

---

# Kotlin을 이용한 Android 프로그래밍

## 전자책 앱 만들기

---

# Contents

---

## I. 3초마다 사진이 바뀌는 전자액자 앱

# 3초마다 사진이 바뀌는 전자액자 앱

▶ 프로젝트 명 : MyGallery

▶ 앱의 기능

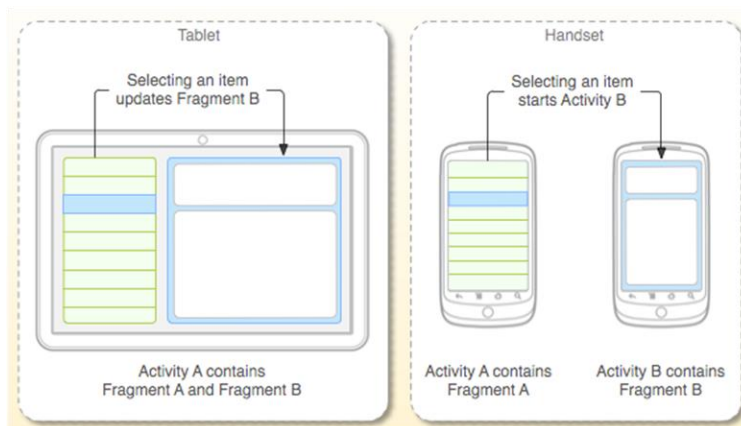
▶ 스마트폰에 저장된 사진이 3초마다 자동으로 슬라이드 되면서 차례대로 화면에 출력됨

▶ 프로젝트 설계

▶ 여러 사진이 좌우 슬라이드되고 3초마다 자동으로 바뀌는 전자액자 앱

▶ 사진은 프래그먼트(fragment)라는 UI 조각으로 구성하고 프래그먼트를 좌우로 슬라이드 할 수 있도록 뷰 페이지저(ViewPager) 사용

▶ Glide 라이브러리로 사진을 로딩하고 timer를 이용하여 자동으로 슬라이드가 되도록 함



# 3초마다 사진이 바뀌는 전자액자 앱

## ▶ 구성요소

- ▶ Content Provider : 사진 정보를 얻기 위해 사용
- ▶ Fragment : UI의 일부를 표현
- ▶ ViewPager : 프래그먼트 여러 개를 좌우 슬라이드로 넘기는 기능
- ▶ FragmentStatePagerAdapter : 페이지가 많을 때 유용한 뷰페이저(ViewPager)용 어댑터
- ▶ Timer : 일정 시간 간격으로 반복되는 동작을 수행

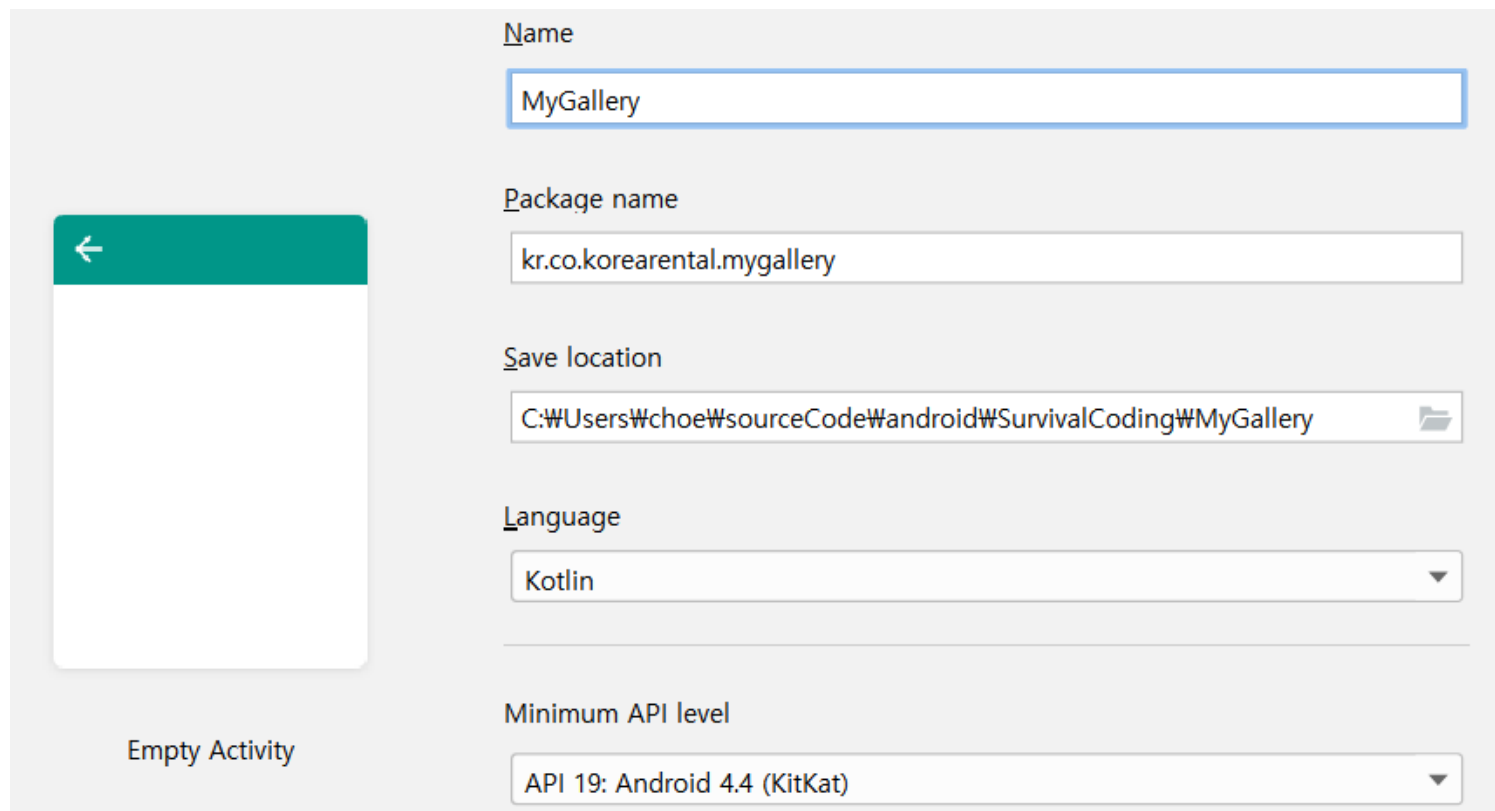
## ▶ 라이브러리 설정

- ▶ Anko 라이브러리 : 인텐트, 다이얼 로그, 로그 등을 효율적으로 구현하게 해주는 라이브러리
- ▶ Glide 라이브러리 : 효율적인 메모리 사용과 자연스러운 사진 로딩에 특화된 라이브러리

# 3초마다 사진이 바뀌는 전자액자 앱

## ▶ 프로젝트 생성

- ▶ 프로젝트 명 : MyGallery
- ▶ minSdkVersion : 19
- ▶ 기본 액티비티 : Empty Activity



Name  
MyGallery

Package name  
kr.co.korearental.mygallery

Save location  
C:\Users\Wchoe\sourceCode\android\SurvivalCoding\MyGallery

Language  
Kotlin

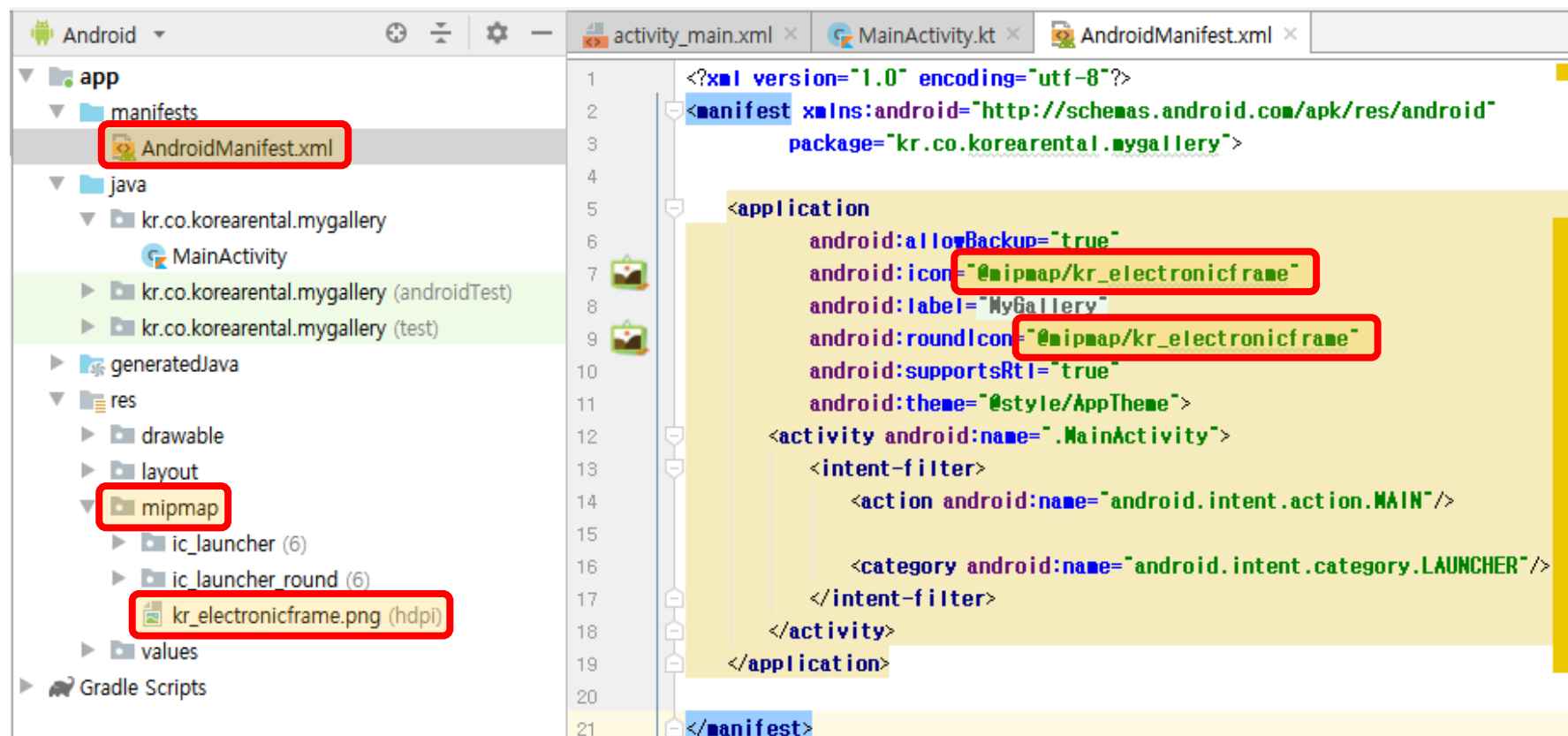
Minimum API level  
API 19: Android 4.4 (KitKat)

Empty Activity

# 3초마다 사진이 바뀌는 전자액자 앱

## ▶ 아이콘 변경

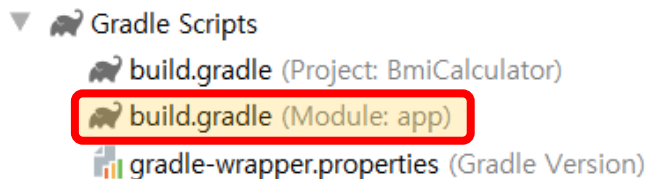
- ▶ 아이콘 파일을 mipmap에 복사 - kt\_electronicframe.png
- ▶ manifest파일에 아이콘 파일 이름 입력



# 나만의 웹 브라우저

## ▶ 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"

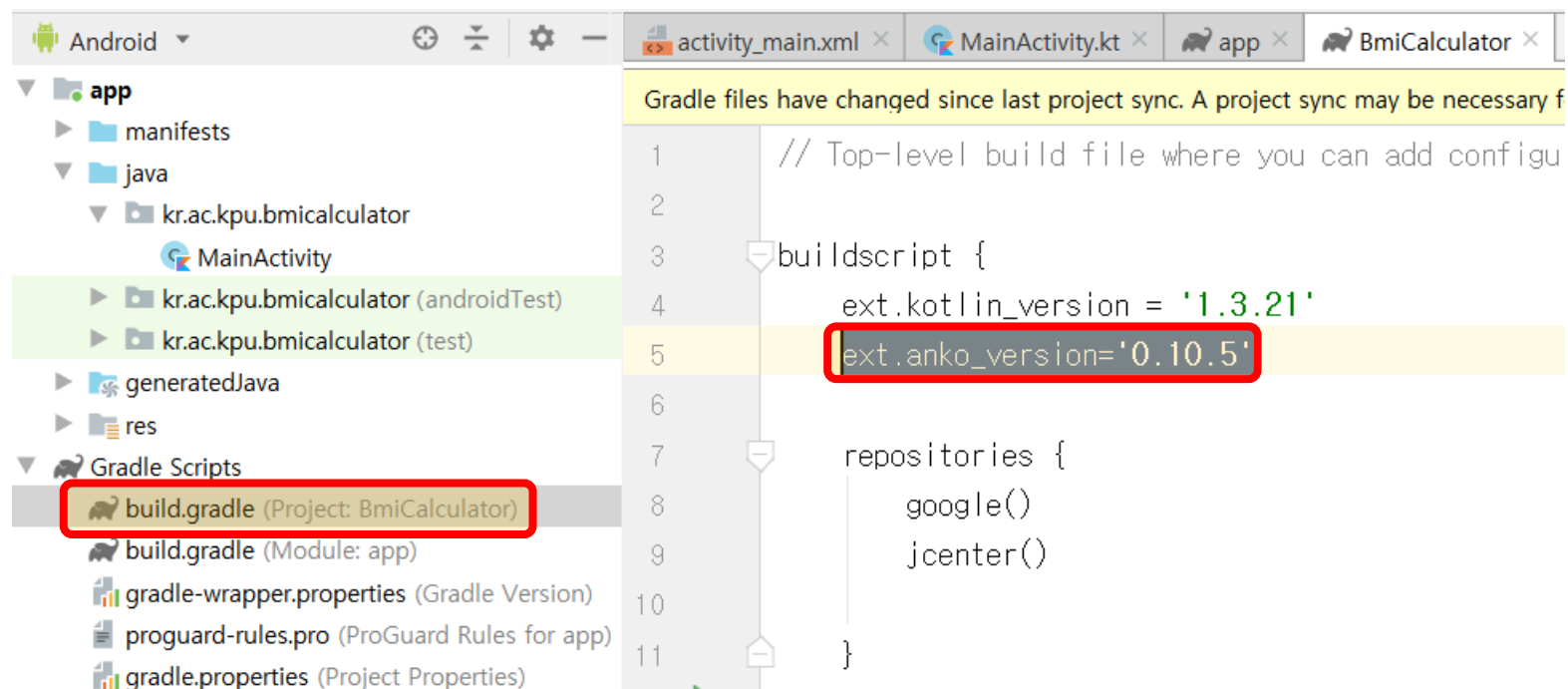
```
25  dependencies {  
26      implementation "org.jetbrains.anko:anko-commons:$anko_version"  
27  }  
28  implementation fileTree(dir: 'libs', include: ['*.jar'])  
29  implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk7:$kotlin_version"  
30  implementation 'com.android.support:appcompat-v7:28.0.0'  
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'
```

# 나만의 웹 브라우저

## ▶ 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 'Gradle Scripts' folder is expanded, and the 'build.gradle (Project: BmiCalculator)' file is selected and highlighted with a red box. The main editor displays the content of this file. A yellow notification bar at the top 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     }
```

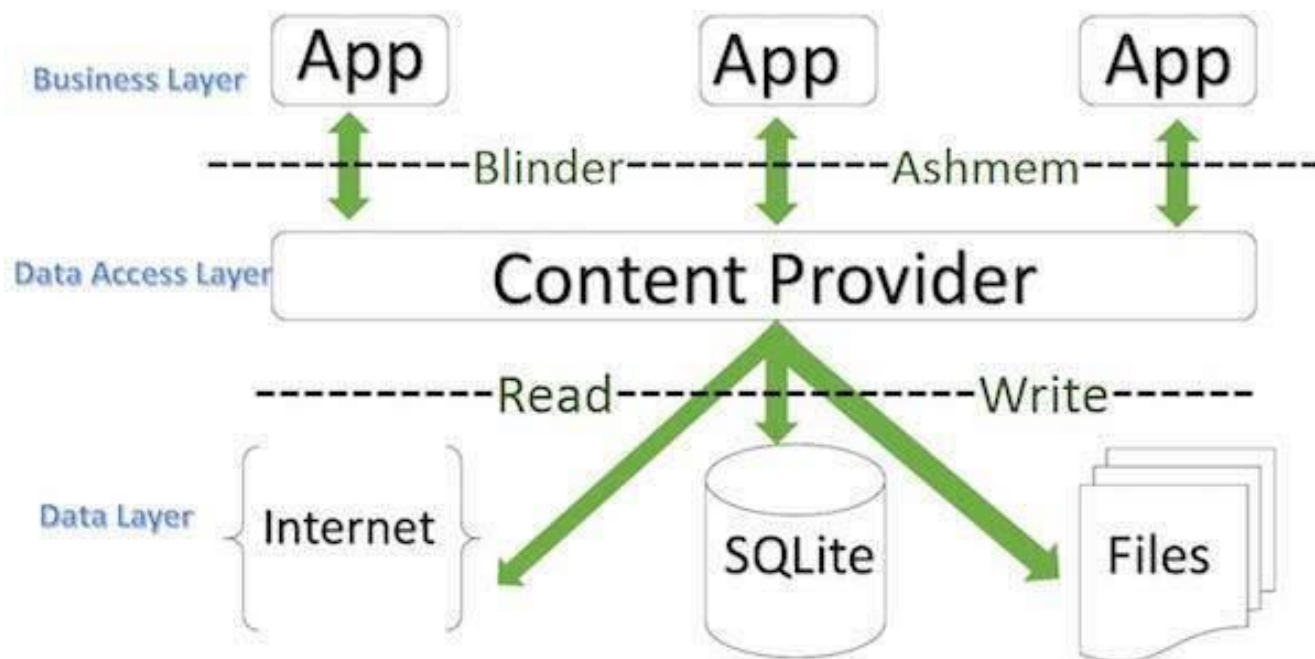
The line `ext.anko_version = '0.10.5'` is highlighted with a yellow background and enclosed in a red rectangular box.



# 3초마다 사진이 바뀌는 전자액자 앱

## ▶ 콘텐츠 프로바이더(Content Provider)개요

- ▶ 앱 내에서 사용할 수 있는 데이터를 공유하기 위한 컴포넌트
  - ▷ 앱의 데이터 접근을 다른 앱에 허용 가능
- ▶ 어플리케이션 계층(Business Layer)과 데이터 계층(Data Layer)의 중간 가교 역할



# 3초마다 사진이 바뀌는 전자액자 앱

## ▶ 콘텐츠 프로바이더(Content Provider)사용

- ▶ 앱은 ContentProvider로 부터 데이터를 얻기 위해 ContentResolver 객체를 사용
  - ▷ getContentResolver()를 통하여 원하는 콘텐츠를 제공 받음
- ▶ 클라이언트 앱에 있는 ContentResolver 객체와 데이터를 제공하는 앱에 있는 ContentProvider 객체는 프로세스간 통신(IPC)을 함
- ▶ ContentResolver 객체의 메소드를 통하여 데이터를 생성, 검색, 업데이트, 삭제 등이 가능

## ▶ 프로젝트에서 활용

- ▶ 사진을 찍으면 내부 저장소에 사진이 저장
- ▶ 동시에 안드로이드 미디어 데이터베이스(MediaStore)에는 사진 정보가 저장됨
  - ▷ 저장된 미디어 데이터(Metadata)는 콘텐츠 프로바이더를 사용하여 다른 앱에 공개 가능
  - ▷ 이미지, 비디오, 오디오 등
  - ▷ 미디어 데이터를 통하여 오늘 찍은 사진들을 자동으로 업로드 하거나 재생 중에 멈췄던 동영상을 해당 시점부터 볼 수 있도록 해줌

# 3초마다 사진이 바뀌는 전자액자 앱

## ▶ 기기의 사진 경로 가져오기

- ▶ 프로바이더를 사용해 사진 정보를 얻으려면 `contentResolver` 객체 사용하여 데이터 얻음
- ▶ 아래는 외부 저장소에 저장된 모든 사진을 최신순으로 정렬하여 `Cursor`라는 객체에 저장하는 코드

## ▶ MediaStore

- ▶ 안드로이드 시스템에서 제공하는 미디어 데이터베이스
- ▶ 파일시스템에 저장되어 있는 미디어 파일을 `MediaStore`에 추가하여 여러 앱에서 이용할 수 있도록 함
- ▶ 시스템이 제공하는 프로바이더를 이용하여 미디어 파일(이미지, 오디오, 비디오)를 쿼리할 수 있음

```
private fun getAllPhotos() {  
    val cursor = contentResolver.query(  
        MediaStore.Images.Media.EXTERNAL_CONTENT_URI, //첫번째 인자는 가져올 데이터를 URI 형태로 지정  
        //사진은 외부 저장소에 있으므로 EXTERNAL_CONTENT_URI  
        projection: null, //두번째 인자는 가져올 데이터를 String 배열로 지정, 데이터의 구조를 모른다면 null, null은 전체 데이터  
        selection: null, //세번째 인자는 데이터를 가져올 조건 지정, 사용하지 않는다면 null  
        selectionArgs: null, //네번째 인자는 세번째 인자와 조합하여 조건을 지정할 경우 사용, 사용하지 않으면 null  
        sortOrder: MediaStore.Images.ImageColumns.DATE_TAKEN + " DESC"  
        //다섯번째 인자는 정렬방법을 지정, 촬영 날짜의 내림 차순 정렬  
    )  
}
```

# 3초마다 사진이 바뀌는 전자액자 앱

## ▶ 기기의 사진 경로 가져오기

### ▶ getAllPhotos()메소드를 onCreate()메소드 내부에서 호출

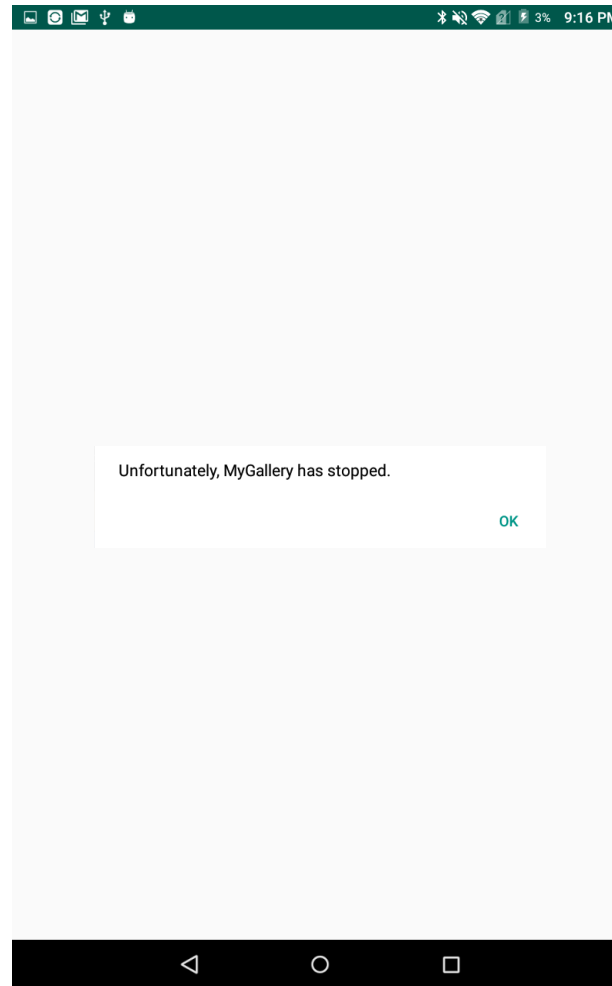
```
override fun onCreate(savedInstanceState: Bundle?) {  
    super.onCreate(savedInstanceState)  
    setContentView(R.layout.activity_main)  
    getAllPhotos()  
}
```

### ▶ 실행해보기!

# 3초마다 사진이 바뀌는 전자액자 앱

▶ 실행화면

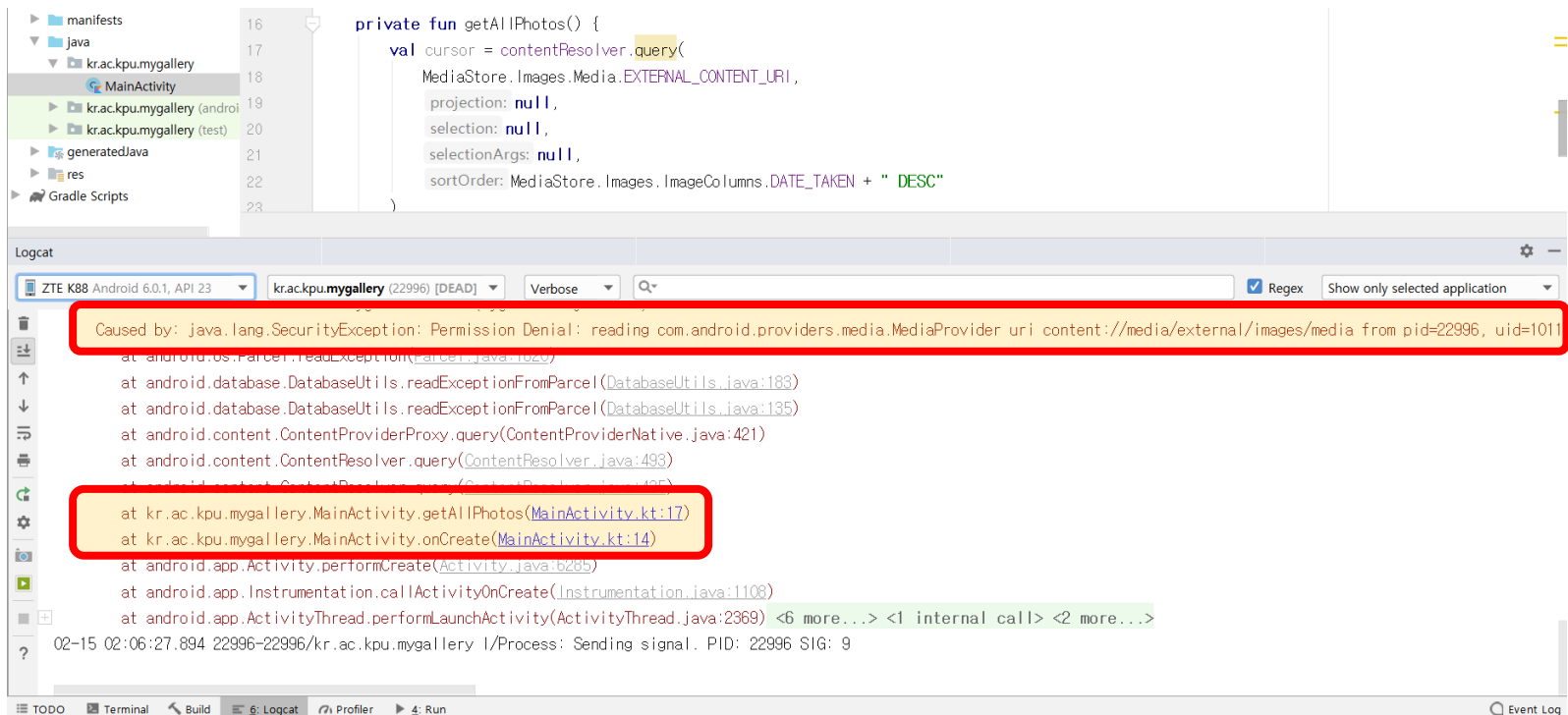
▶ 오류 발생



# 3초마다 사진이 바뀌는 전자액자 앱

## ▶ 기기의 사진 경로 가져오기

- ▶ 앞서 작성한 코드를 실행하면 오류 발생
- ▶ 외부 저장소의 데이터에 접근할 권한이 없어서 발생
- ▶ 에러 발생 위치를 클릭하면 해당 코드를 바로 확인
- ▶ READ\_EXTERNAL\_STORAGE 권한이 없어서 발생하는 에러



# 3초마다 사진이 바뀌는 전자액자 앱

## ▶ 매니페스트에 외부 저장소 읽기 권한 추가

### ▶ 매니페스트 파일을 열고 READ\_EXTERNAL\_STORAGE 외부 저장소 읽기 권한을 앱에 추가

▷ <uses-permission android:name="android.permission.READ\_EXTERNAL\_STORAGE" />

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="kr.ac.kpu.mygallery">
```

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

```
<application
```

```
    android:allowBackup="true"
```

```
    android:icon="@mipmap/ic_launcher"
```

# 3초마다 사진이 바뀌는 전자액자 앱

## ▶ 다시 오류 발생!

▶ 안드로이드 정책이 변경되었음

▶ 안드로이드 6.0부터 모든 앱은 외부에서 리소스 또는 정보를 사용하는 경우 앱에서 사용자에게 권한을 요청해야 함

▶ 메니페스트에 권한을 나열하고 앱을 실행 중에 사용자에게 각 권한을 승인 받으면 해결

▶ getAllPhotos()에서 오류 발생

▶ 위험 권한에 대한 처리가 필요



```
at android.os.Looper.loop(Looper.java:171)
at android.app.ActivityThread.main(ActivityThread.java:5417) <1 internal call>
at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:726)
at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:616)
Caused by: java.lang.SecurityException: Permission Denial: reading com.android.providers.media.MediaProvider uri content://media/external/images/media from pid=23428, uid=10117
at android.os.Parcel.readException(Parcel.java:1620)
at android.database.DatabaseUtils.readExceptionFromParcel(DatabaseUtils.java:183)
at android.database.DatabaseUtils.readExceptionFromParcel(DatabaseUtils.java:135)
at android.content.ContentProviderProxy.query(ContentProviderNative.java:421)
at android.content.ContentResolver.query(ContentResolver.java:493)
at android.content.ContentResolver.query(ContentResolver.java:425)
at kr.ac.kpu.mygallery.MainActivity.getAllPhotos(MainActivity.kt:17)
at kr.ac.kpu.mygallery.MainActivity.onCreate(MainActivity.kt:14)
at android.app.Activity.performCreate(Activity.java:6607)
at android.app.Instrumentation.callActivityOnCreate(Instrumentation.java:1108)
at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:2369) <6 more...> <1 internal call> <2 more...>
02-15 02:12:18.794 23428-23428/kr.ac.kpu.mygallery I/Process: Sending signal. PID: 23428 SIG: 9
```



# 3초마다 사진이 바뀌는 전자액자 앱

## ▶ 안드로이드 권한

- ▶ 안드로이드 시스템에서 권한은 정상(normal)권한과 위험(dangerous)권한으로 분류
  - ▷ 웹 브라우저 예제의 인터넷 사용 권한은 설치 시 허용한 이후 사용하는 정상권한
- ▶ 외부 저장소 읽기 권한은 위험 권한으로 분류
  - ▷ 위험 권한은 실행 할 때마다 사용자에게 권한을 요청해야 함
- ▶ 자주 사용되는 위험 권한

권한 그룹	권한
STORAGE	READ_EXTERNAL_STORAGE WRITE_EXTERNAL_STORAGE
LOCATION	ACCESS_FINE_LOCATION ACCESS_COARSE_LOCATION
SMS	SEND_SMS RECEIVE_SMS
CAMERA	CAMERA

# 3초마다 사진이 바뀌는 전자액자 앱

## ▶ 권한 확인

### ▶ 위험 권한이 필요한 작업은 수행할 때마다 권한을 확인

▷ 권한 설정은 사용자가 언제든지 취소할 수 있으므로 위험 권한이 필요한 작업을 수행할 때마다 확인이 필요

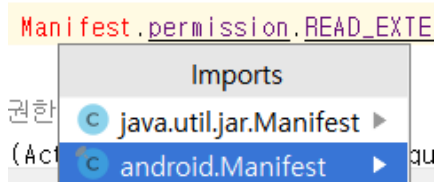
▷ 어제 저장소 읽기 권한이 있더라도 오늘도 있다고 단정할 수 없음

### ▶ 권한 여부를 확인하려면 ContextCompat.checkSelfPermission() 메서드 사용

```
if (ContextCompat.checkSelfPermission(this,
    Manifest.permission.READ_EXTERNAL_STORAGE) != PackageManager.PERMISSION_GRANTED) {
    //권한이 허용되지 않음
}
```

### ▶ 앱에 권한이 있으면 PERMISSION\_GRANTED가 반환, 없으면 PERMISSION\_DENIED 반환

### ▶ Manifest 클래스는 여러 패키지에 존재, 코드 작성 중 어느 것을 import 할 지 물어보면 android 선택



# 3초마다 사진이 바뀌는 전자액자 앱

## ▶ 권한 요청

### ▶ MainActivity 파일을 열고 onCreate() 메서드에 권한을 요청하는 코드 작성

```
// 권한이 부여되었는지 확인
if (ContextCompat.checkSelfPermission(this,
    Manifest.permission.READ_EXTERNAL_STORAGE) != PackageManager.PERMISSION_GRANTED) {
    // 권한이 허용되지 않음
    if (ActivityCompat.shouldShowRequestPermissionRationale(this, Manifest.permission.READ_EXTERNAL_STORAGE)) {
        // 이전에 이미 권한이 거부되었을 때 설명
        alert("사진 정보를 얻기 위해서는 외부 저장소 권한이 필수로 필요합니다", "권한이 필요한 이유") {
            yesButton {
                // 권한 요청
                ActivityCompat.requestPermissions(this@MainActivity,
                    arrayOf(Manifest.permission.READ_EXTERNAL_STORAGE),
                    REQUEST_READ_EXTERNAL_STORAGE)
            }
            noButton { }
        }.show()
    } else {
        // 권한 요청
        ActivityCompat.requestPermissions(this,
            arrayOf(Manifest.permission.READ_EXTERNAL_STORAGE),
            REQUEST_READ_EXTERNAL_STORAGE)
    }
} else {
    // 권한이 이미 허용됨
    getAllPhotos()
}
```

### ▶ 멤버 추가

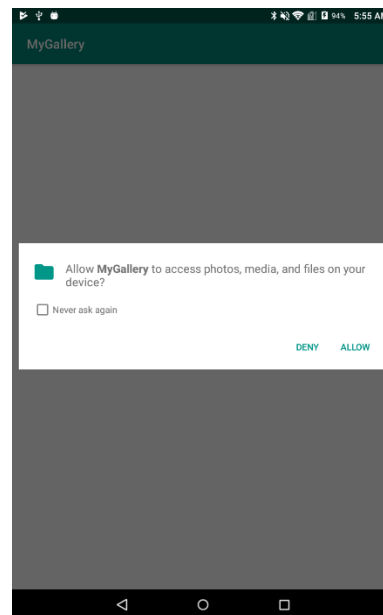
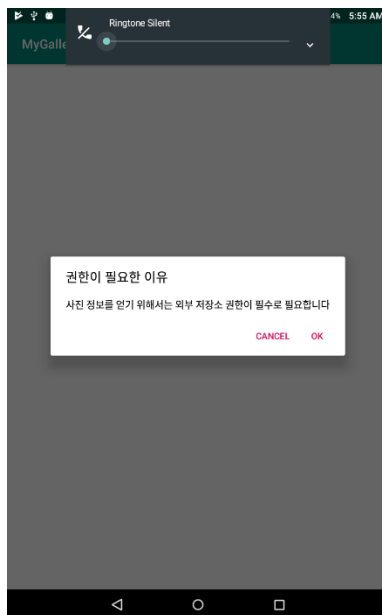
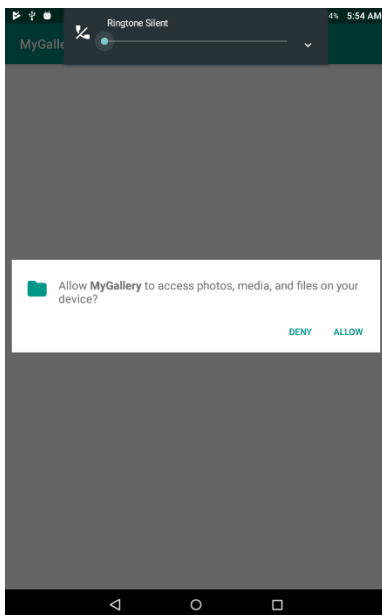
```
private val REQUEST_READ_EXTERNAL_STORAGE = 1000 //권한 요청에 대한 결과를 분기
```

# 3초마다 사진이 바뀌는 전자액자 앱

## ▶ 권한 요청

### ▶ 권한 요청 관련 코드 분석

```
// 권한이 부여되었는지 확인
if (ContextCompat.checkSelfPermission(this,
    Manifest.permission.READ_EXTERNAL_STORAGE) != PackageManager.PERMISSION_GRANTED) {
    // 권한이 허용되지 않음
    | 😊
} else {
    // 권한이 이미 허용됨
    getAllPhotos()
}
```



# 3초마다 사진이 바뀌는 전자액자 앱

## ▶ 권한 요청

### ▶ 권한 요청 관련 코드 분석

```
//사용자가 권한 요청을 거부했는지 확인 하는 메소드, true를 반환하면 거부의사 확인
if (ActivityCompat.shouldShowRequestPermissionRationale(this, Manifest.permission.READ_EXTERNAL_STORAGE)) {
    // 이전에 이미 권한이 거부되었을 때 설명
    alert("사진 정보를 얻기 위해서는 외부 저장소 권한이 필수로 필요합니다", "권한이 필요한 이유") {
        yesButton {
            // 권한 요청 //아래 메서드로 외부 저장소 권한 요청 // 권한이 필요한 이유를 별도의 메시지로 표시
            ActivityCompat.requestPermissions(this@MainActivity,
                arrayOf(Manifest.permission.READ_EXTERNAL_STORAGE),
                REQUEST_READ_EXTERNAL_STORAGE)
        }
        noButton { }
    }.show()
} else {
    // 권한 요청
    ActivityCompat.requestPermissions(this,
        arrayOf(Manifest.permission.READ_EXTERNAL_STORAGE),
        REQUEST_READ_EXTERNAL_STORAGE)
}
```

▶ 권한을 허용하는 것이 중요하지만 사용자에게 매번 알려주는 것은 좋지 않음

- 거부 했을 경우에만 권한의 필요성을 메시지로 표시

# 3초마다 사진이 바뀌는 전자액자 앱

## ▶ 사용 권한 요청 응답 처리

- ▶ 사용자가 권한을 요청하면 시스템은 onRequestPermissionsResult()메서드를 호출하고 사용자의 응답을 전달
  - ▷ 권한이 부여되었는지 확인하려면 이 메서드를 오버라이드 해야함
- ▶ 응답 결과를 확인하여 사진 정보를 가져오거나 권한이 거부되었다는 토스트 메시지를 표시하는 코드를 작성

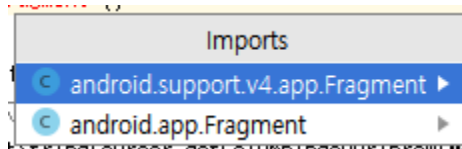
```
override fun onRequestPermissionsResult(requestCode: Int, permissions: Array<out String>, grantResults: IntArray) {  
    super.onRequestPermissionsResult(requestCode, permissions, grantResults)  
  
    when (requestCode) {  
        REQUEST_READ_EXTERNAL_STORAGE -> {  
            if ((grantResults.isNotEmpty()  
                && grantResults[0] == PackageManager.PERMISSION_GRANTED)) {  
                // 권한 허용됨  
                getAllPhotos()  
            } else {  
                // 권한 거부  
                toast("권한 거부 됨")  
            }  
            return  
        }  
    }  
}
```

grantResults배열에 요청한 권한들의 결과가 전달,  
예제는 권한을 하나만 요청했기 때문에 0번 인덱스 값 만 확인  
권한이 승인되면 PERMISSION\_GRANTED를 반환하고  
거부되면 PERMISSION\_DENIED를 반환

# 3초마다 사진이 바뀌는 전자액자 앱

## ▶ 앱 실행

- ▶ 먼저 사진 정보가 잘 읽어지는지 로그로 확인하는 코드를 getAllPhotos()메서드에 추가
- ▶ 사진 정보를 담고있는 Cursor 객체는 내부적으로 데이터 간을 이동하는 포인터를 가지고 있어서 moveToNext()메서드로 다음 정보를 이동하고 그 결과를 true로 반환, while문을 사용하면 모든 데이터를 순환할 수 있음



- ▶ 만약 사진이 없다면 Cursor 객체는 null

```
if (cursor != null) {  
    while (cursor.moveToNext()) {  
        // 사진 경로 Uri 가지고 오기  
        val uri = cursor.getString(cursor.getColumnIndexOrThrow(MediaStore.Images.Media.DATA))  
        Log.d("MainActivity", uri)  
    }  
    cursor.close()  
}
```

사진의 경로가 저장된 데이터베이스의 컬럼명은 DATA 상수에 정의  
getColumnIndexOrThrow() 메서드를 사용하면 해당 컬럼이 몇 번째 인덱스인지 확인  
getString()메서드에 인덱스를 전달하면 해당되는 데이터를 String 타입으로 반환 이것이  
uri 즉 사진이 저장된 위치의 경로가 됨

val uri = cursor.getString(cursor.getColumnIndexOrThrow(MediaStore.Images.Media.DATA))

cursor.close() Cursor 객체를 더이상 사용하지 않을 때는 close() 메서드로 닫아야 함, 만약 닫지 않으면 누수 발생

## ▶ 메모리 누수

- ▶ 메모리가 해제되지 않는 상황이 지속
- ▶ 동작이 느려지고 앱이 비정상 종료

# 3초마다 사진이 바뀌는 전자액자 앱

## ▶ Cursor

- ▶ 데이터베이스에서 가져온 정보를 쉽게 처리하기 위하여 제공된 인터페이스
- ▶ DB에서 값을 가져와서 마치 실제 Table의 한 행(Row), 한 행(Row) 을 참조하는 것 처럼 사용

RecNo	_id	name	contact	email
Click here to define a filter				
1	1	김태희	01000001111	angel@google.com
2	2	방예고	01333331111	asdfm@emdo.com
3	3	낸시랭	01234001111	yaya@hhh.com
4	4	제시카	01600001111	tree777@atat.com
5	5	성유리	01700001111	tiger@tttt.com
6	6	김태우	01800001111	gril@zzz.com

- ▶ Cursor가 행(Row)를 참조하기 때문에 Cursor의 위치를 바꿔주는 메서드들이 존재

메서드	동작
<code>Cursor.moveToFirst();</code>	Cursor를 제일 첫번째 행(Row)으로 이동 시킨다.
<code>Cursor.moveToNext();</code>	Cursor를 다음 행(Row)으로 이동 시킨다.
<code>Cursor.moveToPrevious();</code>	Cursor를 이전 행(Row)으로 이동 시킨다.
<code>Cursor.moveToPosition(position);</code>	Cursor를 해당 Position 행(Row)으로 이동 시킨다.
<code>Cursor.moveToLast();</code>	Cursor를 마지막 행(Row)으로 이동 시킨다.



# 3초마다 사진이 바뀌는 전자액자 앱

## ▶ Cursor

### ▶ 행 이동

RecNo	_jd	name	contact	email
Click here to define a filter				
1	1	김태희	01000001111	angel@google.com
2	2	송혜교	01333331111	asdfrrr@emdo.com
3	3	낸시랭	01234001111	yaya@hhh.com
4	4	제시카	01600001111	tree777@atat.com
5	5	성유리	01700001111	tiger@tttt.com
6	6	김태우	01800001111	gril@zzz.com

← `Cursor.moveToFirst();`

← `Cursor.moveToNext();`

← `Cursor.moveToLast();`

# 3초마다 사진이 바뀌는 전자액자 앱

## ▶ Cursor

### ▶ 참조하고 있는 테이블의 행 Data를 가져오는 메서드

메서드	동작
<code>Cursor.getInt(), Cursor.getString(), Cursor.getLong(), Cursor.getBoolean().....</code>	DB 테이블의 실제 Data를 가지고 옵니다.
<code>Cursor.getColumnIndex(String columnName);</code>	DB 테이블의 해당 필드(컬럼) 이름을 얻어 옵니다.
<code>Cursor.getPosition();</code>	Cursor가 가리키고 있는 DB 테이블 행(Row) Position 을 얻어 옵니다.
<code>Cursor.getColumnName(int columnIndex);</code>	필드(컬럼) index의 해당하는 필드(컬럼) 이름을 얻어 옵니다.
<code>Cursor.getCount();</code>	커서가 참조 할 수 있는 해당 테이블의 행(Row)의 갯수를 얻어 옵니다.
<code>Cursor.getColumnNames();</code>	DB 테이블의 필드(컬럼) 명을 순서대로 배열로 얻어 옵니다.
<code>Cursor.getColumnCount();</code>	DB 테이블의 필드(컬럼) 갯수를 얻어 옵니다.

# 3초마다 사진이 바뀌는 전자액자 앱

## ▶ Cursor

### ▶ 참조하고 있는 테이블의 행 Data를 가져오는 방법

RecNo	_id	0	name	1	contact	2	email	3
Click here to define a filter								
▶	1	1	김태희	01000001111	angel@google.com			
	2	2	송혜교	01333331111	asdffff@amdo.com			
	3	3	낸시랭	01234001111	yaya@nhh.com			
	4	4	제시카	01600001111	tree777@ata.com			
	5	5	성유리	01700001111	tiger@tttt.com			
	6	6	김태우	01800001111	gril@zzz.com			

← **Cursor.moveToFirst();**  
현재 커서가 위치한 행(Row)  
의 값들이 대상이 된다.

**Cursor.getInt(0) = 1**      **Cursor.getString(3) = angel@google.com**

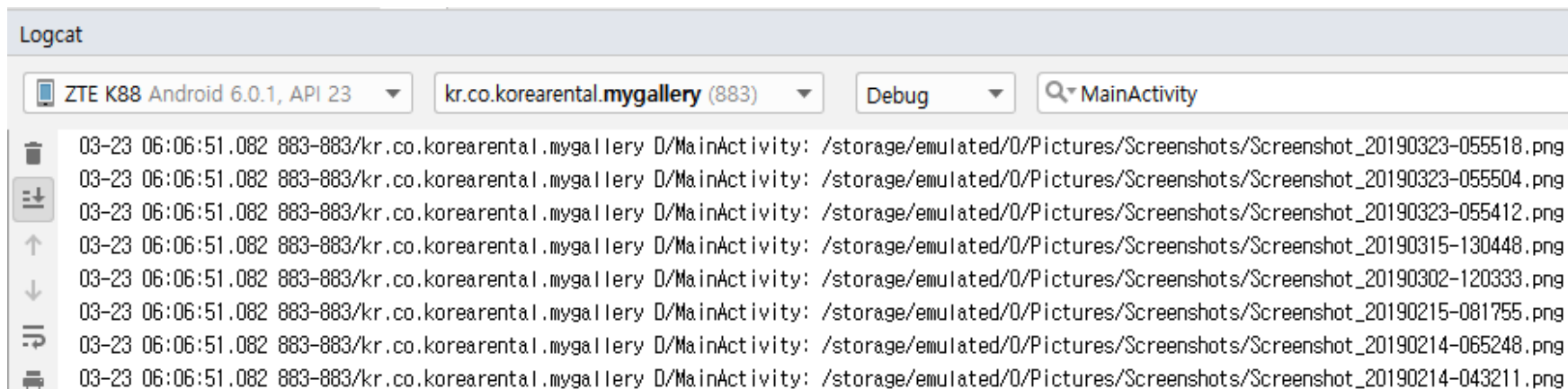
DB에 정의된 필드의 TYPE에 의해 int로 설정한 필드는 **getInt()**, String으로 설정한 필드는 **getString()**으로 값을 얻어와야 합니다.

**Cursor.getInt(columnIndex)** : 실제 Data를 얻어 올 수 있습니다.

# 3초마다 사진이 바뀌는 전자액자 앱

▶ Logcat 에서 MainActivity 태그를 필터링하여 사진의 URI가 표시되면 성공

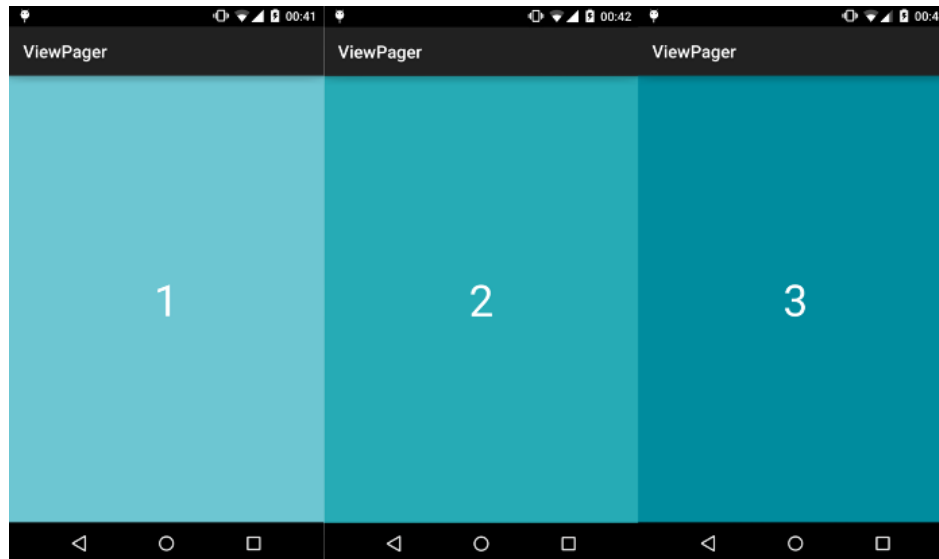
▶ 아무 사진도 없는 경우 표시 안됨



# 3초마다 사진이 바뀌는 전자액자 앱

## ▶ 전자 액자 구현

- ▶ 전자액자 앱은 여러 화면이 좌우로 슬라이드 되는 구조
- ▶ 화면은 프래그먼트로 작성하고 ViewPager를 사용하여 좌우 슬라이드 구현
  - ▷ 이러한 앱화면에는 ViewFlipper나 ViewPager를 사용
  - ▷ ViewFlipper는 하나의 레이아웃에 뷰가 많으면 성능이 저하되므로 ViewPager를 주로 사용
- ▶ 뷰페이저 내부에서 Glide 라이브러리 사용
  - ▷ 사진 로딩 시 메모리를 관리하고 성능을 향상시킴



# 3초마다 사진이 바뀌는 전자액자 앱

## ▶ 프래그먼트의 개요

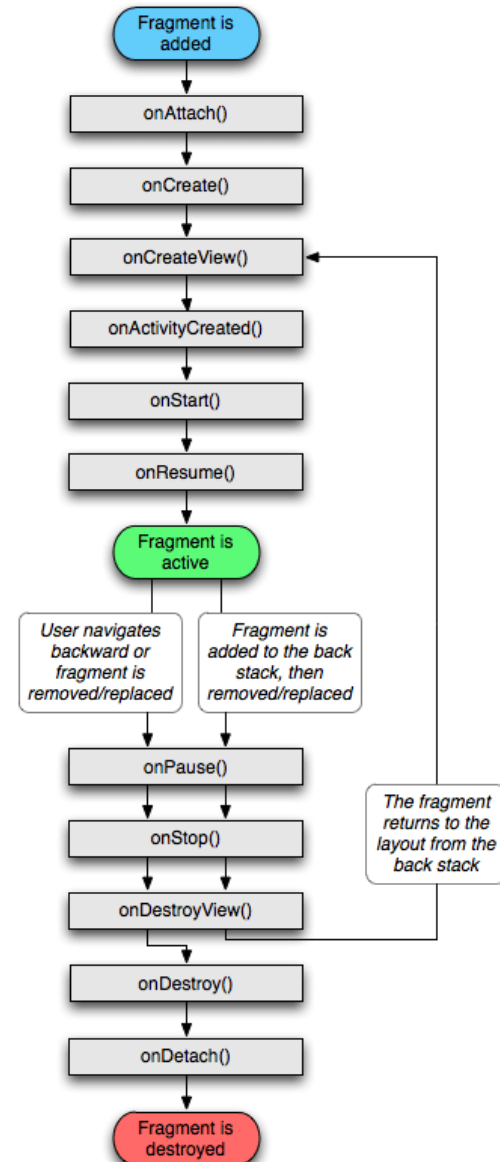
- ▶ 프래그먼트 여러 개를 조합하여 액티비티 하나를 구성할 수 있고 한 번 작성한 프래그먼트는 재사용할 수 있음
- ▶ 하나의 화면에서 일부를 조금 다른 화면으로 구성 하고 싶을 때는 프래그먼트를 사용



# 3초마다 사진이 바뀌는 전자액자 앱

## ▶ 프래그먼트의 개요

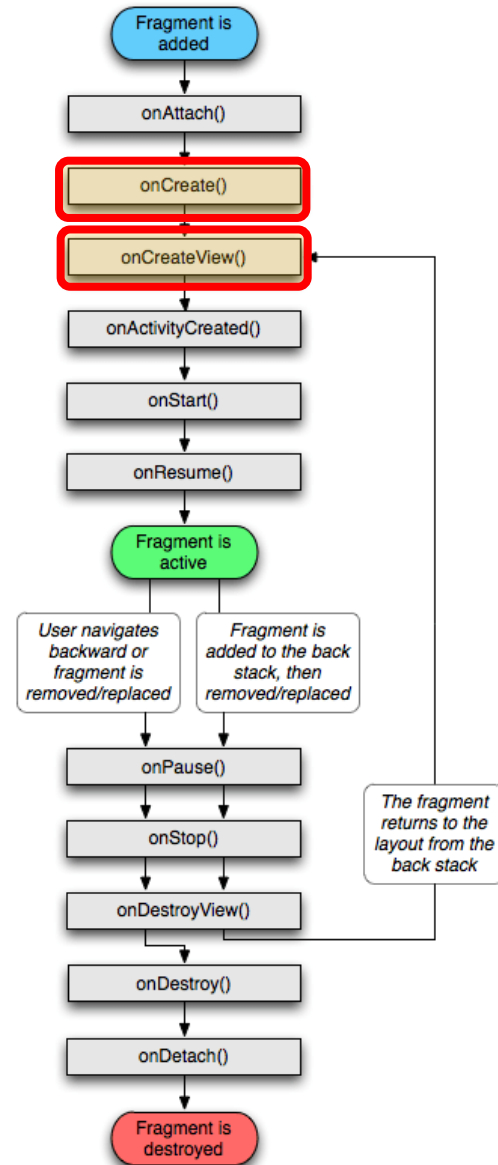
- ▶ 프래그먼트 (fragment) 는 하나의 액티비티가 여러 개의 화면을 가지도록 만들기위해 고안된 개념
- ▶ 재사용성이 높음
- ▶ 프래그먼트는 액티비티처럼 독자적인 생명주기를 가지고 있음
- ▶ 액티비티에 비해서 훨씬 많은 생명주기 콜백 메서드가 있는데 모두 다 알 필요는 없음



# 3초마다 사진이 바뀌는 전자액자 앱

## ▶ 프래그먼트의 생명주기

- ▶ onCreate() : 프래그먼트를 생성할 때 넘어오는 인자를 주로 여기서 처리
- ▶ onCreateView() : 주로 프래그먼트에 표시할 뷰를 레이아웃 파일로부터 읽어오는 부분
  - ▷ 액티비티의 onCreate()와 동일한 역할
  - ▷ onCreateView() 메서드에서 완성된 레이아웃 뷰는 생명주기에는 포함되지 않는 onCreateView() 메서드로 전달되며 이쪽에서 뷰가 완성된 이후에 이벤트 처리 등을 수행
- ▶ 예제에서는 주로 onCreate(), onCreateView(), onCreateView() 세가지 콜백 메서드를 사용

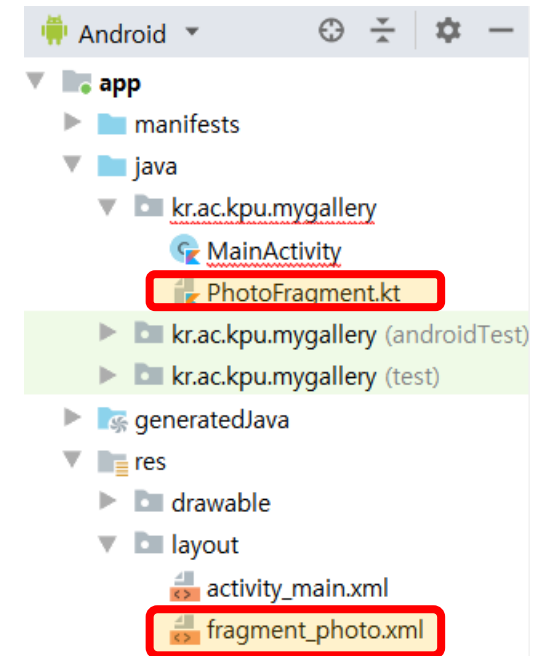
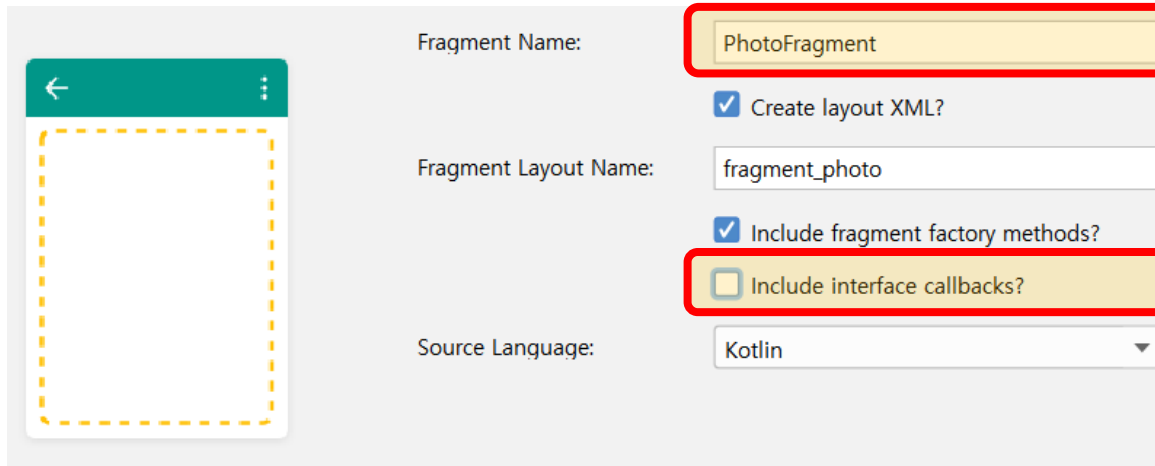




# 3초마다 사진이 바뀌는 전자액자 앱

## ▶ 프래그먼트 생성

▶ File → New → Fragment → Fragment(Blank)



### ▶ Create layout XML? :

- XML 레이아웃 파일을 생성. 참고로 이번 예제는 XML 레이아웃 파일을 사용하므로 체크

### ▶ Include fragment factory method :

- 프래그먼트를 생성할 때 인자를 넘겨줄 경우 체크

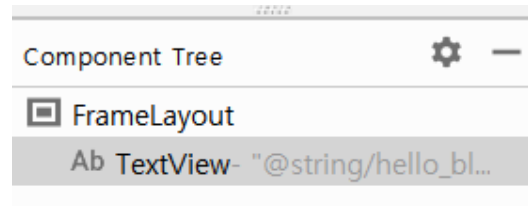
### ▶ Include interface callbacks? :

- 프래그먼트에서 발생한 이벤트를 액티비티에 전달할 경우 체크 / 예제에서는 사용하지 않음

# 3초마다 사진이 바뀌는 전자액자 앱

## ▶ 프래그먼트의 레이아웃 수정

- ▶ 프래그먼트도 액티비티처럼 하나의 클래스와 XML 레이아웃 파일로 구성
- ▶ PhotoFragment는 프로젝트에서 하나의 사진을 꼭 차게 보여주는 프래그먼트로 구현
- ▶ 자동 생성된 fragment\_photo.xml파일은 FrameLayout 안에 텍스트 뷰가 하나 배치되어 있음



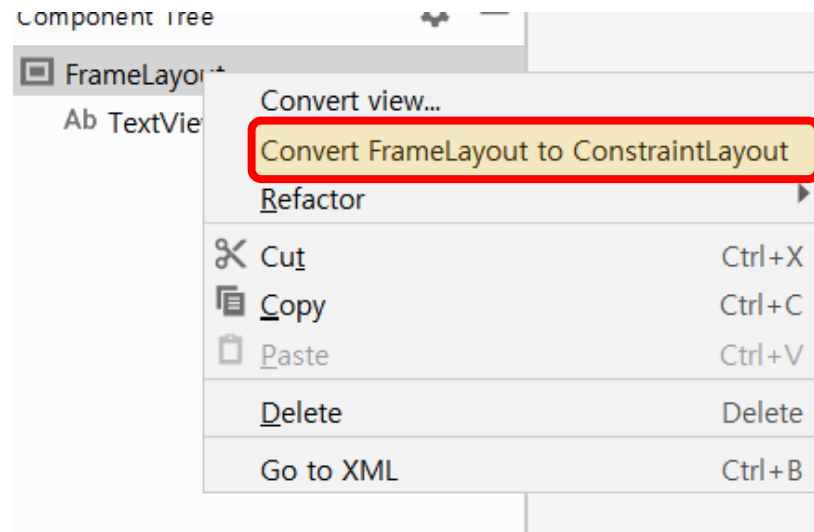
# 3초마다 사진이 바뀌는 전자액자 앱

## ▶ 프래그먼트의 레이아웃 수정

▶ FrameLayout을 ConstraintLayout으로 변경하고 이미지 뷰를 화면에 꽉 채우기

▶ 안드로이드 스튜디오에서는 기존 레이아웃 등을 ConstraintLayout으로 변환하는 방법을 제공

▶ 컴포넌트 트리 창의 FrameLayout을 우 클릭 - Convert FrameLayout to ConstraintLayout

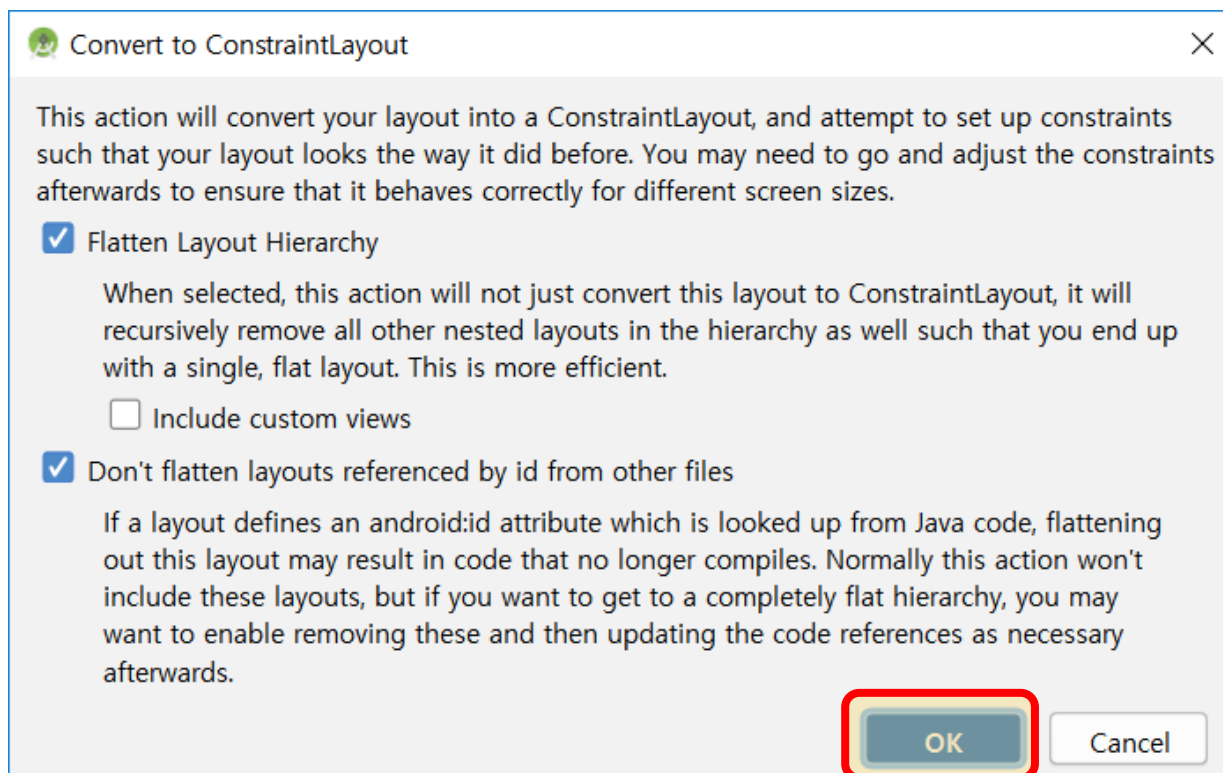


# 3초마다 사진이 바뀌는 전자액자 앱

## ▶ 프래그먼트의 레이아웃 수정

▶ FrameLayout을 ConstraintLayout으로 변경하고 이미지 뷰를 화면에 꽉 채우기

▶ 옵션을 선택하는 화면이 표시되면 기본값 그대로 OK 클릭



# 3초마다 사진이 바뀌는 전자액자 앱

## ▶ 프래그먼트의 레이아웃 수정

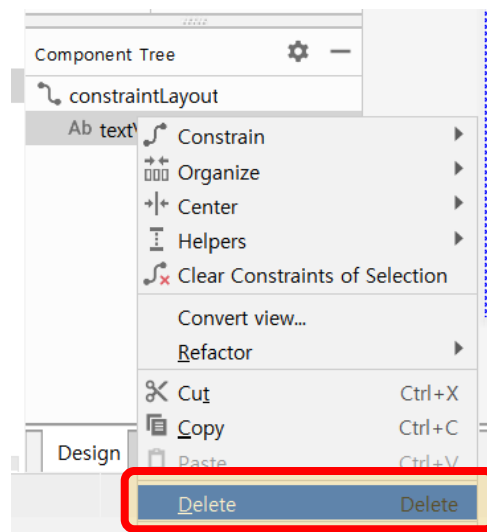
### ▶ FrameLayout을 ConstraintLayout으로 변경하고 이미지 뷰를 화면에 꽉 채우기

#### ▷ ConstraintLayout으로 변환

- 아직 컴포넌트 트리에는 frameLayout이라고 표시되는데 ID가 frameLayout으로 지정되어 있기 때문임
- id를 constraintLayout으로 변경



#### ▷ 컴포넌트 트리 창에서 textView 삭제

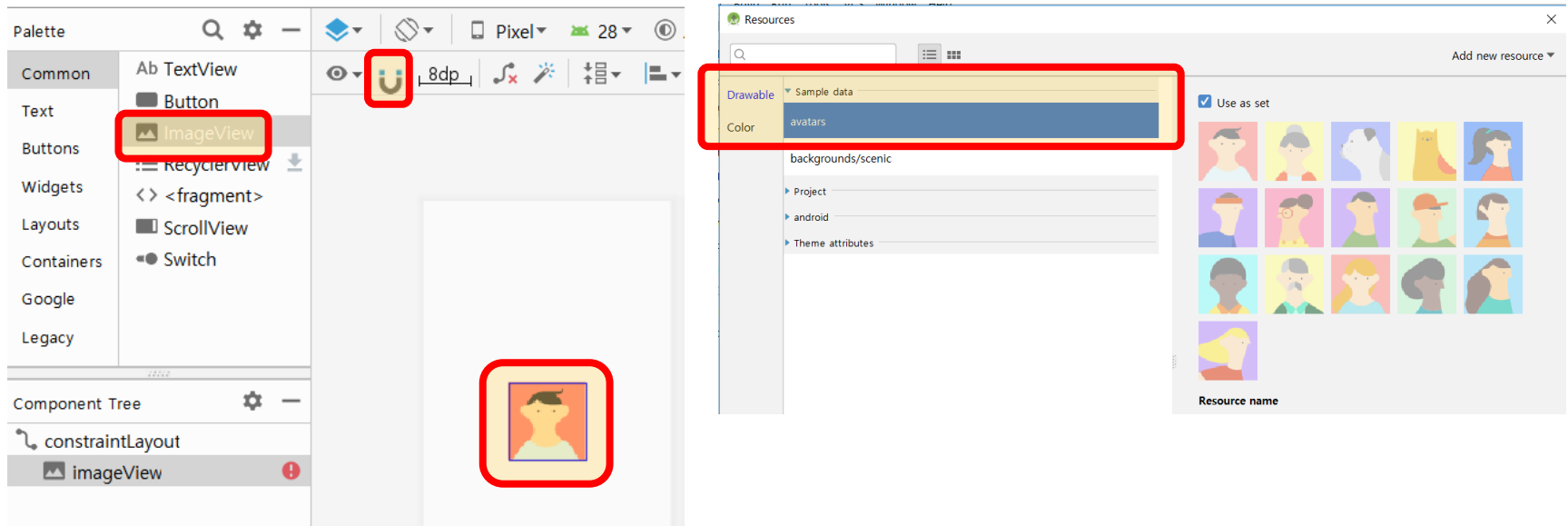


# 3초마다 사진이 바뀌는 전자액자 앱

## ▶ 프래그먼트의 레이아웃 수정

### ▶ 이미지 뷰를 배치

- ▶ 이미지 리소스를 선택하는 화면이 표시되면 샘플 이미지를 선택 - sample data는 스튜디오 3.2 부터 제공
- ▶ [Drawable] - [Sample Data] - [avatars] - 아무거나 선택
- ▶ 샘플 이미지는 디자인하는 동안에만 표시되고 실제로는 적용되지 않음.



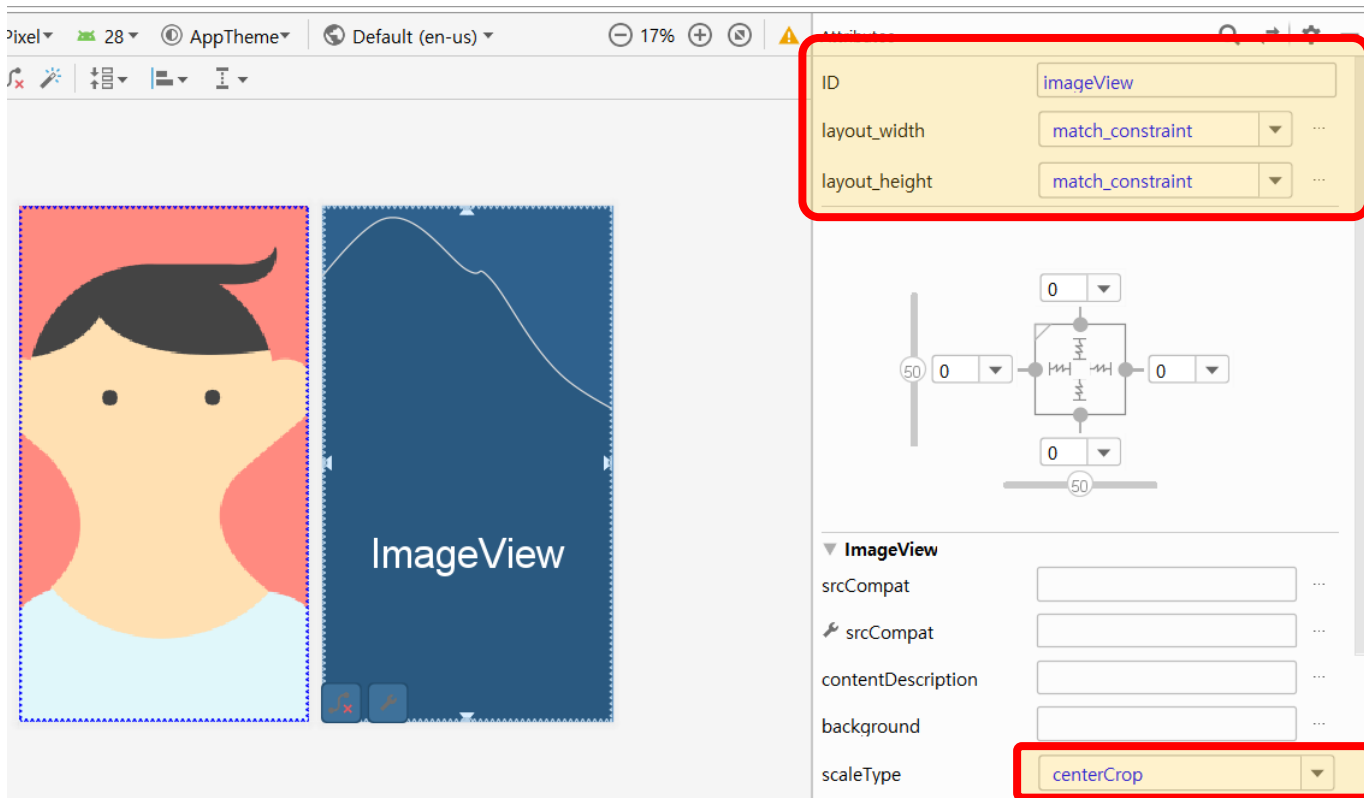
# 3초마다 사진이 바뀌는 전자액자 앱

## ▶ 프래그먼트의 레이아웃 수정

### ▶ 뷰의 속성을 아래와 같이 설정

▶ scaleType 속성 : ImageView의 상태에 따라 이미지 크기 조절 또는 이동

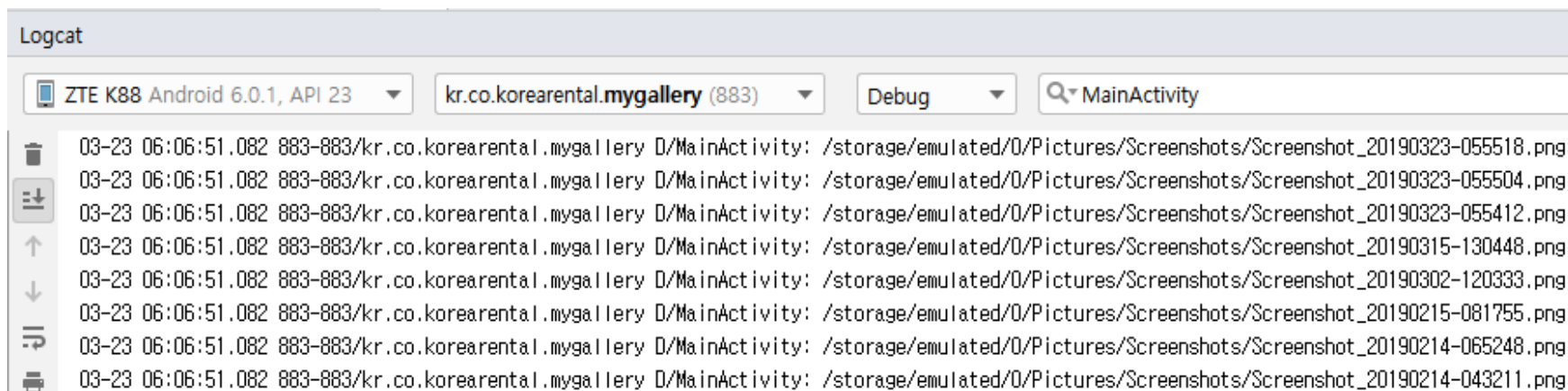
▶ centerCrop 값 : 이미지가 잘리더라도 원본 비율을 유지하며 ImageView를 가득 채움



# 3초마다 사진이 바뀌는 전자액자 앱

## ▶ Glide 라이브러리 설정

- ▶ 앞서 ContentResolver를 통하여 MediaStore에서 사진의 정보를 얻었으므로 이를 setImageURI() 메서드를 사용해서 이미지 뷰에 표시할 수 있음



## ▶ 기존의 이미지 표시 방법

- ▶ imageView.SetImageURI(Uri.parse("경로"))
- ▶ 이미지를 표시할 때는 이 메서드보다 Glide 라이브러리를 사용하는 방법을 추천
  - ▶ Glide 라이브러리는 사용하지 않는 리소스를 자동으로 해제하고 메모리를 효율적으로 관리



# 3초마다 사진이 바뀌는 전자액자 앱

## ▶ Glide 라이브러리 설정

▶ 두 가지 중 한가지 방법을 선택하여 build.gradle 파일에 의존성을 추가

▷ 첫 번째 방법 : Anko 라이브러리 의존성 추가와 동일 한 방법

- implementation 'com.github.bumptech.glide:glide:4.9.0'
- build.gradle(module) 파일을 수정했다면 Sync Now 링크를 클릭하여 싱크

▷ 관련 정보 확인

- <https://github.com/bumptech/glide>

▷ 최신 버전 확인

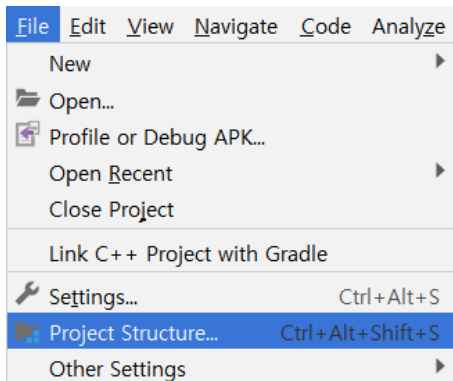
- <http://bumptech.github.io/glide/doc/download-setup.html>

# 3초마다 사진이 바뀌는 전자액자 앱

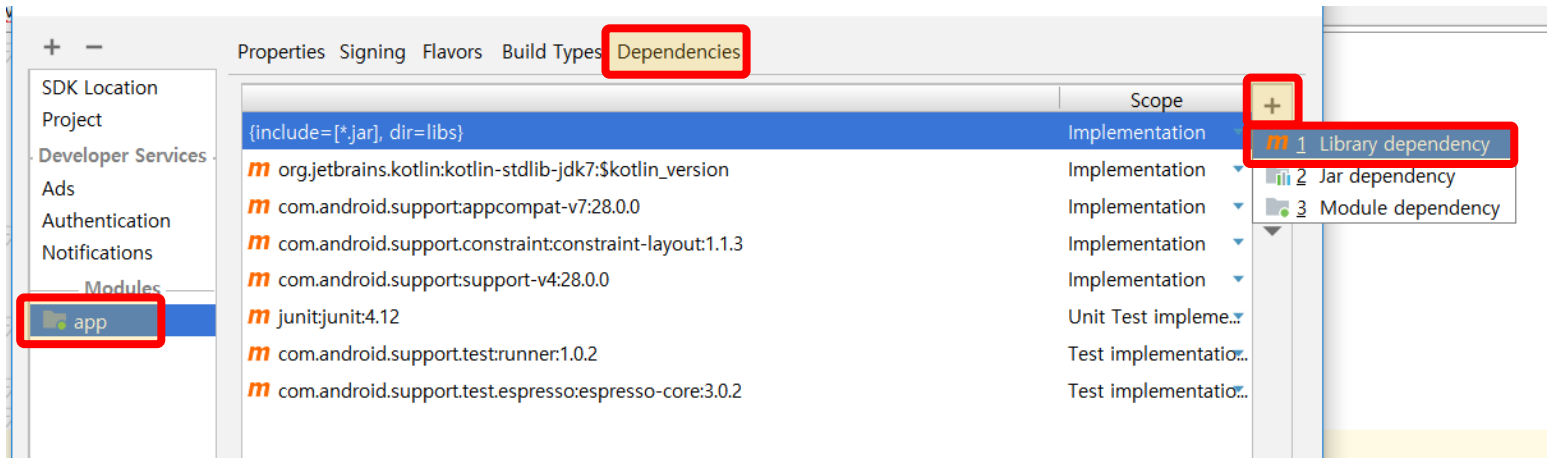
## ▶ Glide 라이브러리 설정

▶ 두 번째 방법 :안드로이드 스튜디오의 메뉴를 활용

▶ File → Project Structure



▶ App 모듈 - Dependencies 탭 - '+' 기호 - Library dependency



# 3초마다 사진이 바뀌는 전자액자 앱

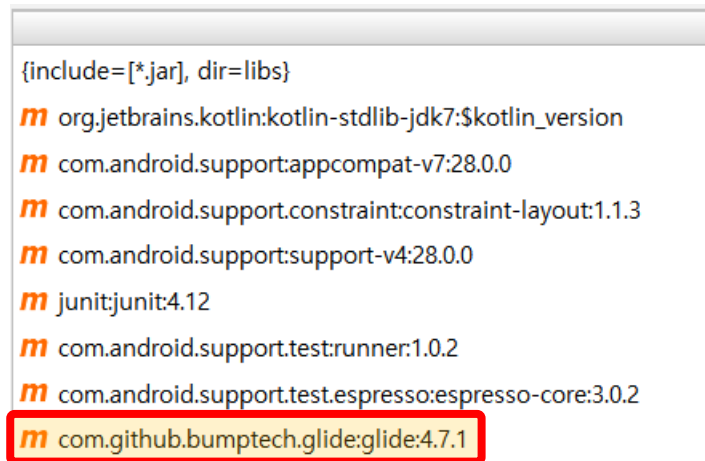
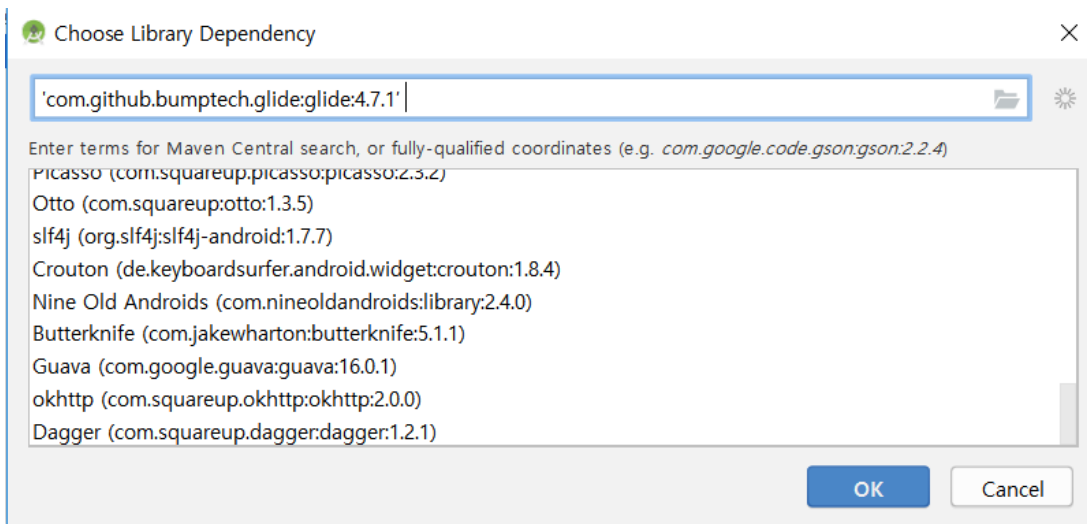
## ▶ Glide 라이브러리 설정

▶ 두 번째 방법 :안드로이드 스튜디오의 메뉴를 활용(계속)

▶ 잠시 후 com.github.bumptech.glide:glide를 선택하고 OK 클릭

■ 없으면 직접 입력 : com.github.bumptech.glide:glide:4.9.0

▶ Glide 라이브러리가 추가되었는지 확인하고 OK 클릭



# 3초마다 사진이 바뀌는 전자액자 앱

## ▶ 프래그먼트에 사진 표시하기

### ▶ PhotoFragment.kt에서 작성

#### ▷ 확인 후 없는 부분만 작성

```
private const val ARG_URI = "uri"
```

클래스 밖에 const 키워드를 선언하면 컴파일 시간에 결정되는 상수가 됨  
private 접근제한자로 설정되어 PhotoFragment.kt 파일내에서만 사용 가능

```
class PhotoFragment : Fragment() {  
    private var uri: String? = null  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
  
        arguments?.let {  
            uri = it.getString(ARG_URI)  
        }  
    }  
}
```

ARG\_URI키 값에 저장된 uri 정보를 변수에 저장

프래그먼트가 생성되면 onCreate메소드가 호출되고  
ARG\_URI키에 저장된 uri정보를 가져와서 변수에 저장

# 3초마다 사진이 바뀌는 전자액자 앱

## ▶ 프래그먼트에 사진 표시하기

### ▶ PhotoFragment.kt에서 작성

- ▷ newInstance()를 이용하여 프래그먼트를 생성할 수 있고 uri 전달
- ▷ 위의 uri는 Bundle 객체에 ARG\_URI키로 저장되고 value는 Fragment객체의 arguments 프로퍼티에 저장
  - Bundle 클래스는 Map 클래스이며 안드로이드에서 구조체처럼 사용됨
  - putString(key, value) //액티비티는 Intent로 데이터를 전달하며 프래그먼트는 Bundle을 사용

```
companion object { 코틀린에서 정적 메소드 생성
    @JvmStatic
    fun newInstance(uri: String) =
        PhotoFragment().apply {
            arguments = Bundle().apply {
                putString(ARG_URI, uri)
            }
        }
}
```

# 3초마다 사진이 바뀌는 전자액자 앱

## ▶ Object 클래스

- ▶ 코틀린에서의 익명 클래스는 이름이 없는 클래스를 의미하며 실제 개발에서 많이 사용
- ▶ 클래스를 선언하지만 클래스명이 없이 선언
  - ▷ 이름이 없기 때문에 반복해서 객체를 생성하는 것은 불가능하면 생성과 동시에 객체를 사용
  - ▷ 코드상에서 객체를 단 한번만 생성할 경우 사용
  - ▷ 단 한번 생성하기 때문에 굳이 정식명으로 클래스를 생성하는 것이 번거롭고 프로그래밍이 복잡해지기 때문에 사용
- ▶ class 예약어를 사용하지 않고 object 예약어로 정의
- ▶ 익명클래스의 위치는 상관없음

```
package kr.co.korearental.mygallery

val obj1 = object{
    var no1: Int = 10
    fun myFun(){
    }
}
```

```
class Outer {
    val obj2 = object {
        var no2: Int = 10
        fun myFun(){
        }
    }
}
```

# 3초마다 사진이 바뀌는 전자액자 앱

## ▶ Object 클래스의 형태

```
val obj = object{  
}
```

```
object 클래스명{  
}
```

▶ 두 번째와 같이 선언하면 클래스 선언과 동시에 클래스명과 같은 이름의 객체를 생성

# 3초마다 사진이 바뀌는 전자액자 앱

## ▶ Object 클래스 예제

### ▶ object클래스의 객체 생성 가능여부와 접근 방법

```
class NormalClass {  
    fun myFun(){ }  
}  
  
object ObjectClass {  
    fun myFun() { }  
}  
  
fun main(args: Array<String>) {  
    val obj1: NormalClass = NormalClass()  
    val obj2: NormalClass = NormalClass()  
    obj1.myFun()  
  
    //    val obj3: ObjectClass = ObjectClass()//error, 생성자를 사용할 수 없음  
  
    ObjectClass.myFun()  
}
```



# 3초마다 사진이 바뀌는 전자액자 앱

## ▶ 싱글톤 패턴(Singleton)

- ▶ 프로그램 전체에서 객체를 하나만 만들어서 사용해야 될 경우 싱글톤으로 디자인
- ▶ 외부에서 직접 생성하지 못하도록 생성자를 private으로 선언하고 내부에 static 함수를 두어 그 함수에서 대신 생성하는 형태로 작성
- ▶ object클래스는 코틀린으로 싱글톤 패턴의 클래스를 생성할 때 사용
- ▶ JAVA 싱글톤 패턴

```
public class Test{  
    private static Test obj;  
    private Test(){  
    }  
    static Test getInstance(){  
        if(obj == null) obj = new Test();  
        return obj;  
    }  
}
```

## ▶ Kotlin 싱글톤 패턴

```
object Test{  
}
```

# 3초마다 사진이 바뀌는 전자액자 앱

▶ 특정 내부에 이름있는 object클래스를 선언하였을 경우

▶ 이름이 있는 object 클래스를 최상위에 작성하지 않고 특정 클래스 안에 작성할 수도 있음

▶ object 클래스를 내부의 멤버에 접근할 경우 외부 클래스의 객체가 아닌 클래스명으로 접근

```
class Outer {  
    object NestedClass {  
        val no: Int = 0  
        fun myFun() { }  
    }  
}  
  
fun main(args: Array<String>) {  
    val obj=Outer()  
    //    obj.NestedClass.no//error  
  
    Outer.NestedClass.no  
    Outer.NestedClass.myFun()  
}
```

```
class NormalClass {  
    fun myFun(){ }  
}  
  
object ObjectClass {  
    fun myFun() { }  
}  
  
fun main(args: Array<String>) {  
    val obj1: NormalClass = NormalClass()  
    val obj2: NormalClass = NormalClass()  
    obj1.myFun()  
  
    //    val obj3: ObjectClass =  
    ObjectClass()//error, 생성자를 사용할 수 없  
    음  
  
    ObjectClass.myFun()  
}
```

# 3초마다 사진이 바뀌는 전자액자 앱

## ▶ companion 예약어

- ▶ object 클래스 이름을 생략하고 외부 클래스의 이름으로 접근하기 위하여 object 예약어 앞에 companion을 입력

```
class Outer {  
    companion object NestedClass {  
        val no: Int = 0  
        fun myFun() { }  
    }  
}  
  
fun main(args: Array<String>) {  
    Outer.NestedClass.no  
    Outer.NestedClass.myFun()  
  
    Outer.no  
    Outer.myFun()  
}
```

# 3초마다 사진이 바뀌는 전자액자 앱

## ▶ 프래그먼트에 사진 표시하기

### ▶ PhotoFragment.kt에서 작성

▶ newInstance()를 이용하여 프래그먼트를 생성할 수 있고 uri 전달

▶ 이 값은 Bundle 객체에 ARG\_URI키로 저장되고 arguments 프로퍼티에 저장

```
companion object { 코틀린에서 정적 메소드 생성
    @JvmStatic
    fun newInstance(uri: String) =
        PhotoFragment().apply {
            arguments = Bundle().apply {
                putString(ARG_URI, uri)
            }
        }
}
```

▶ 사진 경로를 Cursor 객체로부터 가져올 때마다 PhotoFragment.newInstance(uri)로 프래그먼트를 생성

# 3초마다 사진이 바뀌는 전자액자 앱

## ▶ 어노테이션(Annotation)

- ▶ 코드에 부가 정보를 추가하기 위하여 클래스, 함수, 프로퍼티 선언 앞에 작성하는 구문
- ▶ @기호로 시작
- ▶ 목적
  - ▷ 컴파일러에게 코드 문법 에러를 체크하기 위한 정보를 제공
  - ▷ 개발 툴이나 빌더에게 코드 자동 추가를 위한 정보 제공
  - ▷ 실행 시 특정 기능을 실행하기 위한 정보 제공
- ▶ 대부분 제공하는 어노테이션을 상용하지만 직접 만들어 사용 가능

## ▶ @JvmStatic

- ▶ companion object를 사용하여 앞의 예제와 같이 구성한 코드를 자바에서 사용하려면 속성 및 함수가 자바의 필드/메서드로 해석되도록 해야함
- ▶ 코틀린에서 함수는 @JvmStatic 어노테이션을 사용해야 자바코드에서 이 함수를 정적 메서드로 사용할 수 있음

# 3초마다 사진이 바뀌는 전자액자 앱

## ▶ 프래그먼트에 사진 표시하기

### ▶ PhotoFragment.kt에서 작성

▶ newInstance()를 이용하여 프래그먼트를 생성할 수 있고 uri 전달

▶ 이 값은 Bundle 객체에 ARG\_URI키로 저장되고 arguments 프로퍼티에 저장

```
companion object { 코틀린에서 정적 메소드 생성
    @JvmStatic
    fun newInstance(uri: String) =
        PhotoFragment().apply {
            arguments = Bundle().apply {
                putString(ARG_URI, uri)
            }
        }
}
```

▶ 사진 경로를 Cursor 객체로부터 가져올 때마다 PhotoFragment.newInstance(uri)로 프래그먼트를 생성

# 3초마다 사진이 바뀌는 전자액자 앱

## ▶ 프래그먼트에 사진 표시하기

### ▶ PhotoFragment.kt에서 작성

#### ▷ 확인 후 없는 부분만 작성

onCreateView()에서 프래그먼트에 표시될 뷰 생성

```
override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,
    savedInstanceState: Bundle?): View? {
    레이아웃이 아닌 곳에서 xml을 활성화 시키려면 인플레이트 해야 함
    return inflater.inflate(R.layout.fragment_photo, container, false)
}
```

뷰가 생성된 직후 호출되는 onCreateView(view: 생성된 뷰, savedInstanceState: 상태를 저장하는 객체)

```
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)
    Glide.with(this).load(uri).into(imageView) Glide라이브러리를 사용하여 사진을 이미지 뷰에 표시
}
Glide.with(this)로 사용 준비를 하고 load() 메서드에 uri값을 인자로 주고 해당 이미지를 부드럽게 로딩
이미지가 로딩되면 into() 메서드로 imageView 에 표시하기
```

코드는 한 줄이지만 성능은 매우 향상됨

# 3초마다 사진이 바뀌는 전자액자 앱

## ▶ 액티비티에 ViewPager 추가

- ▶ 뷰페이저 : 여러 프래그먼트들을 좌우로 슬라이드하는 뷰
- ▶ 뷰페이저를 사용하려면 데이터, 어댑터, 뷰 세 가지가 필요한 어댑터 패턴을 구현해야 함
  - 데이터 : 프래그먼트 (화면)
  - 어댑터 : 프래그먼트를 어느 화면에 표시할 것인지 관리하는 객체
  - 뷰 : 뷰페이저



# 3초마다 사진이 바뀌는 전자액자 앱

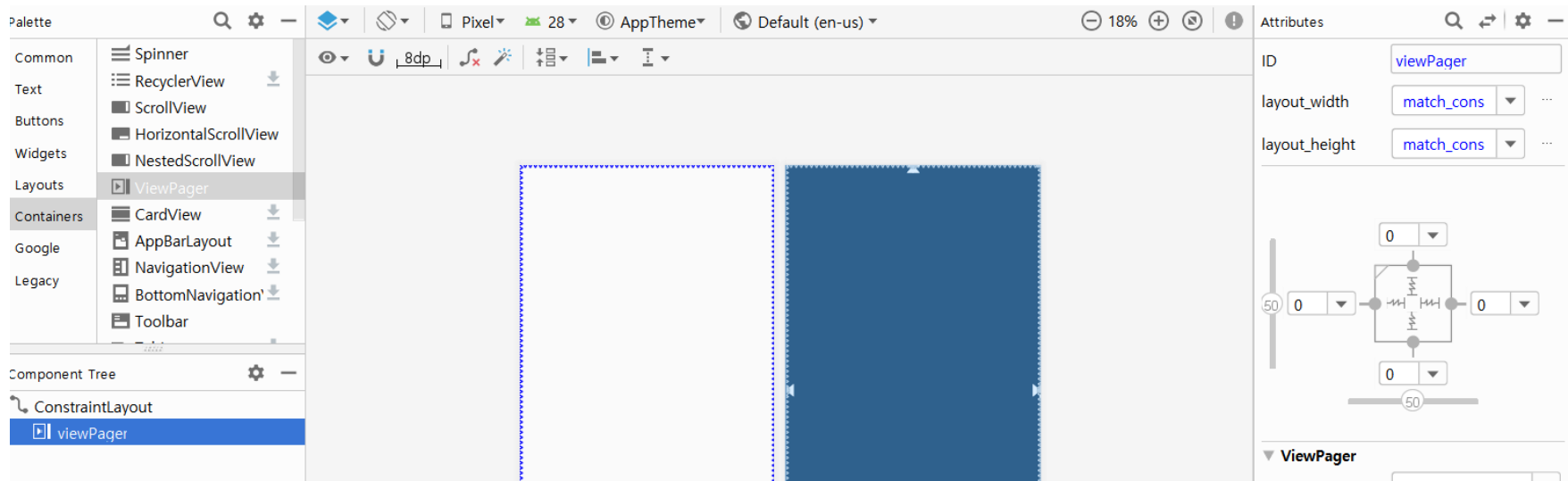
## ▶ 액티비티의 레이아웃에 ViewPager를 추가

▶ ViewPager 는 별도의 라이브러리에 포함된 뷰이기 때문에 의존성을 추가해야 함

▷ 디자인 창을 사용하면 의존성을 자동 추가하므로 디자인 창 사용

▷ 뷰페이저를 추가할때 suppor-v4 라이브러리의 의존성이 필요하다는 화면이 표시되면 OK

■ 이미 의존성이 추가되어 있으면 이 화면은 표시되지 않음



# 3초마다 사진이 바뀌는 전자액자 앱

▶ ViewPager 에 표시할 내용을 정의하려면 어댑터 필요

▶ 어댑터는 아이테의 목록 정보를 가진 객체

▶ FragmentPagerAdapter : 페이지 내용이 영구적일 때 적합 / 한 번 로딩한 페이지는 메모리에 보관하기 때문에 빠르지만 페이지가 많으면 많은 메모리를 사용

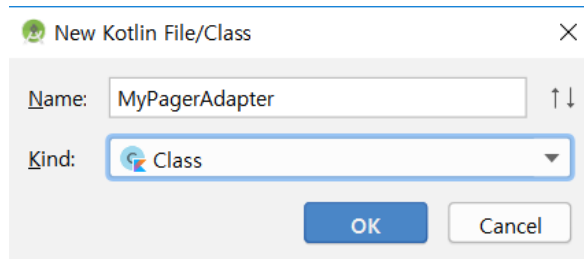
▶ FragmentStatePagerAdapter : 많은 수의 페이지가 있을 때 적합 / 보이지 않는 페이지를 메모리에서 제거할 수 있기 때문에 상대적으로 적은 메모리를 차지

▶ 예제에서는 FragmentStatePagerAdapter 사용

# 3초마다 사진이 바뀌는 전자액자 앱

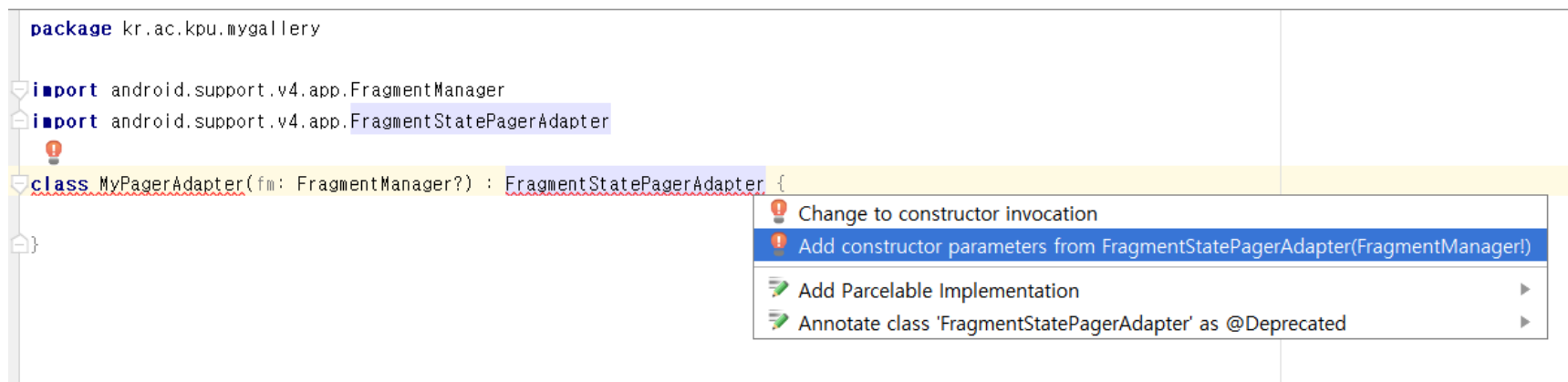
## ▶ 새로운 클래스 작성

### ▶ File->New->Kotlin File/Class



### ▶ FragmentStatePagerAdapter 클래스를 상속

- ▶ 슈퍼 클래스의 생성자를 추가하는 Add constructor parameters from FragmentStatePagerAdapter(FragmentManager!) 선택

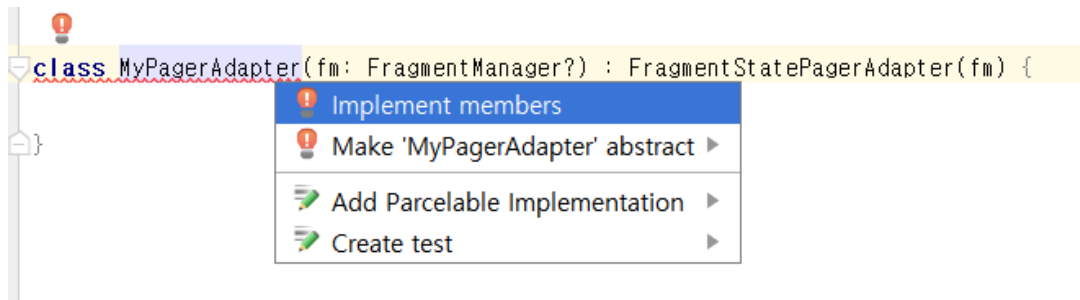


# 3초마다 사진이 바뀌는 전자액자 앱

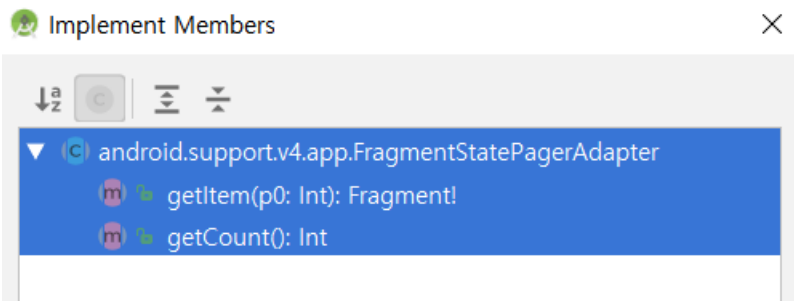
## ▶ 새로운 클래스 작성

▶ 생성자 파라미터가 추가되고 다시 빨간 줄이 표시되는 클래스 이름에 커서를 두고 단축키 Alt + Enter 누르기

▶ 미구현된 멤버를 구현하는 Implement members 를 클릭



▶ 모든 메서드 선택



# 3초마다 사진이 바뀌는 전자액자 앱

## ▶ 자동으로 생성된 코드 수정

```
class MyPagerAdapter(fm: FragmentManager?) : FragmentStatePagerAdapter(fm) {  
    // 뷰페이저가 표시할 프래그먼트 목록  
    private val items = ArrayList<Fragment>() 어댑터가 프래그먼트 목록을 가지도록 함  
                                           이 목록은 updateFragment()로 외부에서 추가할 수 있음  
  
    // position 위치의 프래그먼트  
    override fun getItem(position: Int): Fragment {  
        return items[position] position에 어떤 프래그먼트를 표시할지 정의  
    }  
  
    // 아이템의 갯수  
    override fun getCount(): Int { 아이템(프래그먼트)개수를 정의  
        return items.size  
    }  
  
    // 아이템 갱신  
    fun updateFragments(items : List<Fragment>) {  
        this.items.addAll(items)  
    }  
}
```

# 3초마다 사진이 바뀌는 전자액자 앱

## ▶ 전자액자 완성

- ▶ MainActivity.kt 파일을 열고 getAllPhotos() 메서드 수정

- ▷ 프래그먼트를 아이템으로 하는 ArrayList 를 생성

- ▷ 사진을 Cursor 객체로부터 가져올 때마다 PhotoFragment.newInstance(uri)로 프래그먼트를 생성하면서 fragments 리스트에 추가

# 3초마다 사진이 바뀌는 전자액자 앱

## ▶ 전자액자 완성

### ▶ MainActivity.kt 파일을 열고 getAllPhotos( ) 메서드 수정

```
private fun getAllPhotos() {  
  
    val fragments = ArrayList<Fragment>() // 프래그먼트를 아이템으로 하는 ArrayList 생성  
    if (cursor != null) {  
        while (cursor.moveToNext()) {  
            // 사진 경로 Uri 가지고 오기  
            val uri = cursor.getString(cursor.getColumnIndexOrThrow(MediaStore.Images.Media.DATA))  
            Log.d("MainActivity", uri)  
            fragments.add(PhotoFragment.newInstance(uri)) PhotoFragment.newInstance(uri)로 프래그먼트를  
생성하면서 fragment에 리스트에 추가  
        }  
        cursor.close() |  
    }  
  
    // 어댑터  
    val adapter = MyPagerAdapter(supportFragmentManager)  
    adapter.updateFragments(fragments)  
    viewPager.adapter = adapter  
}  
  
MyPagerAdapter 를 생성하면서 프래그먼트 매니저를 생성자의 인자로 전달해야 함  
프래그먼트 매니저는 getSupportFragmentManager () 메서드로 가져올 수 있고  
supportFragmentManager 프로퍼티로 접근할 수 있음  
어댑터를 생성하면 updateFragments() 메서드를 사용하여 프래그먼트 리스트를 전달  
어댑터를 viewPager에 설정
```

### ▶ 앱 실행하기

# 3초마다 사진이 바뀌는 전자액자 앱

▶ 사진이 3초마다 자동으로 슬라이드 되는 기능 추가

▶ 구현 순서

- 1) timer로 3초마다 코드 실행하기
- 2) runOnUiThread로 timer 내부에서 UI 조작하기



# 3초마다 사진이 바뀌는 전자액자 앱

▶ MainActivity 파일의 getAllPhotos () 메서드의 마지막에 다음 코드를 추가

// 3초마다 자동으로 슬라이드

```
timer(period = 3000) { 3초마다 실행되는 타이머를 생성
    runOnUiThread { timer가 백그라운드 스레드로 동작해 UI를 변경하도록 runOnUiThread로 코드 감싸기
        if (viewPager.currentItem < adapter.count - 1) {
            viewPager.currentItem = viewPager.currentItem + 1
        } else {
            viewPager.currentItem = 0
        }
    }
}
```

---

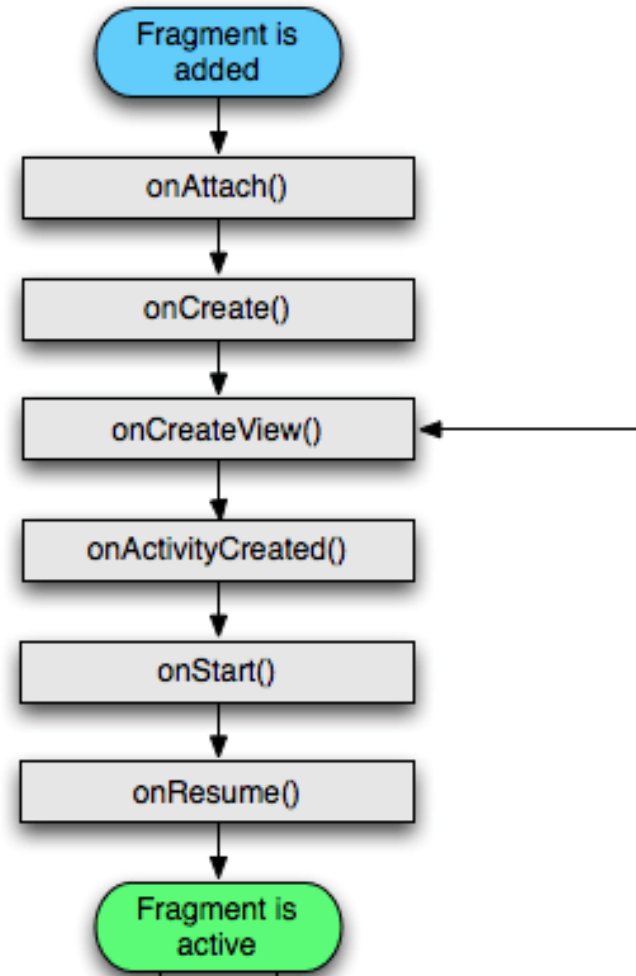
**Q & A**

---

# 3초마다 사진이 바뀌는 전자액자 앱

## ▶ 프래그먼트의 기초

- ▶ `onAttach()` : 액티비티에 붙을 때 호출. 이때부터 액티비티의 참조를 사용할 수 있음
- ▶ `onCreate()` : 프래그먼트가 생성될 때 호출. 아직 레이아웃은 완성되기 전
- ▶ `onCreateView()` : 레이아웃을 생성하기 전에 호출. 완성된 뷰를 반환하게 되는데 아직 레이아웃이 완성되기 전
- ▶ `onActivityCreated()` : 액티비티의 `onCreate()` 메서드가 수행된 직후에 호출됨
- ▶ `onStart()` : 프래그먼트가 사용자에게 보여질 때 호출
- ▶ `onResume()` : 사용자와 상호작용하기 시작



# 3초마다 사진이 바뀌는 전자액자 앱

## ▶ 프래그먼트의 기초

- ▶ onPause() : 프래그먼트가 일시 중지이거나 더 이상 사용자와 상호작용하지 않음
- ▶ onStop() : 프래그먼트가 중지됨
- ▶ onDestroyView() : 프래그먼트가 해당 자원을 정리할 수 있도록 함
- ▶ onDestroy() : 프래그먼트가 파괴될 때 호출됨
- ▶ onDetach() : 프래그먼트가 액티비티에서 완전히 제거될 때 호출됨

