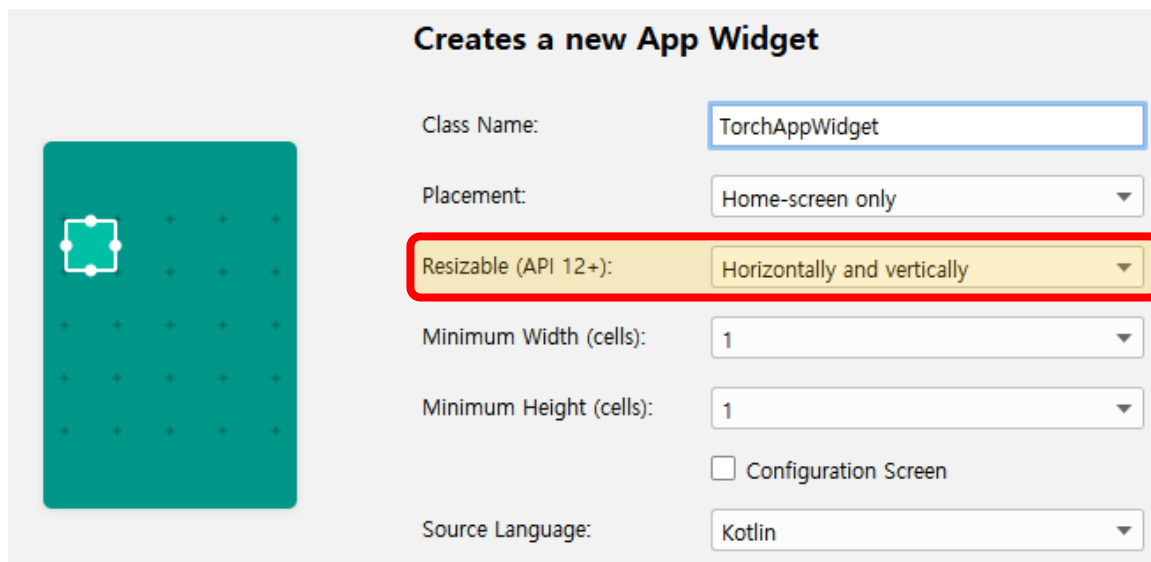
▷ 위젯의 배치위치를 설정

- Home-screen only : 홈화면에만 배치 가능
- Home-screen and Keyguard : 홈 화면과 잠금 화면에 배치 가능
- Keyguard only(API 17+) : 잠금 화면에만 배치 가능



손전등 앱 만들기

▶ 앱 위젯 속성



Creates a new App Widget

Class Name:

Placement:

Resizable (API 12+):

Minimum Width (cells):

Minimum Height (cells):

☐ Configuration Screen

Source Language:

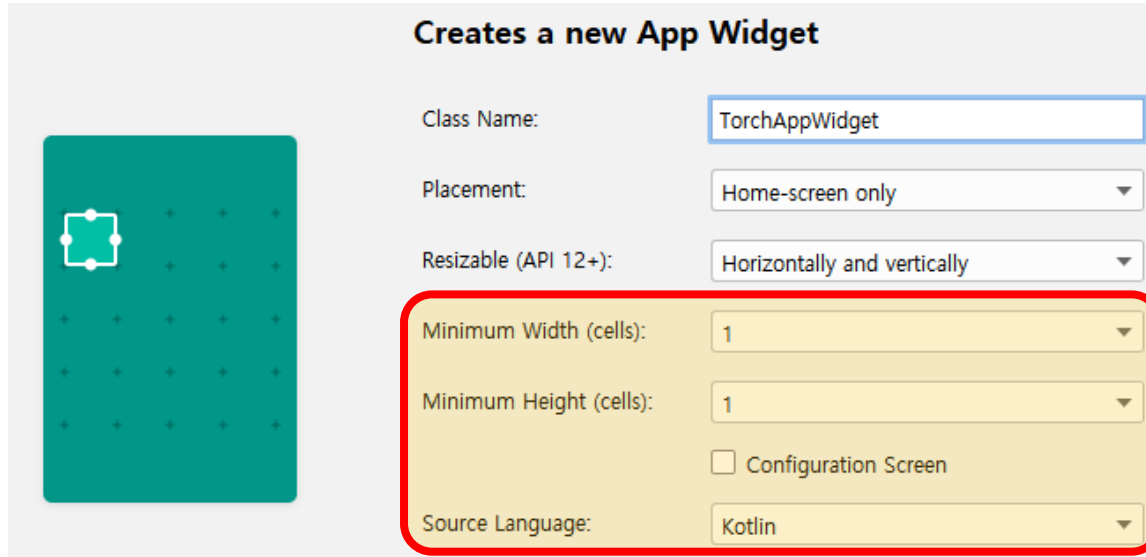
▶ Resizable(API 12+)

▷ 위젯 크기를 변경하는지 설정

- Horizontally and Vertically : 가로와 세로로 크기 변경 가능
- Only horizontally : 가로로만 크기 변경 가능
- Only vertically : 세로로만 크기 변경 가능
- Not resizable : 크기 변경 불가

손전등 앱 만들기

▶ 앱 위젯 속성



Creates a new App Widget

Class Name: TorchAppWidget

Placement: Home-screen only

Resizable (API 12+): Horizontally and vertically

Minimum Width (cells): 1

Minimum Height (cells): 1

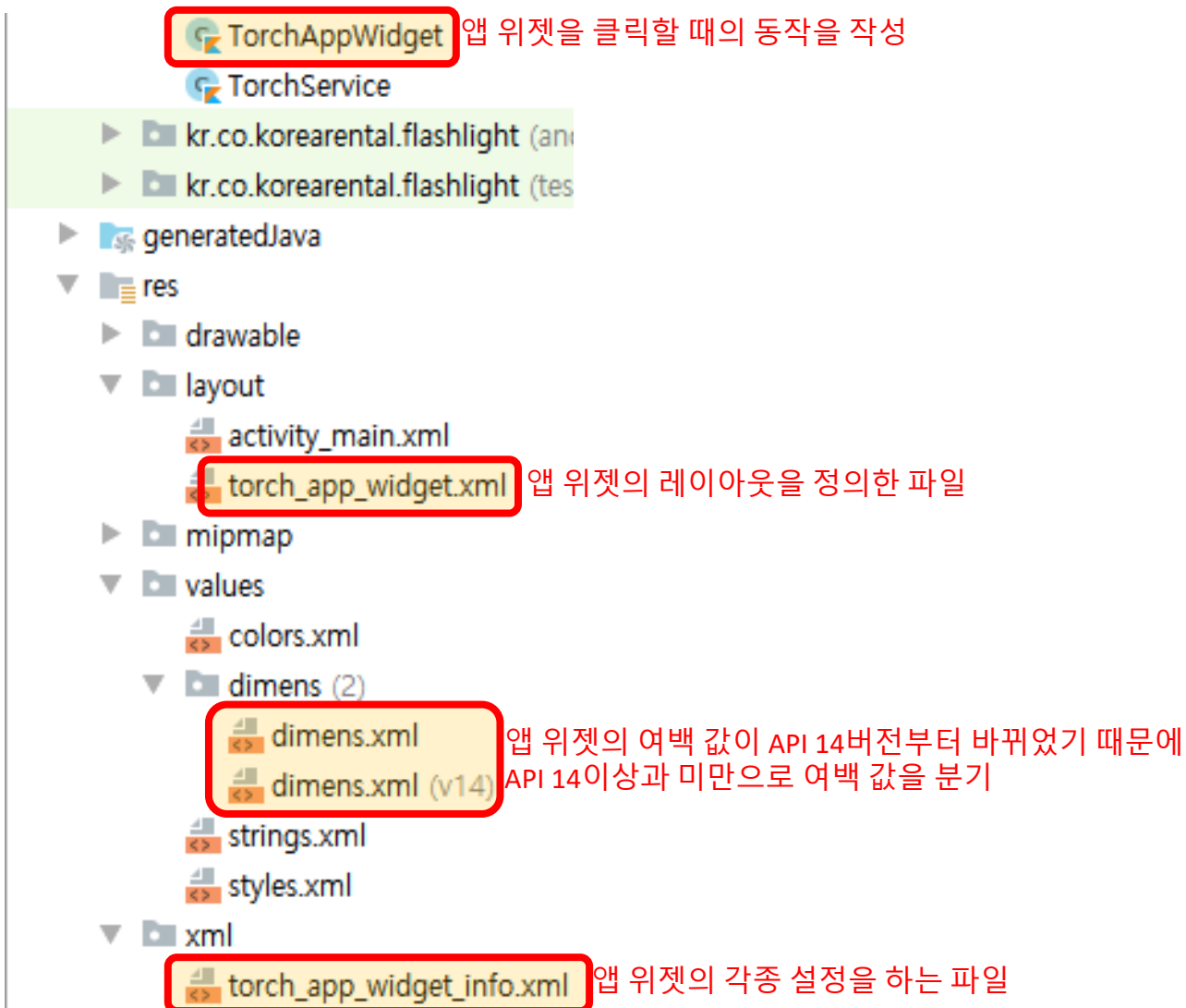
☐ Configuration Screen

Source Language: Kotlin

- ▶ Minimum Width (cells) : 가로 크기를 1 ~ 4중에 선택
- ▶ Minimum Height (cells) : 세로 크기를 1 ~ 4 중에 선택
- ▶ Configuration Screen : 위젯의 환경설정 액티비티를 생성
- ▶ Source language : java or kotlin

손전등 앱 만들기

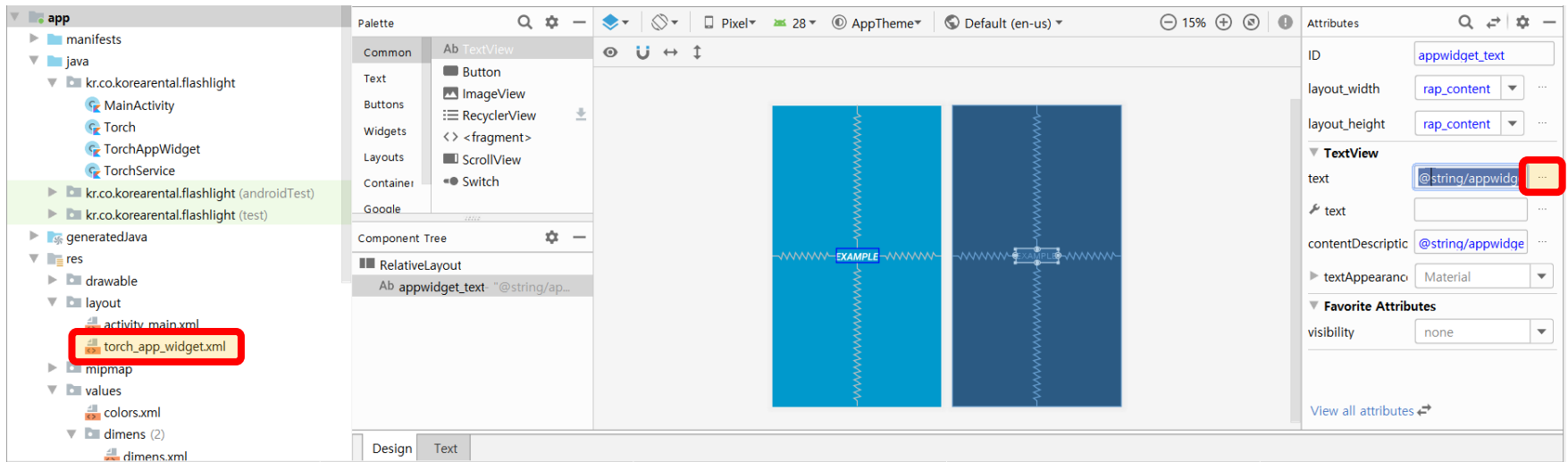
▶ 앱 위젯이 만들어지면서 생성된 파일



손전등 앱 만들기

▶ 앱 위젯 레이아웃 수정

- ▶ 위젯 배치했을 때 표시할 레이아웃인 layout/torch_app_widget.xml 파일을 열기
- ▶ 컴포넌트 트리 창에서 텍스트 뷰를 선택하고 속성 창에서 text 속성을 수정하기 위하여 ... 클릭

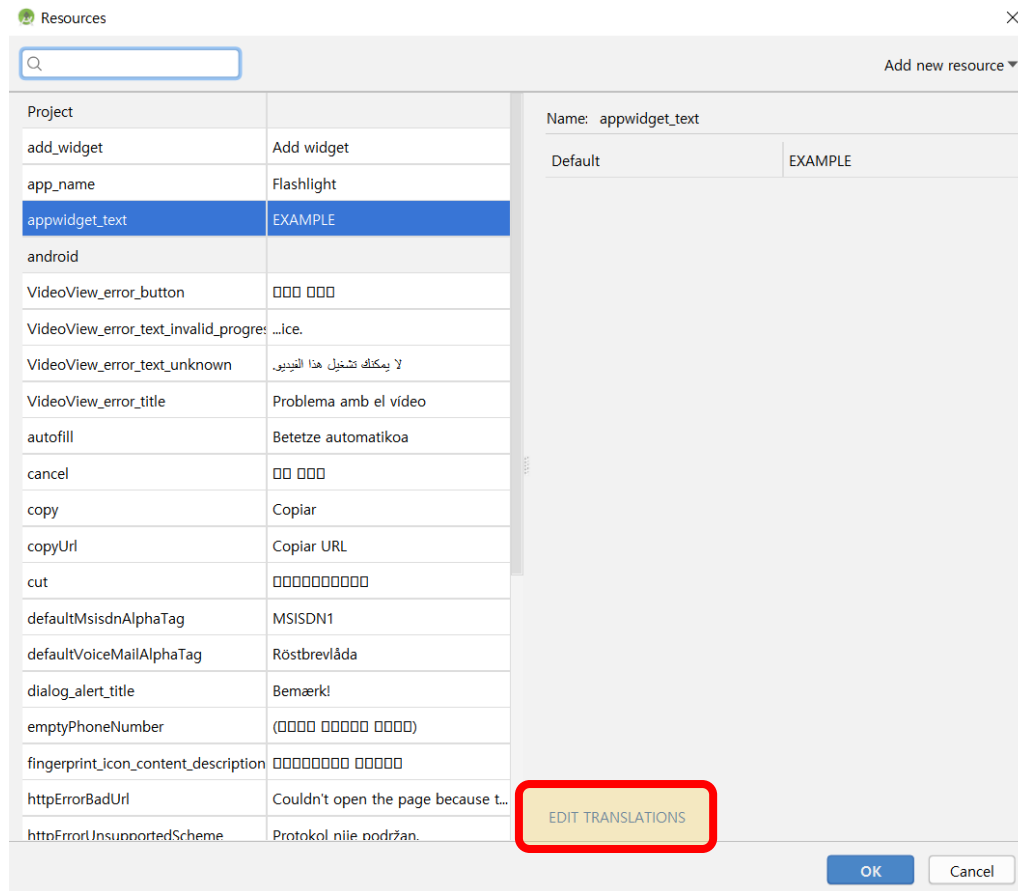


손전등 앱 만들기

▶ 앱 위젯 레이아웃 수정

▶ 텍스트 뷰의 문자는 strings.xml 파일에 appwidget_text라는 이름의 문자열로 지정되어 있음

▶ EDIT TRANSLATIONS를 클릭



손전등 앱 만들기

▶ 앱 위젯 레이아웃 수정

- ▶ 교육 초반에 다루었던 번역 에디터 화면 표시
- ▶ appwidget_text를 손전등으로 수정

+

-

⊕

Show All Keys ▾

Show All Locales ▾

↺

?

| | | | |
|----------------|------------------|--------------------------|------------|
| app_name | app\src\main\res | <input type="checkbox"/> | Flashlight |
| appwidget_text | app\src\main\res | <input type="checkbox"/> | 손전등 |
| add_widget | app\src\main\res | <input type="checkbox"/> | Add widget |

- ▶ value/strings.xml 파일을 열어서 수정 가능

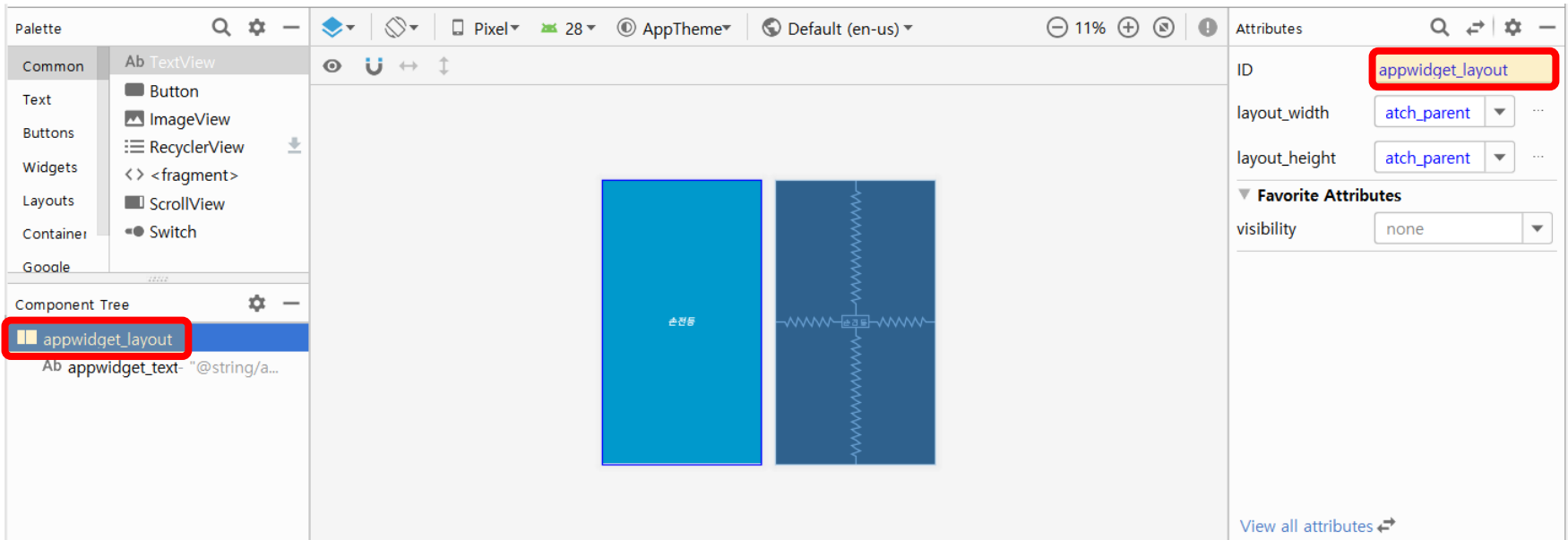
```
<resources>
    <string name="app_name">Flashlight</string>
    <string name="appwidget_text">손전등</string>
    <string name="add_widget">Add widget</string>
</resources>
```

손전등 앱 만들기

▶ 앱 위젯 레이아웃 수정

▶ 위젯을 클릭하면 플래시가 켜지도록 레이아웃 전체에 클릭 이벤트를 연결할 예정

▶ RelativeLayout을 선택하고 ID를 appwidget_layout으로 수정



손전등 앱 만들기

▶ 앱 위젯에서 손전등 켜기

▶ 위젯을 클릭하면 플래시가 켜지도록 코드 작성

▶ TorchAppWidget.kt 파일 수정

▶ 앱 위젯용 파일은 AppWidgetProvider라는 브로드캐스트 리시버를 확장한 클래스를 상속

- 앱 위젯이 업데이트되거나, 활성화 및 비활성화되거나, 삭제될 때 브로드캐스트 이벤트를 받음
- 따라서 이 클래스를 확장하여 내 앱의 위젯이 해당 이벤트들을 받았을 때 어떻게 동작할지를 구현

```
class TorchAppWidget : AppWidgetProvider() {
```

```
    override fun onUpdate(context: Context, appWidgetManager: AppWidgetManager, appWidgetIds: IntArray) {  
        // There may be multiple widgets active, so update all of them  
        for (appWidgetId in appWidgetIds) { 배치되어 있는 모든 위젯을 업데이트  
            updateAppWidget(context, appWidgetManager, appWidgetId)  
        }  
    }  
}
```

손전등 앱 만들기

▶ 앱 위젯에서 손전등 켜기

▶ 위젯을 클릭하면 플래시가 켜지도록 코드 작성 - TorchAppWidget.kt

```
override fun onEnabled(context: Context) { 위젯이 처음 생설될 때 호출  
    // Enter relevant functionality for when the first widget is created  
}
```

```
override fun onDisabled(context: Context) { 여러 개의 위젯에서 마지막 위젯이 제거될 때 호출  
    // Enter relevant functionality for when the last widget is disabled  
}
```

손전등 앱 만들기

▶ 앱 위젯에서 손전등 켜기

▶ 위젯을 클릭하면 플래시가 켜지도록 코드 작성 - TorchAppWidget.kt

▶ `PendingIntent.getActivity()` : 액티비티 실행

▶ `PendingIntent.getService()` : 서비스 실행

▶ `PendingIntent.getBroadcast()` : 브로드캐스트 실행

```
companion object {
```

위젯이 업데이트 될 때 실행되는 메소드

```
internal fun updateAppWidget(context: Context, appWidgetManager: AppWidgetManager, appWidgetId: Int) {
```

```
    val widgetText = context.getString(R.string.appwidget_text)
```

```
    val views = RemoteViews(context.packageName, R.layout.torch_app_widget) RemoteViews는 위젯에 배치하는 전용 뷰
```

```
    views.setTextViewText(R.id.appwidget_text, widgetText) RemoteViews에서 텍스트 값을 설정하는 메소드
```

```
    val intent = Intent(context, TorchService::class.java)
```

```
    val pendingIntent = PendingIntent.getService(context, requestCode: 0, intent, flags: 0) 리퀘스트 코드와 플래그는  
    위젯을 클릭하면 위에서 정의된 인텐트 실행 사용하지 않음
```

```
    views.setOnClickPendingIntent(R.id.appwidget_layout, pendingIntent)
```

레이아웃을 모두 수정했다면 `AppWidgetManager`를 사용하여 위젯을 업데이트

```
    appWidgetManager.updateAppWidget(appWidgetId, views)
```

```
}
```

```
}
```


손전등 앱 만들기

▶ 앱 위젯에서 손전등 켜기

▶ 위젯을 클릭하면 플래시가 켜지도록 코드 작성 - TorchService

▷ 위젯을 클릭하면 TorchService 가 실행

▷ 서비스에서는 인텐트에 On/Off 액션을 지정해서 제어했는데 위젯의 경우 어떤 경우가 On/Off 인지 알 수 가 없으므로 액션을 지정할 수 없었음

▷ 따라서 액션이 지정되지 않아도 플래시가 동작하도록 TorchService.kt 파일을 수정

손전등 앱 만들기

▶ 앱 위젯에서 손전등 켜기

▷ 액션이 지정되지 않아도 플래시가 동작하도록 TorchService.kt 파일을 수정

```
private var isRunning = false 플래시의 상태가 저장되는 변수

override fun onStartCommand(intent: Intent?, flags: Int, startId: Int): Int {
    when (intent?.action) {
        // 앱에서 실행할 경우
        "on" -> {
            torch.flashOn()
            isRunning = true 플래시 상태를 켜짐으로 전환
        }
        "off" -> {
            torch.flashOff()
            isRunning = false 플래시의 상태를 꺼짐으로 전환
        }
        // 서비스에서 실행할 경우
        else -> { isRunning 값에 따라서 플래시를 켜거나 끄는 동작이 결정
            isRunning = !isRunning
            if (isRunning) {
                torch.flashOn()
            } else {
                torch.flashOff()
            }
        }
    }
    return super.onStartCommand(intent, flags, startId)
}
```

손전등 앱 만들기

▶ 앱 위젯에 배치 가능한 뷰

▶ 레이아웃(ConstraintLayout은 아직 지원되지 않음)

▷ FrameLayout, LinearLayout, RelativeLayout, GridLayout

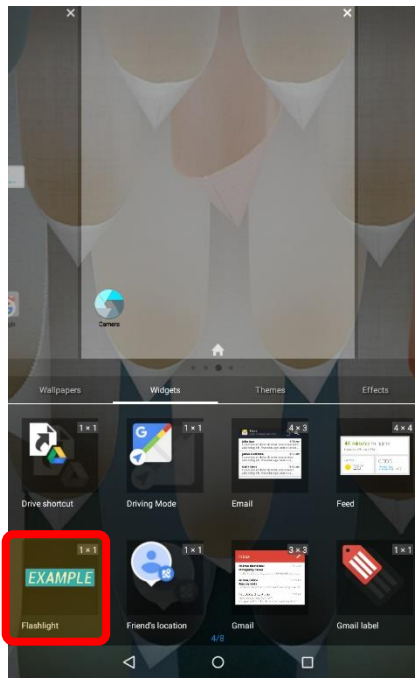
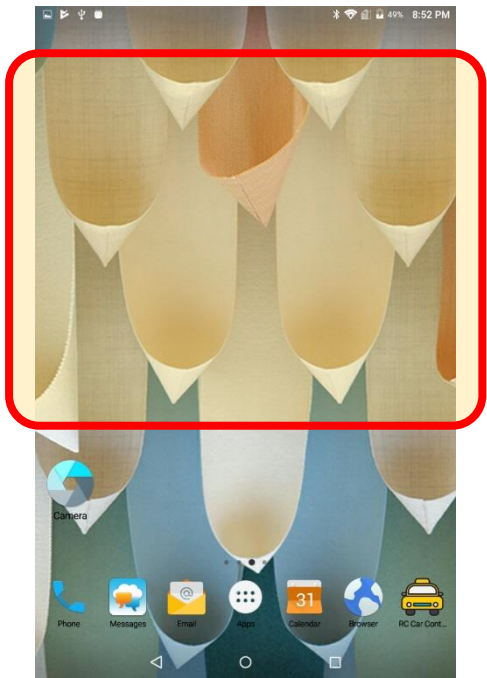
▶ 기타

▷ AnalogClock, Button, Chronometer, ImageButton, ImageView, ProgressBar, TextView, ViewFlipper, ListView, GridView, StackView, AdapterViewFlipper

손전등 앱 만들기

▶ 앱 위젯 배치

- ▶ 위젯 코드가 반영되도록 앱을 실행하고 바로 종료
- ▶ 런처의 빈 공간을 길게 클릭하면 위젯 배치 가능
- ▶ 위젯을 클릭하고 스크롤하여 우리가 구현한 Flashlight 위젯을 다시 길게 클릭하여 런처의 빈공간에 넣음(또는 드래그)



손전등 앱 만들기

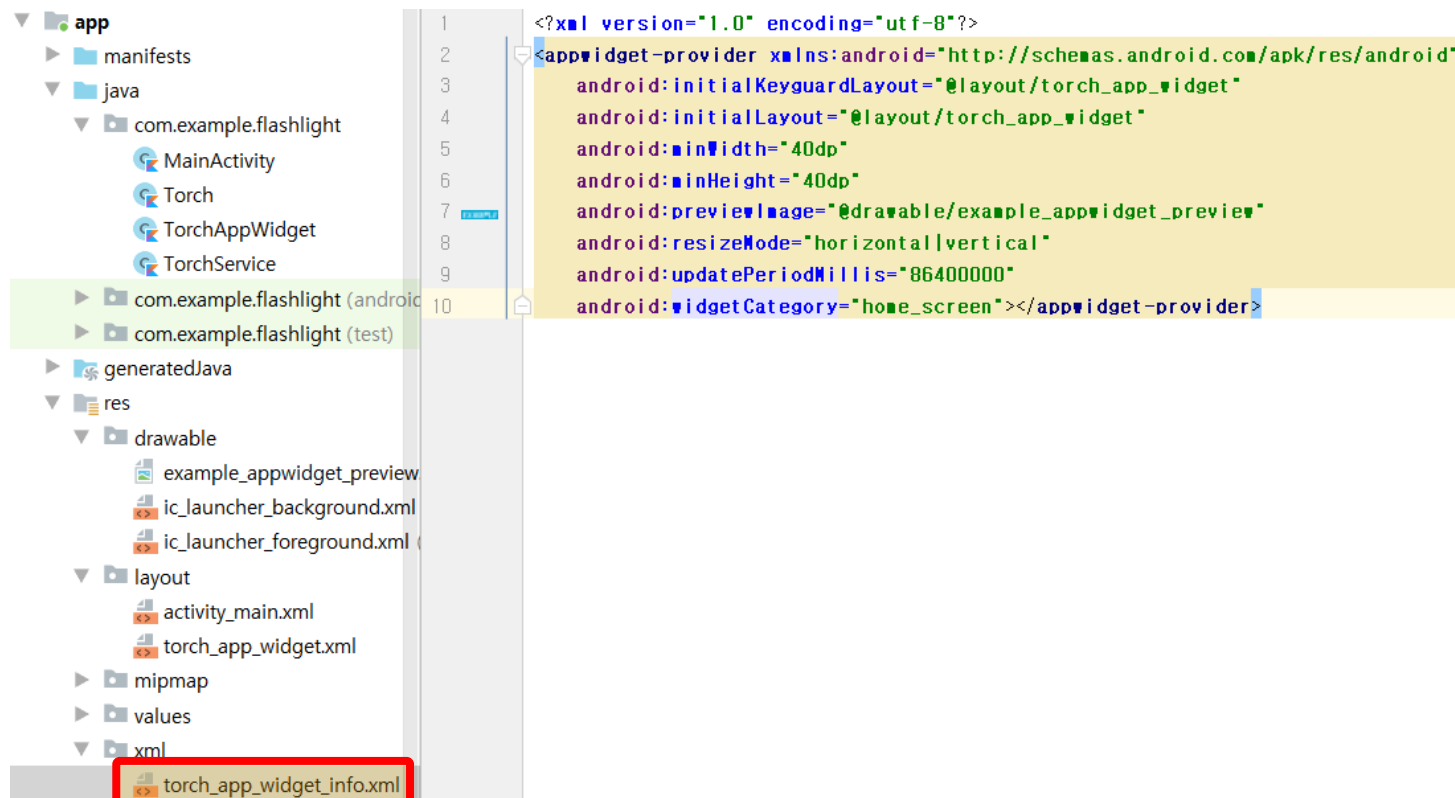
▶ 앱 위젯 배치

▶ 미리보기 이미지는 앱 위젯을 생성 시 drawable 폴더에 생성된 이미지 파일

▶ example_appwidget_preview.png

EXAMPLE

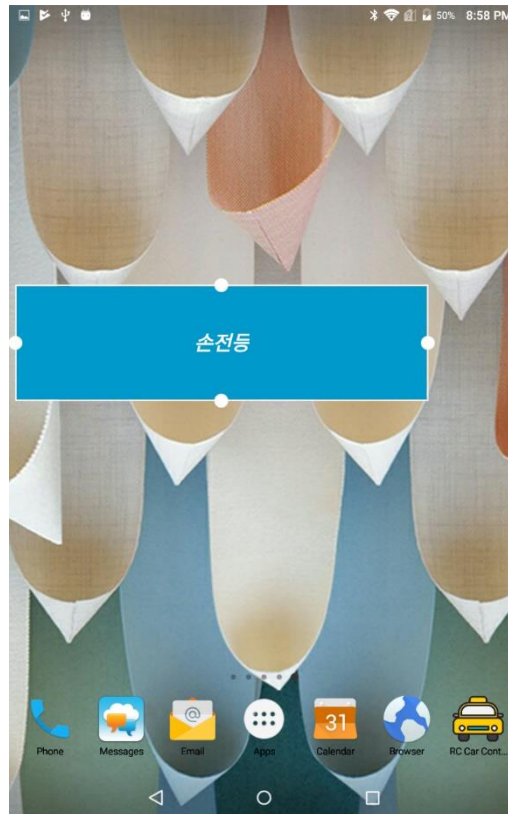
▶ 이미지를 포함한 각종 위젯의 정보는 xml/torch_app_widget_info.xml파일에 작성되어 있음



손전등 앱 만들기

▶ 앱 위젯 배치

- ▶ resizeMode 설정이 horizontal과 vertical로 설정되어 위젯의 크기를 사방으로 조절 가능
- ▶ 위젯의 크기를 조절하려면 위젯을 길게 클릭하여 크기 조절 모드로 변경 후 아이콘을 드래그하여 원하는 크기로 변경 조절이 끝나면 런처의 빈공간을 클릭



손전등 앱 만들기

▶ 앱 위젯 사용하기

▶ 위젯이 잘 작동하는지 테스트

▷ 에뮬레이터와 안드로이드 6.0미만에서는 동작하지 않음

▶ 위젯이 동작하지 않을 때

▷ 앱에서 플래시를 한번 켜다 끄고 위젯이 동작하는지 확인

▷ 위젯을 배치한 상태에서 코드를 수정하여 재실행하면 기존의 위젯이 동작하지 않을 수 있음, 위젯을 삭제하고 다시 배치

Q & A
