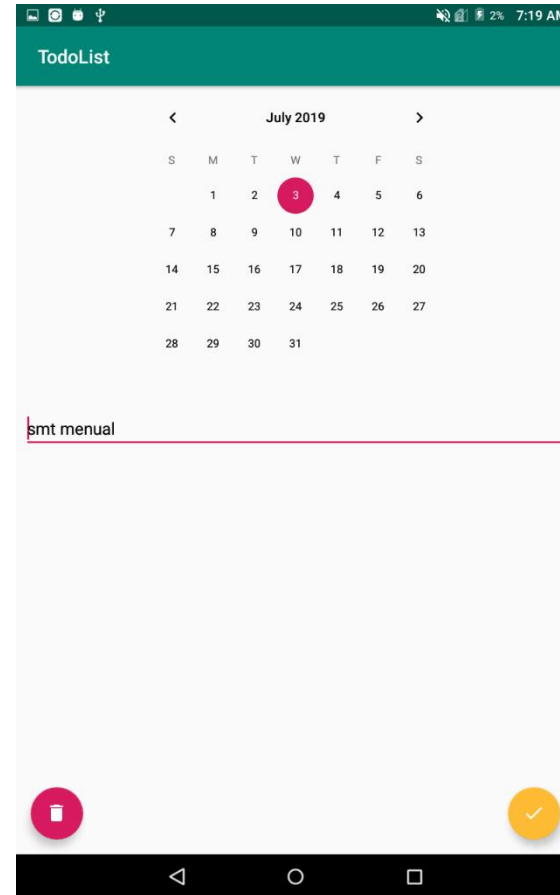
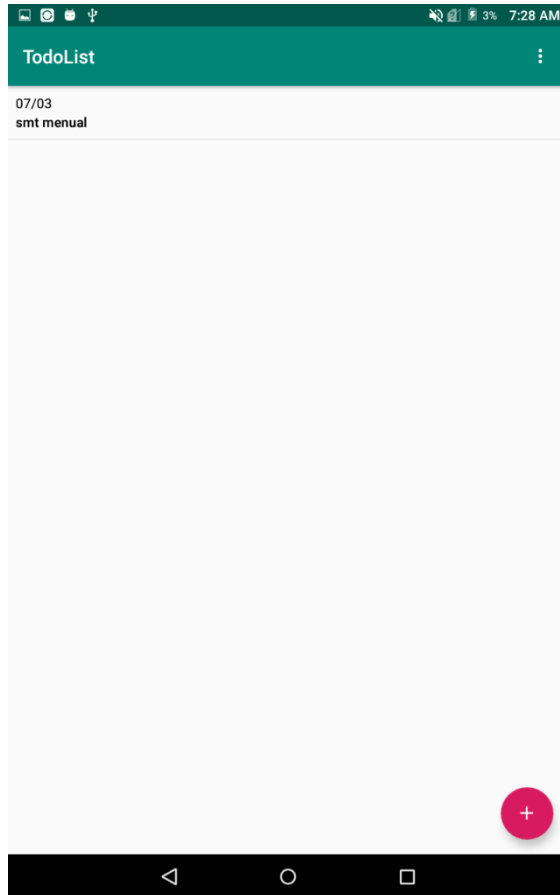

Kotlin을 이용한 Android 프로그래밍

Todo 리스트 앱 만들기

Contents

- I. Todo리스트 앱에서는 해야 할 일의 목록을 만들고 DB에 추가 및 수정, 삭제 작업을 할 수 있음



To do 리스트 앱 만들기

- ▶ 프로젝트 명 : Todo 리스트

- ▶ 앱의 기능

- ▶ 할 일의 목록을 표시하고 내용을 데이터베이스에 새로 추가하거나 수정, 삭제 작업을 할 수 있음

- ▶ 프로젝트 설계

- ▶ Todo 리스트 프로젝트는 2개의 화면으로 구성

- ▷ 할 일의 목록을 표시하는 화면

- ▷ 할 일을 추가, 수정, 삭제하는 화면

- ▶ 할 일 목록은 ListView를 사용하여 표현

- ▶ 데이터베이스로는 Realm 사용

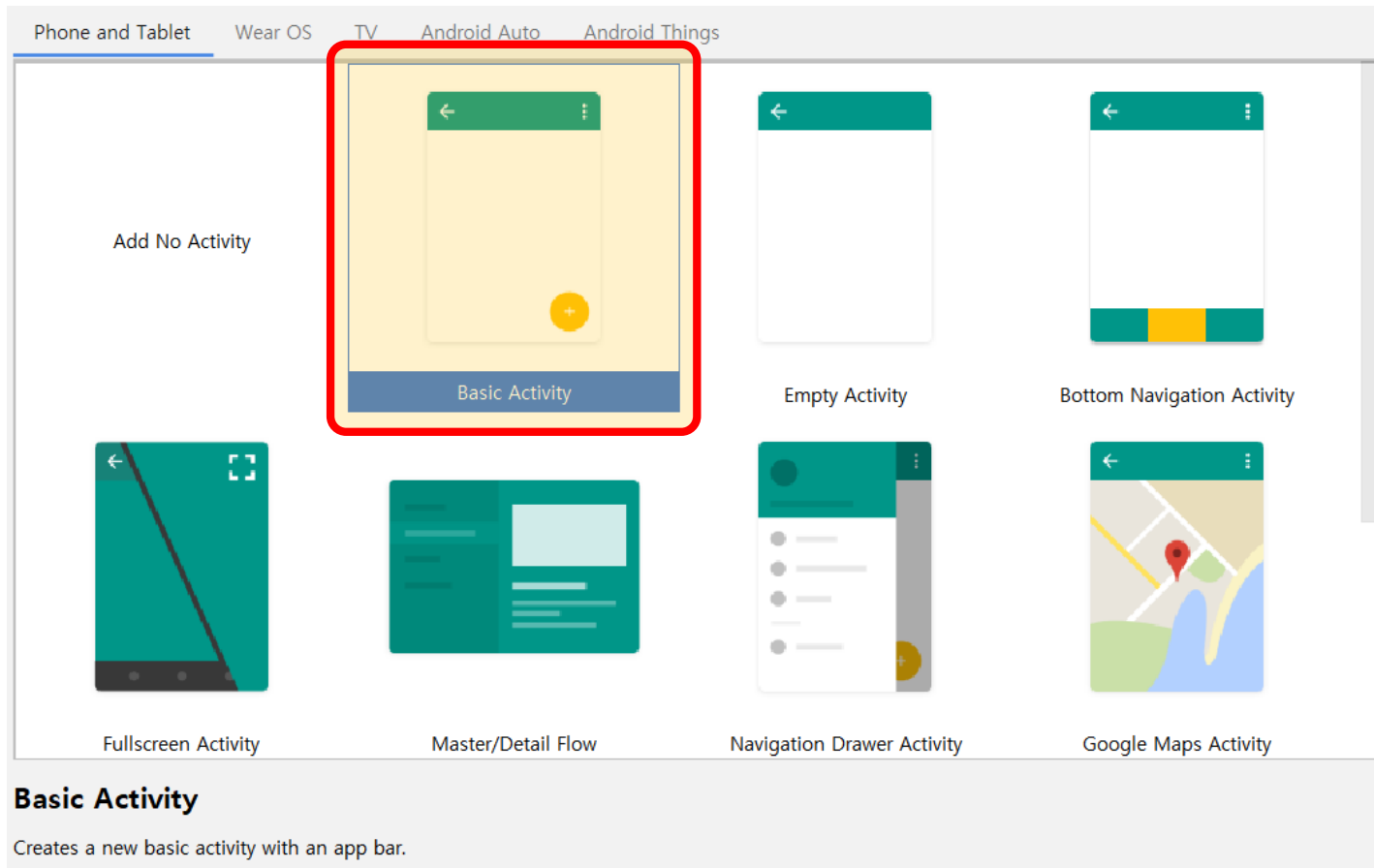
- ▷ SQL문법을 잘 몰라도 쉽게 데이터베이스를 사용 가능

- ▶ ListView의 어댑터에서 RealmBaseAdapter를 상속하면 DB와 ListView를 쉽게 연동할 수 있음

To do 리스트 앱 만들기

▶ Basic Activity으로 액티비티 선택

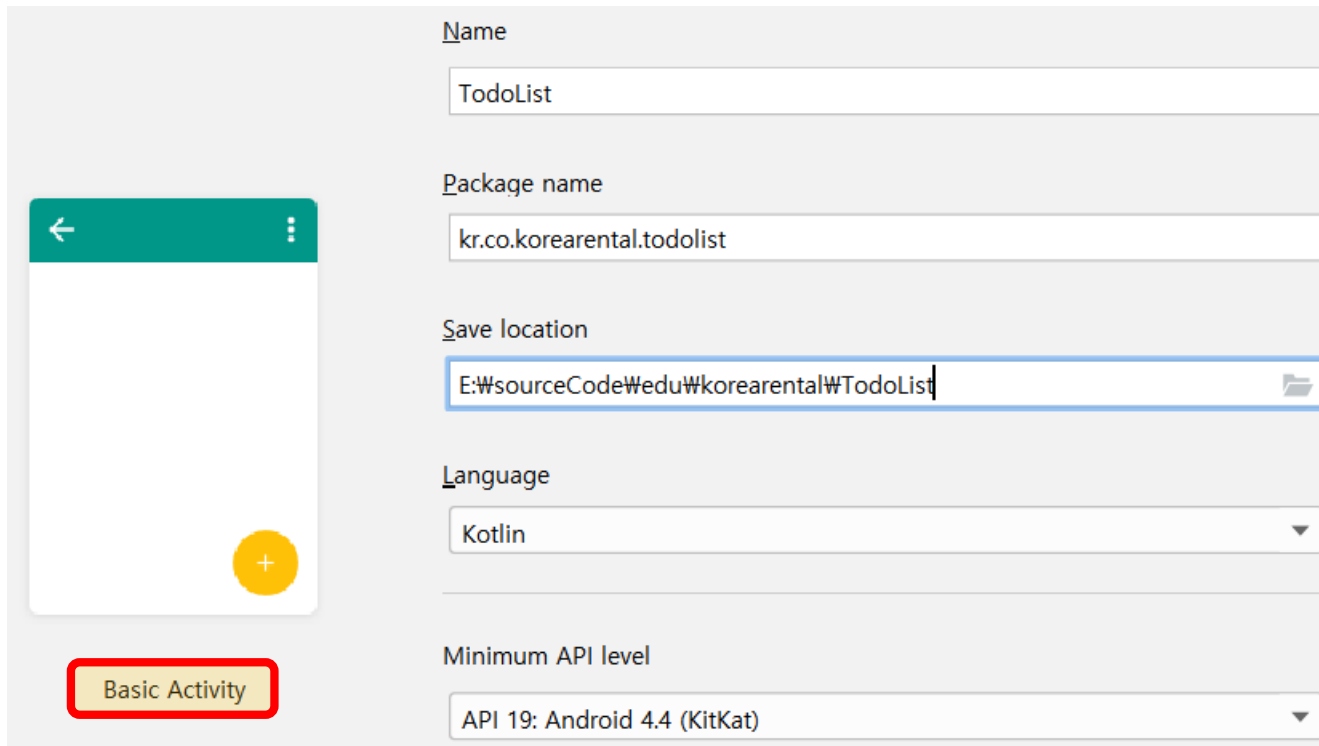
- ▶ Basic Activity는 플로팅 액션 버튼 FAB(Floating Action Button)과 메뉴가 미리 작성된 기본 액티비티



To do 리스트 앱 만들기

▶ 프로젝트 생성

- ▶ 프로젝트 명 : TodoList
- ▶ minSdkVersion : 19(Android 4.4 KitKat)
- ▶ 기본 액티비티 : BasicActivity



Name
TodoList

Package name
kr.co.korearental.todolist

Save location
E:\sourceCode\Wedu\Wkorearental\WTodoList

Language
Kotlin

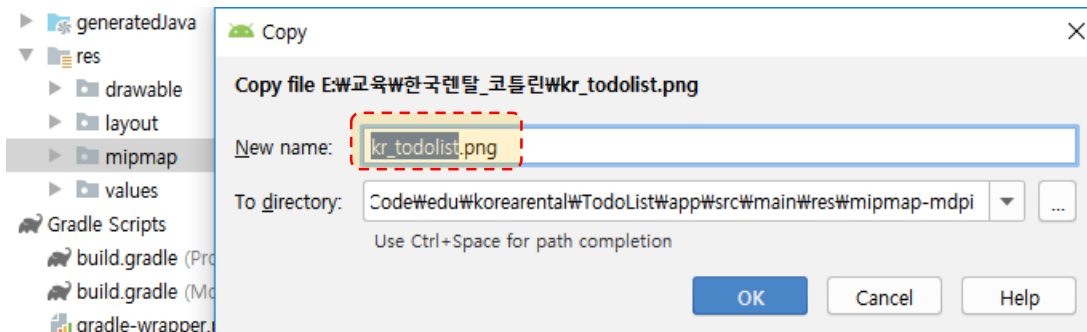
Minimum API level
API 19: Android 4.4 (KitKat)

To do 리스트 앱 만들기

▶ 프로젝트 설정

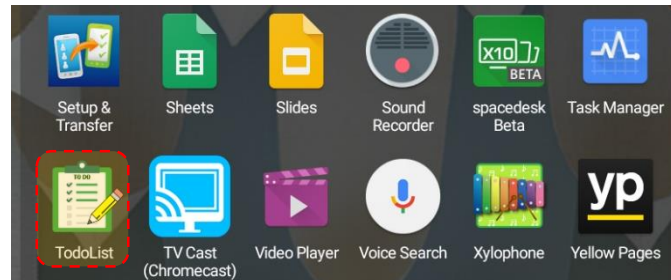
▶ kr_todolist.png로 아이콘 변경

- ▶ res/mipmap 에 아이콘 이미지 복사
- ▶ 안드로이드 앱에서 사용되는 이미지의 포맷은 대부분 png이며 jpg도 가능
- ▶ 파일명은 반드시 소문자 영문으로 작성



▶ 매니페스트 수정

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/kr_todolist"
    android:label="ToDoList"
    android:roundIcon="@mipmap/kr_todolist"
    android:supportRtl="true"
    android:theme="@style/AppTheme">
```



To do 리스트 앱 만들기

▶ Gradle에 Anko 라이브러리 의존성 추가

▶ 프로젝트 창에서 모듈 수준의 build.gradle 파일에 아래코드 삽입 후 sync 클릭

▶ implementation "org.jetbrains.anko:anko-commons:0.10.5"

▼ Gradle Scripts

build.gradle (Project: BmiCalculator)

build.gradle (Module: app)

gradle-wrapper.properties (Gradle Version)

proguard-rules.pro (ProGuard Rules for app)

gradle.properties (Project Properties)

settings.gradle (Project Settings)

local.properties (SDK Location)

dependencies {

implementation "org.jetbrains.anko:anko-commons:0.10.5"

implementation fileTree(dir: 'libs', include: ['*.jar'])

implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk7:\$kotlin_version"

implementation 'com.android.support:appcompat-v7:28.0.0'

implementation 'com.android.support.constraint:constraint-layout:1.1.3'

testImplementation 'junit:junit:4.12'

androidTestImplementation 'com.android.support.test:runner:1.0.2'

androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.2'

}

Gradle files have changed since last project sync. A project sync may be necessary for. Sync Now

To do 리스트 앱 만들기

▶ 벡터 드로어블 하위 호환 설정

▶ 안드로이드 5.0 미만의 기기에서도 벡터 이미지가 잘 표시되도록 모듈 수준의 그레이들 파일에 다음 코드를 추가한 후 싱크

▶ `vectorDrawables.useSupportLibrary = true`

```
7  android {  
8      compileSdkVersion 28  
9      defaultConfig {  
10         applicationId "kr.ac.kpu.bmicalculator"  
11         minSdkVersion 19  
12         targetSdkVersion 28  
13         versionCode 1  
14         versionName "1.0"  
15         testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"  
16  
17         vectorDrawables.useSupportLibrary = true  
18     }  
}
```

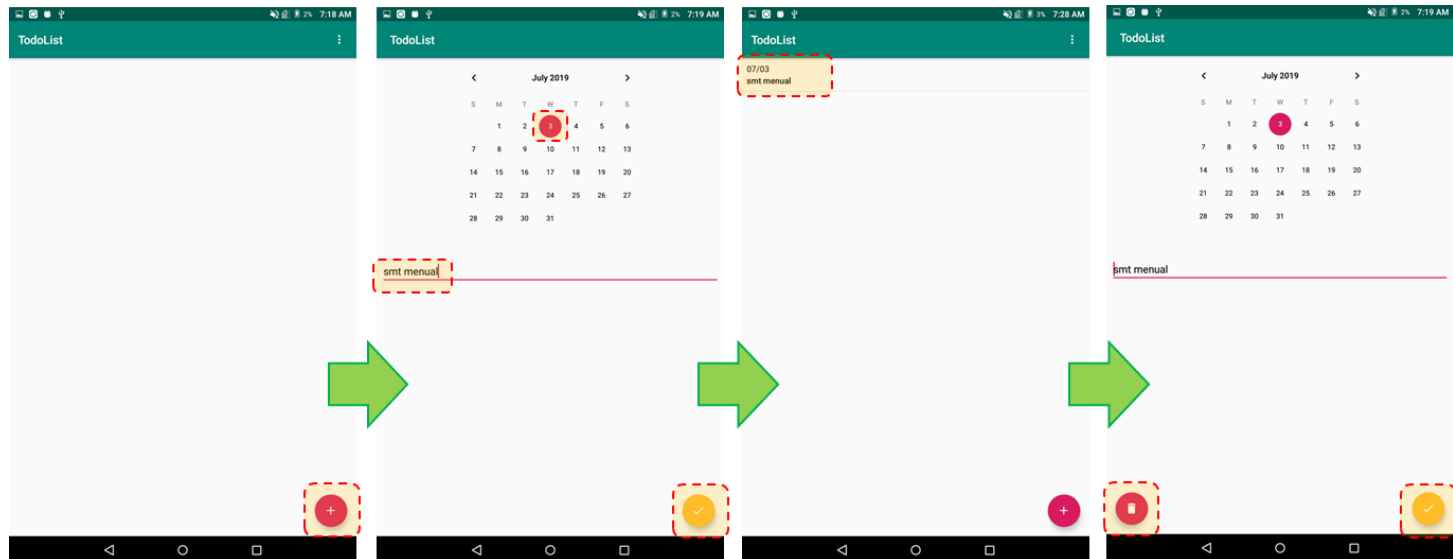
Gradle files have changed since last project sync. A project sync may be necessary for [Sync Now](#)

To do 리스트 앱 만들기

▶ 동작 분석

▶ 레이아웃의 구성요소를 분석

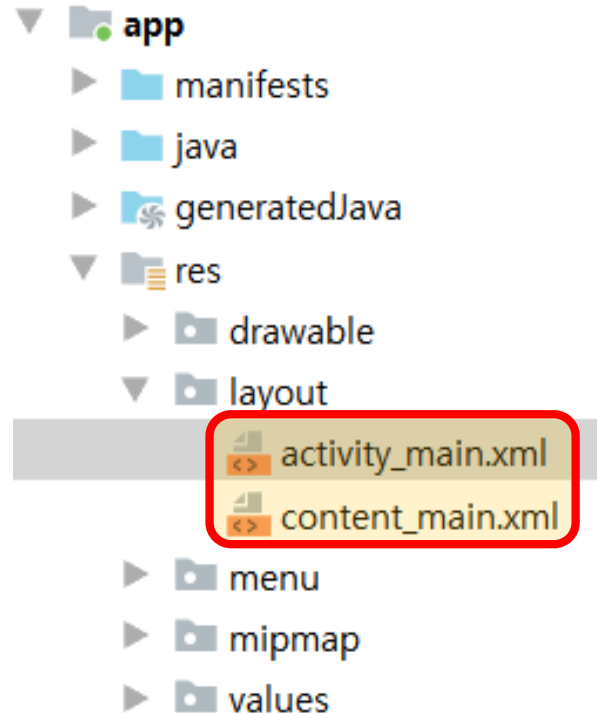
- ▶ 첫 번째 화면에는 할 일 목록을 표시하는 ListView를 표시
 - 새로운 할 일을 추가하려면 FAB를 클릭하여 두 번째 화면으로 이동
- ▶ 두 번째 화면에는 날짜를 선택하는 CalenderView를 표시
 - 할 일 내용은 EditText에 작성하고 완료되면 완료 FAB를 클릭
 - 첫 번째 액티비티에서 ListView(여러 항목 중에 하나)를 선택했을 경우, 삭제 버튼이 활성화됨



To do 리스트 앱 만들기

▶ BasicActivity 개요

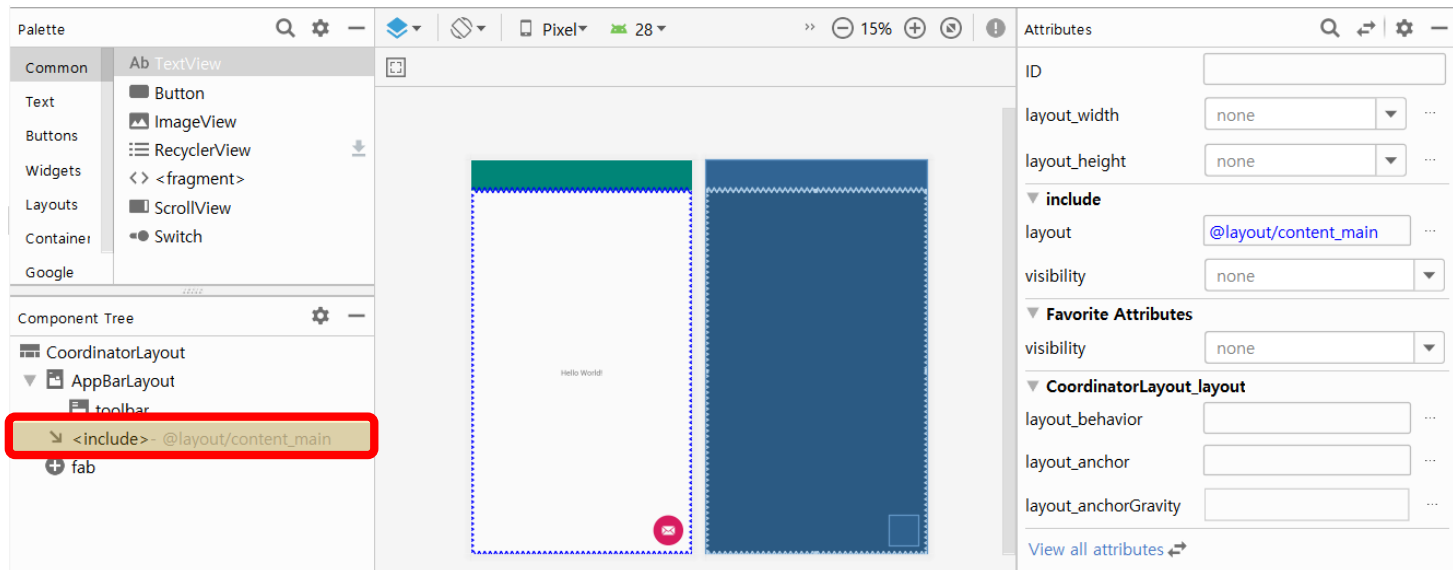
- ▶ 기본 액티비티로 Basic Activity를 선택하여 프로젝트를 생성했다면 레이아웃 파일이 activity_main.xml, content_main.xml 두 개가 생성됨



To do 리스트 앱 만들기

▶ BasicActivity 개요

- ▶ activity_main.xml의 컴포넌트 트리 창과 속성 창에서 content_main.xml 파일을 확인
- ▶ xml 파일은 다른 xml 파일을 include 속성으로 포함할 수 있음
 - ▶ 복잡한 화면을 구성할 때는 여러 xml 파일로 분리하여 레이아웃을 작성 가능
 - ▶ 또는 부분적으로 반복되는 디자인을 모든 액티비티에 적용할 경우 사용
- ▶ 이번 프로젝트의 activity_main.xml 파일은 레이아웃 전체를 구성하며 액션바, 플로팅 액션 버튼, content_main.xml 파일로 구성되어 있음



To do 리스트 앱 만들기

▶ 레이아웃 include 방법(예시)

- ▶ 텍스트 모드에서 포함하고자 하는 위치에 아래와 같이 코드 작성
- ▶ activity_main.xml에서 content_main.xml을 include하는 경우

```
<android.support.design.widget.AppBarLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:theme="@style/AppTheme.AppBarOverlay">

    <android.support.v7.widget.Toolbar
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="?attr/actionBarSize"
        android:background="?attr/colorPrimary"
        app:popupTheme="@style/AppTheme.PopupOverlay" />

</android.support.design.widget.AppBarLayout>

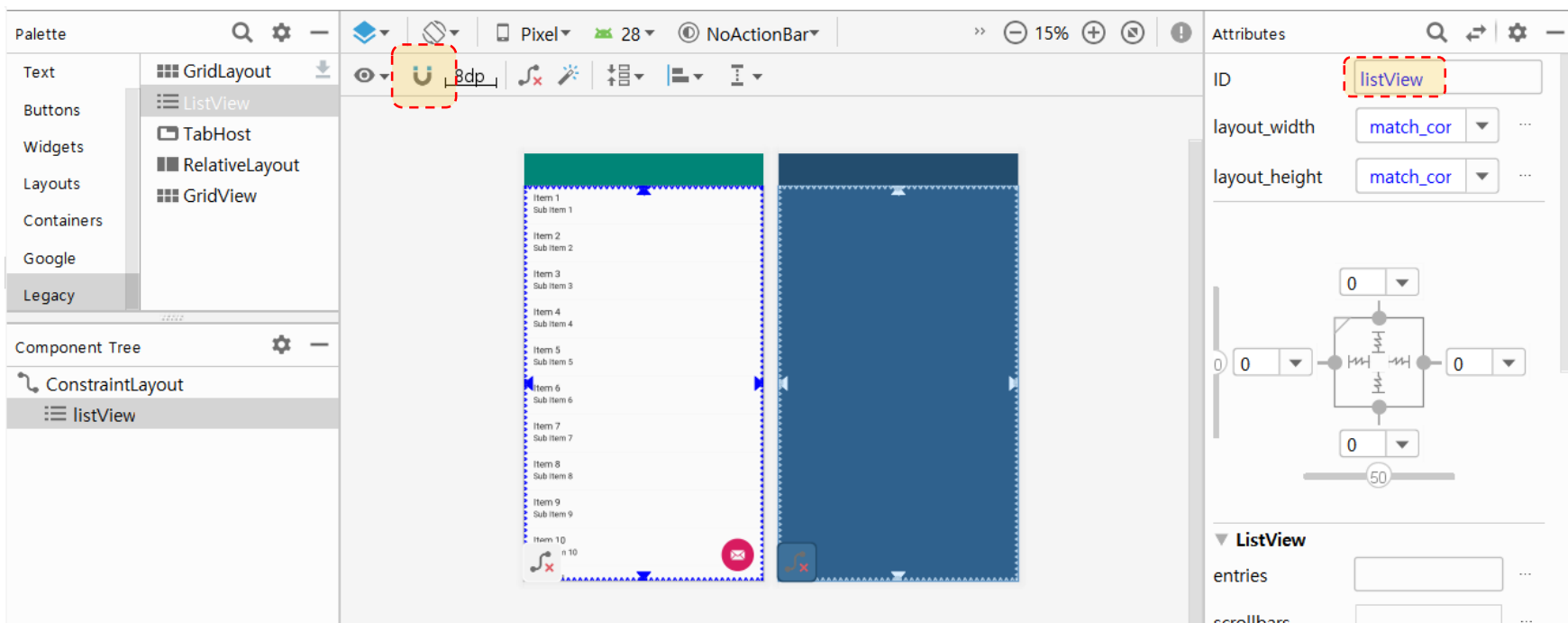
<include layout="@layout/content_main" />

<android.support.design.widget.FloatingActionButton
    android:id="@+id/fab"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="bottom|end"
    android:layout_margin="16dp"
    android:tint="@android:color/white"
    app:srcCompat="@drawable/ic_add_black_24dp" />
```

To do 리스트 앱 만들기

▶ 첫 번째 화면의 레이아웃 작성

- ▶ content_main.xml 파일을 열고 기본 배치된 텍스트 뷰(hello world)를 삭제
- ▶ AutoConnect 모드를 켜고 팔레트 창에서 Legacy 카테고리의 ListView를 선택한 후 드래그하여 화면 중앙으로 배치

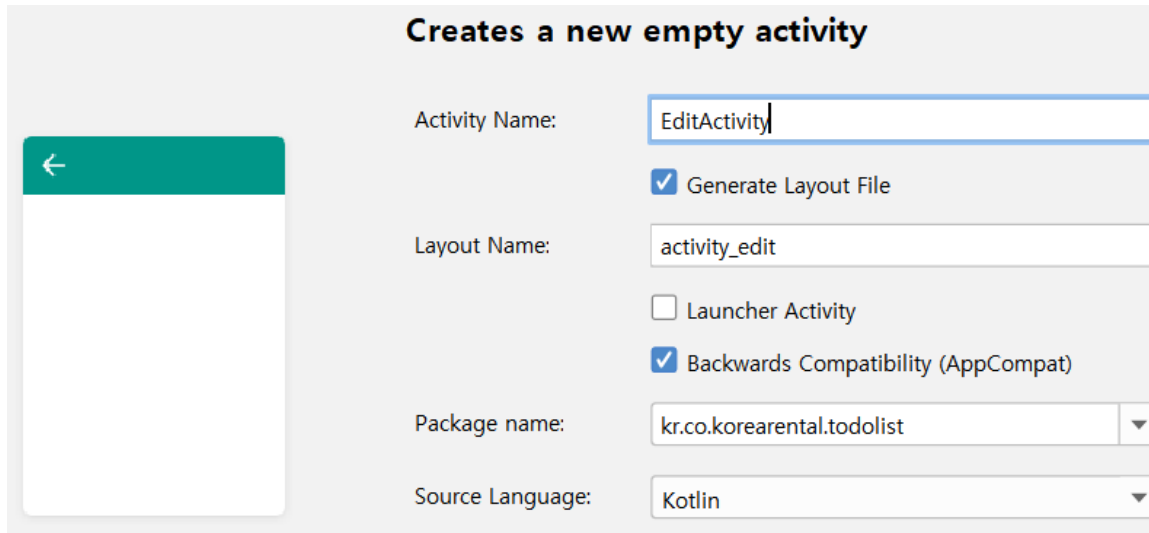


To do 리스트 앱 만들기

▶ 두 번째 액티비티 추가

▶ 두 번째 화면을 추가

- ▶ 프로젝트 창의 패키지명에서 마우스 우 클릭 또는 안드로이드 스튜디오 상단 메뉴에서 File → New → Activity → EmptyActivity를 클릭
- ▶ 액티비티 이름을 설정하는 화면이 표시되면 액티비티의 이름을 EditActivity로 설정하고 Finish를 클릭



Creates a new empty activity

Activity Name:

☒ Generate Layout File

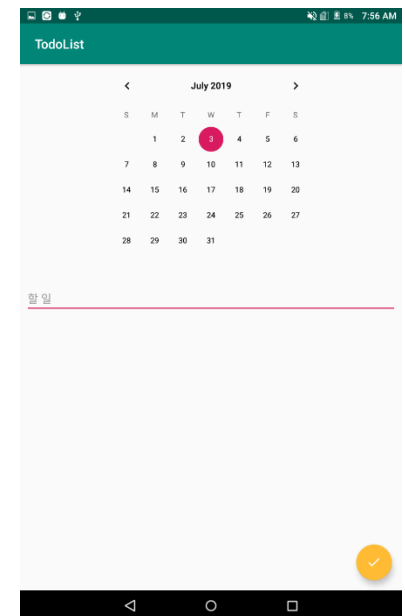
Layout Name:

☐ Launcher Activity

☒ Backwards Compatibility (AppCompat)

Package name:

Source Language:



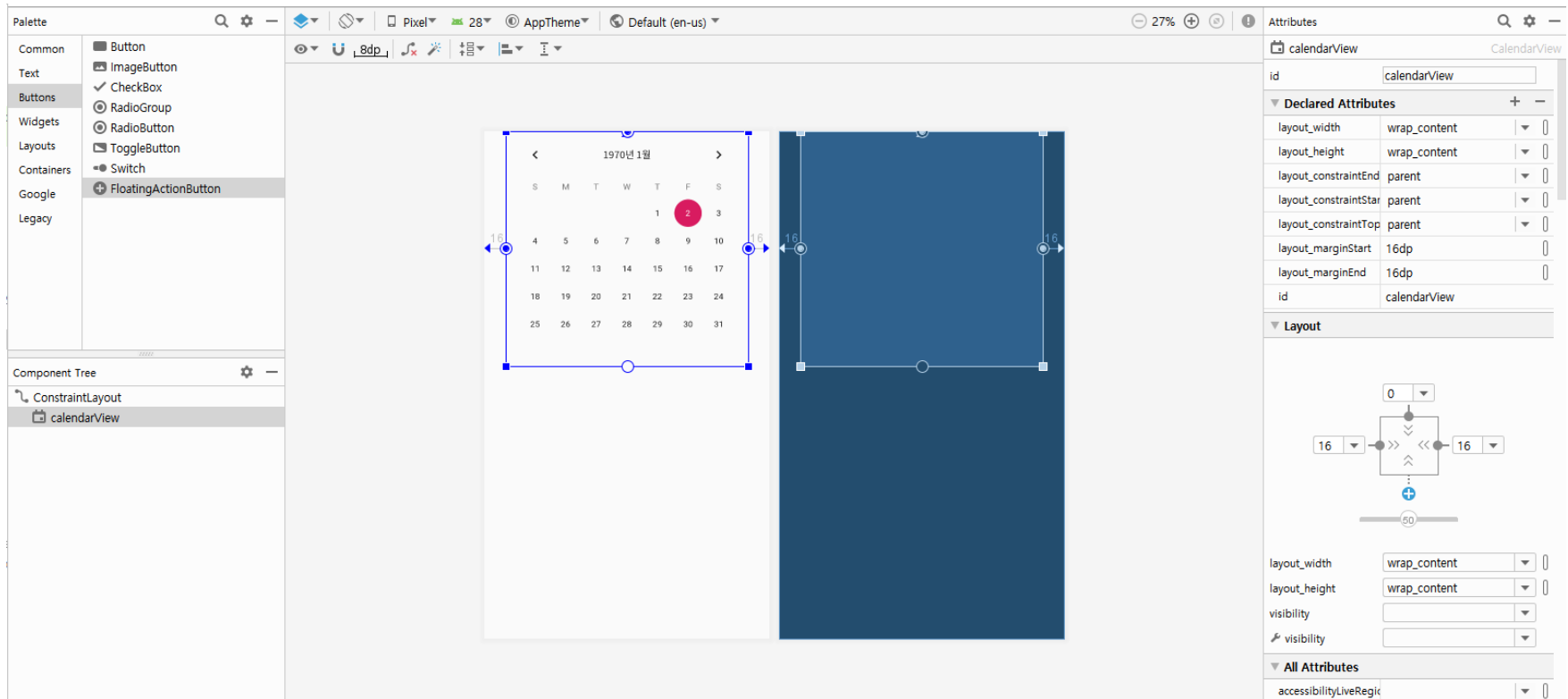
To do 리스트 앱 만들기

▶ 달력 표시용 CalendarView 배치

▶ 두 번째 액티비티가 생성되면 activity_edit.xml 파일을 열고 레이아웃을 작성

▶ 먼저 팔레트 창의 Widgets 카테고리에서 Calendar를 드래그하여 레이아웃 상단에 배치

■ CalendarView는 날짜를 선택하는 달력 모양을 제공하는 뷰



To do 리스트 앱 만들기

▶ 할 일을 입력하는 EditText 추가

- ▶ 팔레트 창의 Text 카테고리에서 Plain Text를 드래그 하여 CalendarView의 아래에 적당히 배치
- ▶ EditText와 CalendarView와의 제약을 추가
 - ▶ 배치한 EditText의 상단 제약 추가 아이콘 클릭 후 드래그하여 CalendarView의 하단 제약 아이콘까지 드래그

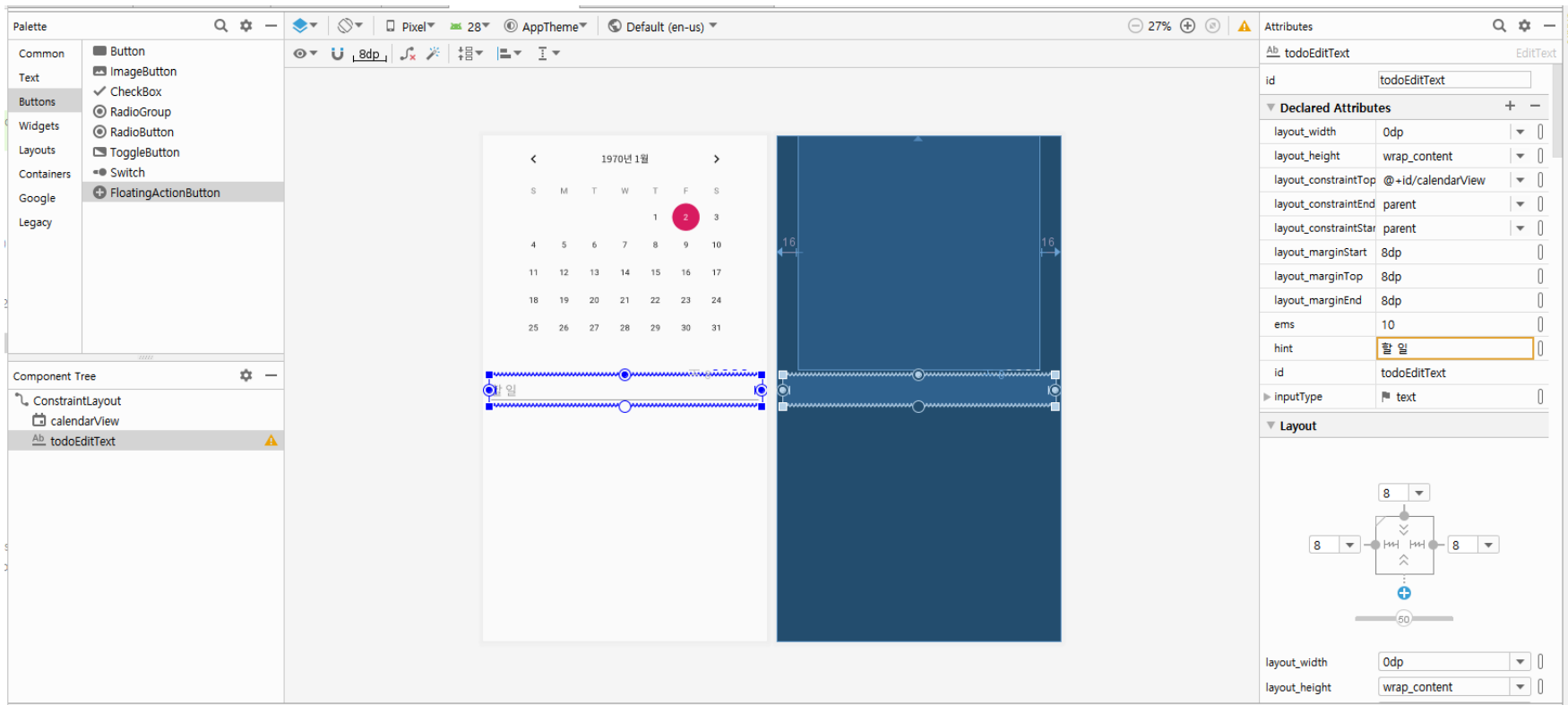


To do 리스트 앱 만들기

▶ 할 일을 입력하는 EditText 추가

▶ 속성 입력

▶ 아이디는 `todoEditText` 로 하고 추가적인 속성은 아래 화면을 참고하여 설정



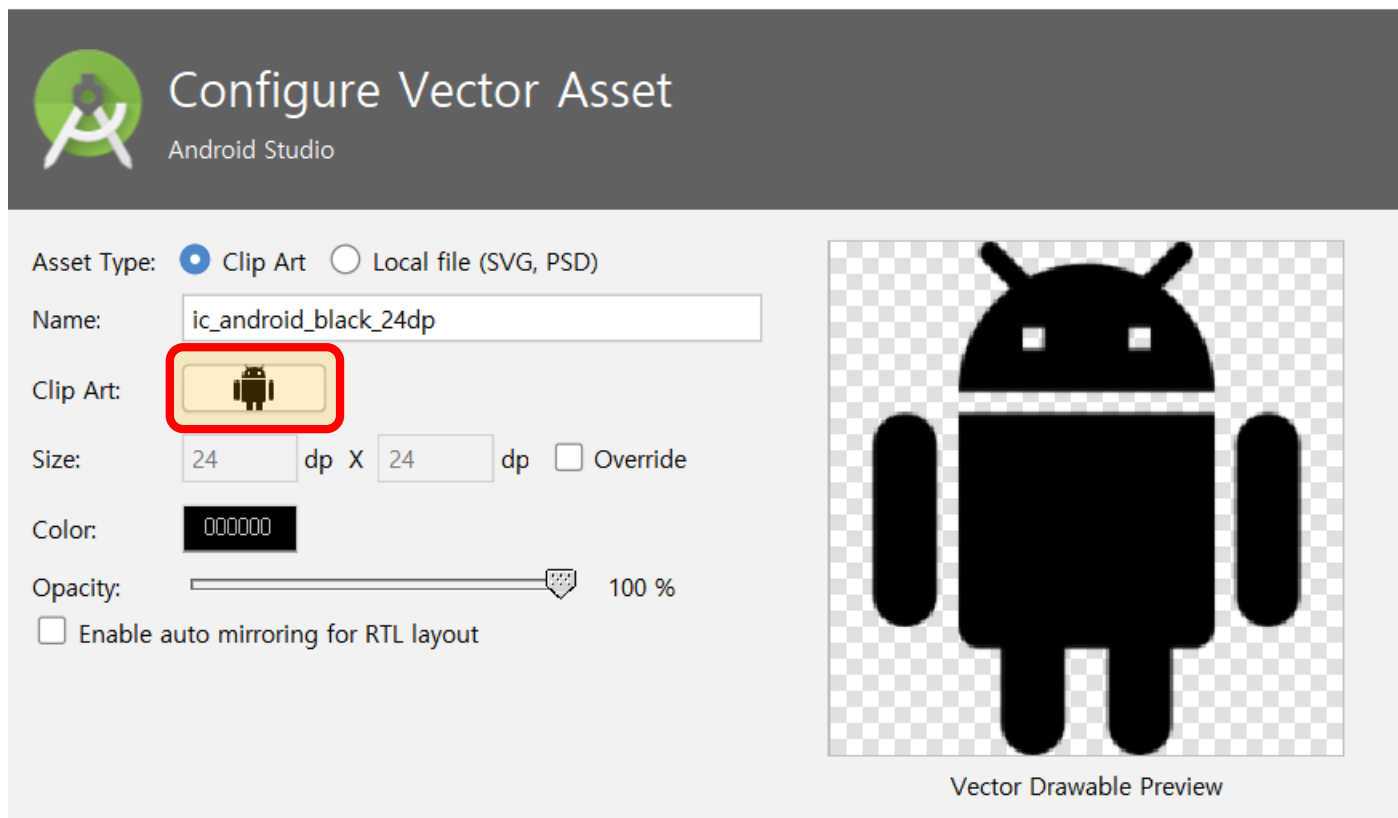
To do 리스트 앱 만들기

▶ 필요한 이미지 리소스 추가

▶ 완료 버튼과 삭제 버튼에 표시할 이미지 리소스 준비

▶ File → New → Vector Asset

▶ Clip Art 아이콘 클릭

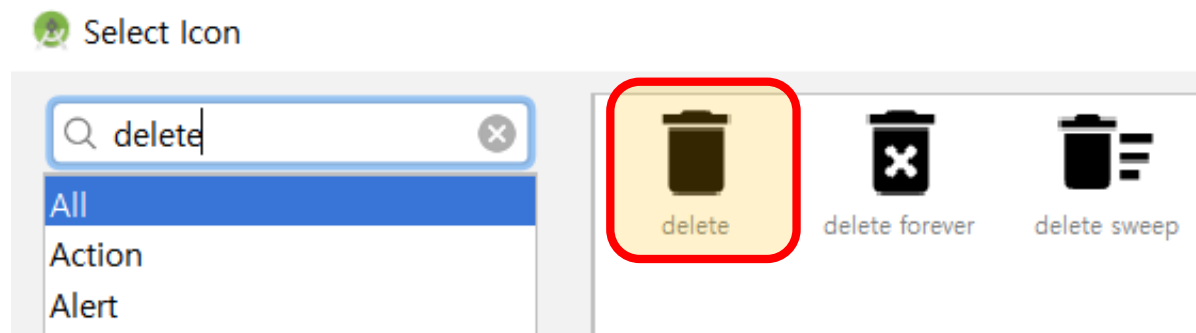
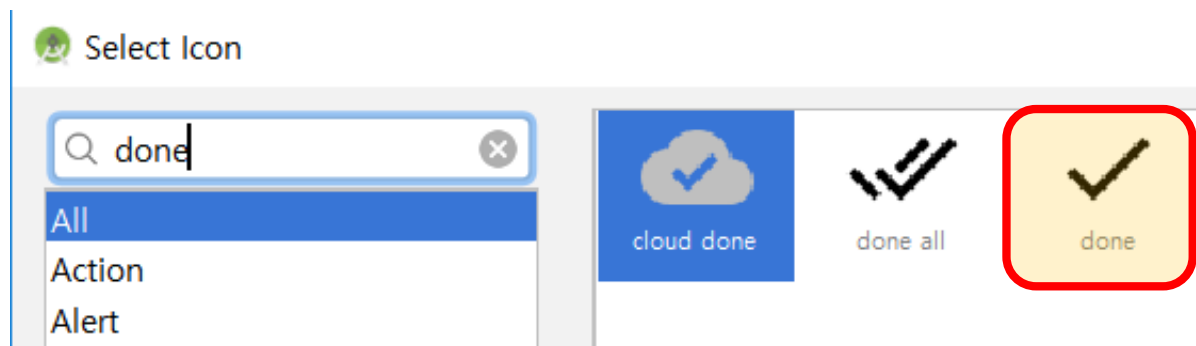


To do 리스트 앱 만들기

▶ 필요한 이미지 리소스 추가

▶ 완료 버튼과 삭제 버튼에 표시할 이미지 리소스 준비

- ▷ 검색 창에 각각 done을 검색하여 아이콘을 선택하고 OK 클릭
- ▷ 다음 화면에서 Next를 클릭하고 Finish를 클릭
- ▷ 같은 방법으로 삭제 아이콘도 생성



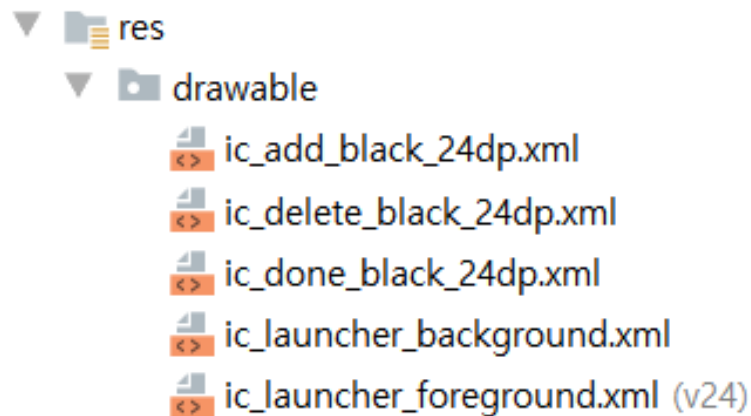
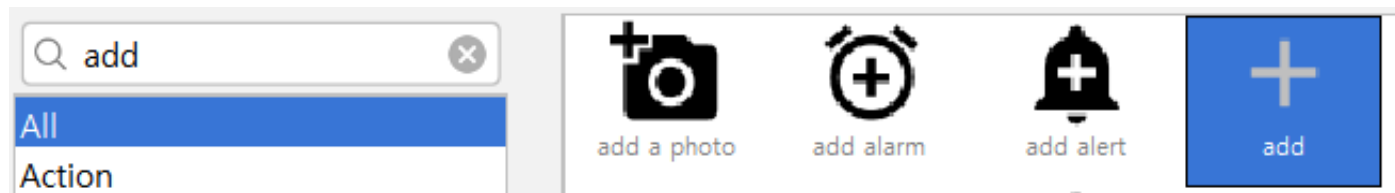
To do 리스트 앱 만들기

▶ 필요한 이미지 리소스 추가

▶ 추가로 첫 번째 화면의 FAB의 이미지도 + 아이콘으로 교체

▶ add 아이콘도 생성

▶ 프로젝트 창의 res/drawable 폴더에 리소스가 추가되면 성공

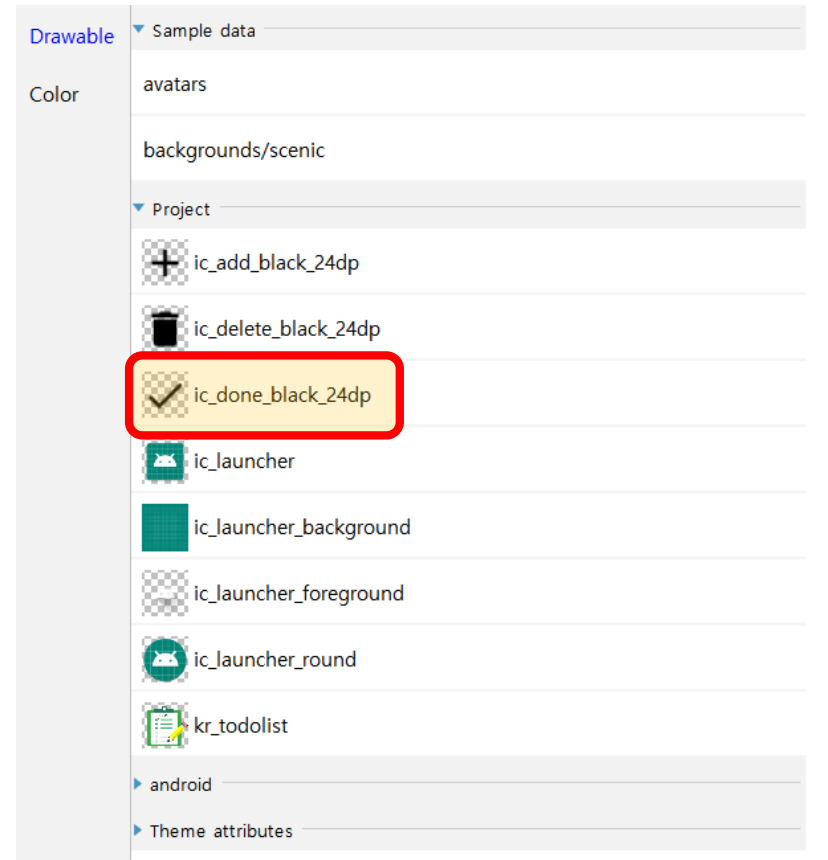
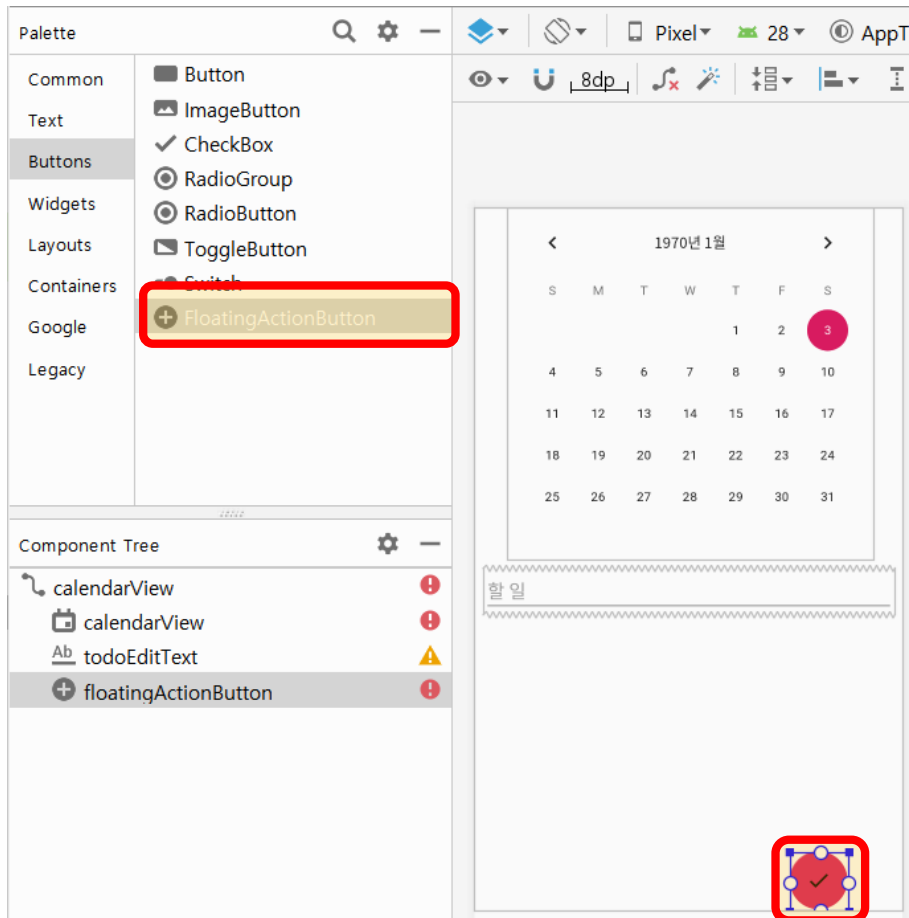


To do 리스트 앱 만들기

▶ 완료 버튼 추가

▶ 팔레트 창의 Buttons 카테고리에 있는 FloatingActionButton을 레이아웃 우측 하단에 배치

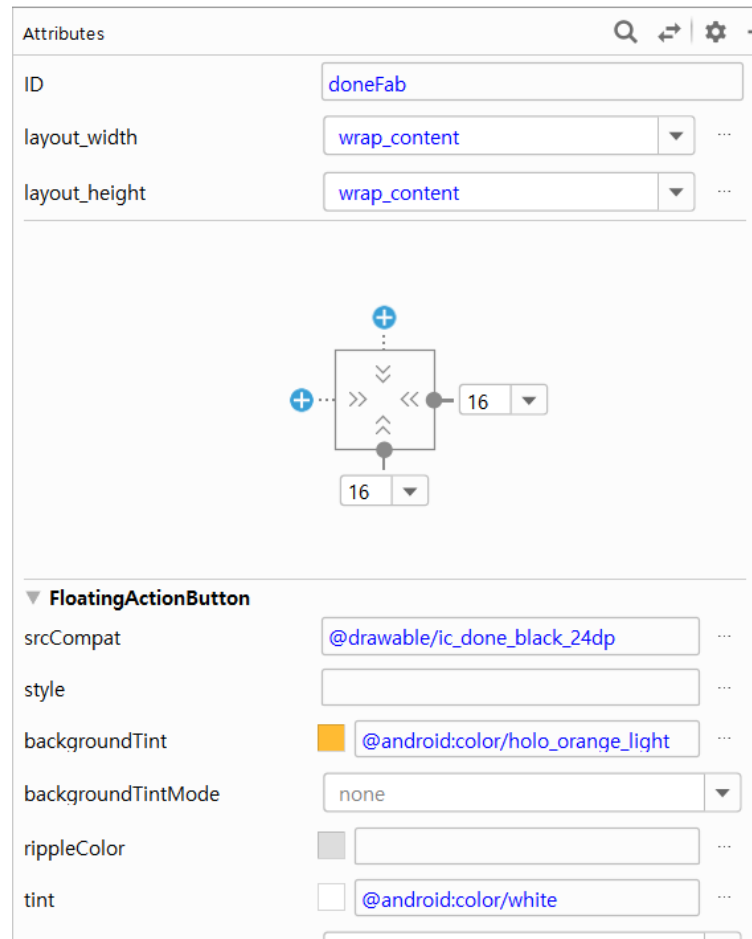
▶ Drawable의 Project를 클릭하여 ic_done_black_24dp를 선택하고 OK를 클릭



To do 리스트 앱 만들기

▶ 완료 버튼 추가

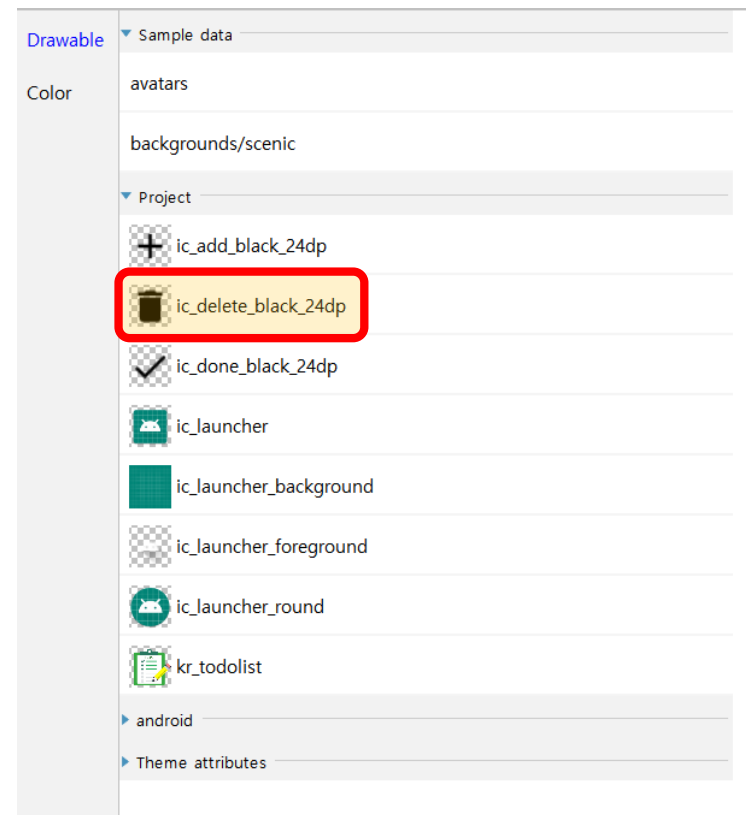
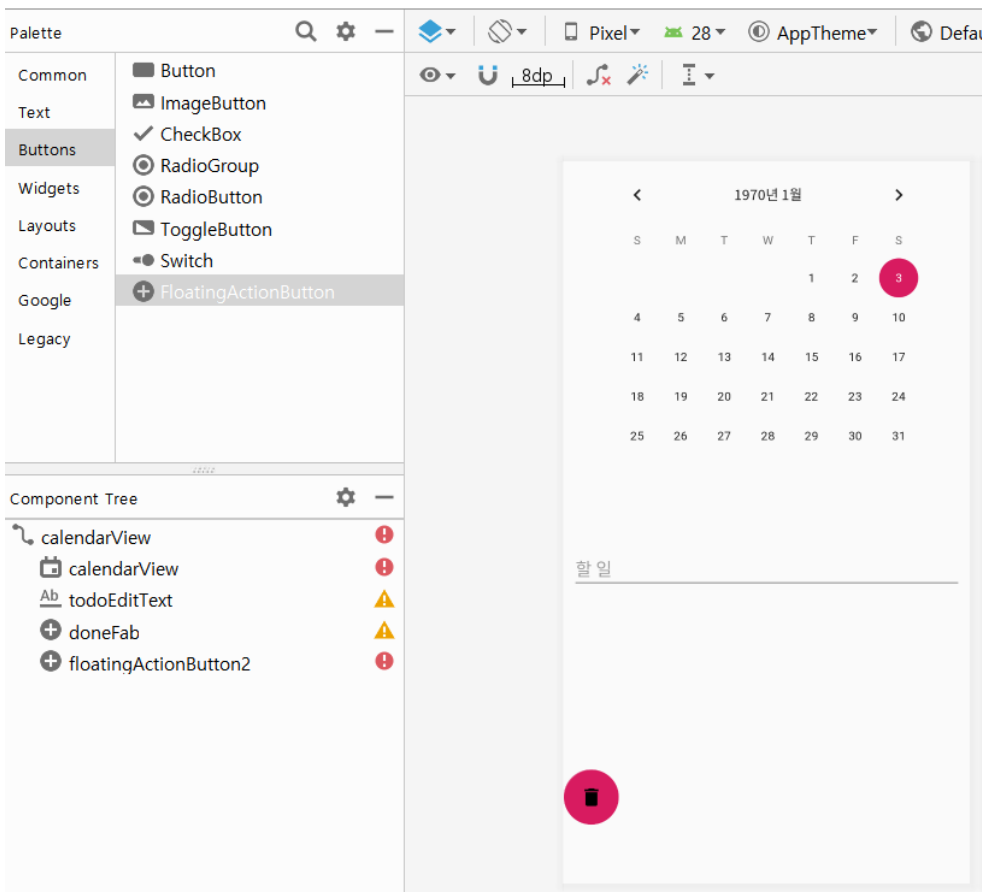
▶ 완료 버튼이 추가되면 다음과 같이 속성을 설정



To do 리스트 앱 만들기

▶ 삭제 버튼 추가

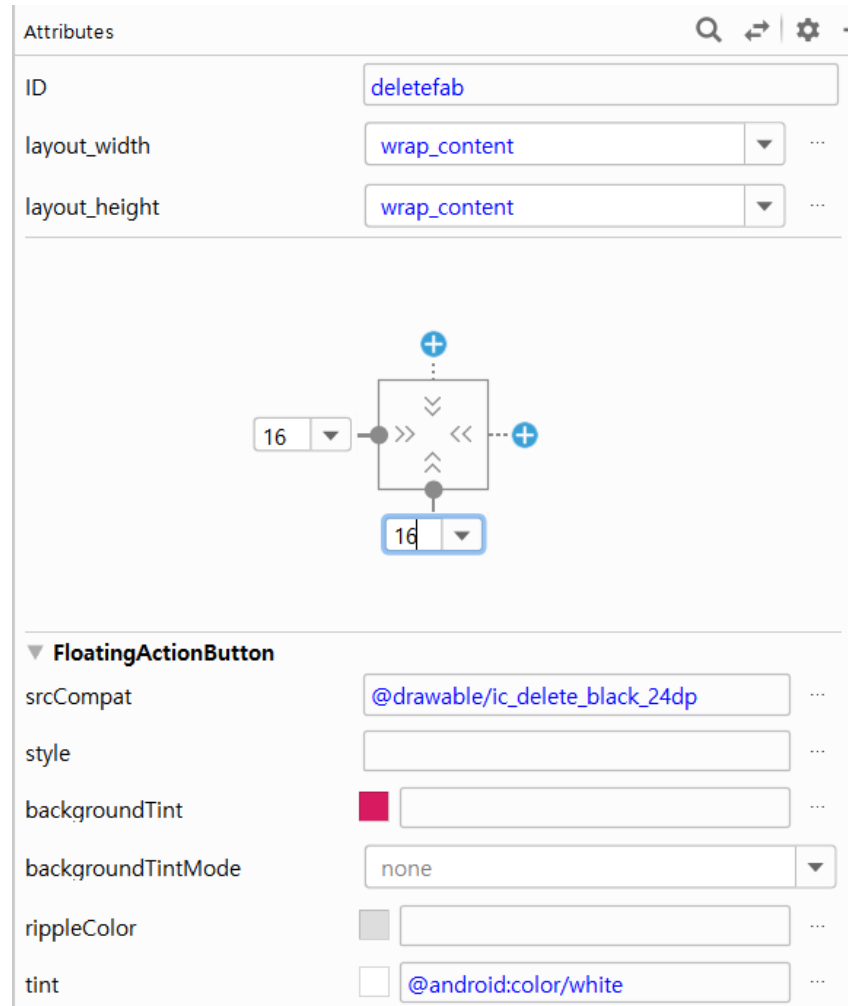
▶ 완료 버튼과 같은 방법으로 FAB를 레이아웃의 왼쪽 하단에 추가하고 다음과 같이 설정



To do 리스트 앱 만들기

▶ 삭제 버튼 추가

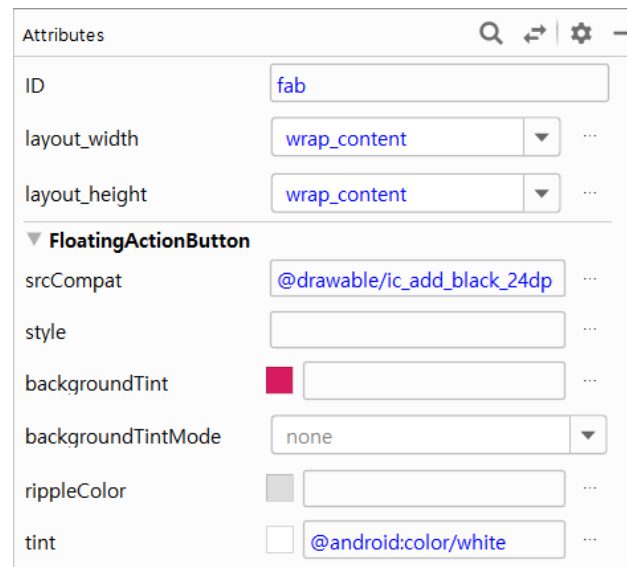
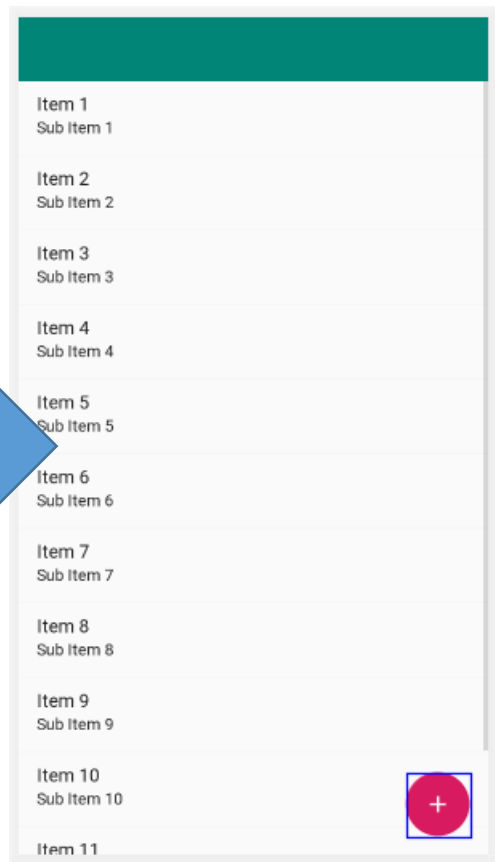
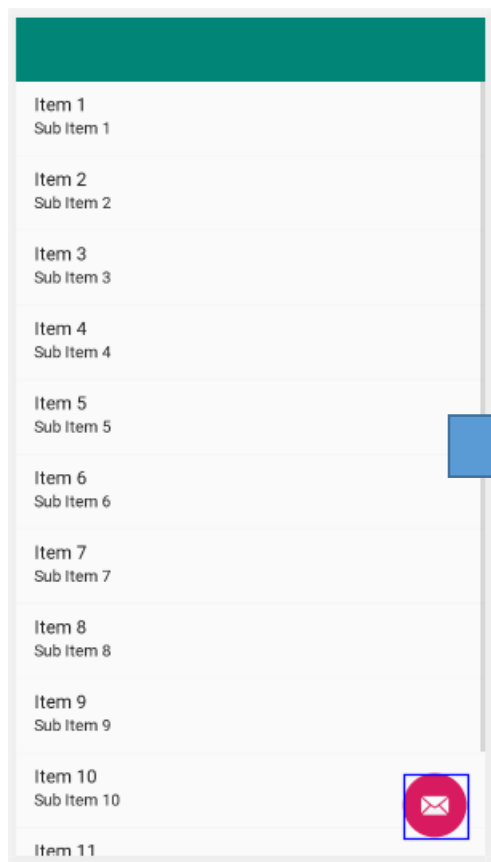
▶ 속성을 아래와 같이 설정



To do 리스트 앱 만들기

▶ 할 일의 내용을 추가하는 버튼의 이미지 변경

▶ activity_main.xml 파일을 열고 추가 버튼의 이미지와 이미지 색상을 변경



To do 리스트 앱 만들기

▶ 데이터베이스의 필요성

- ▶ 앞서 진행한 비만도 계산기를 만들 때 SharedPreferences로 간단한 데이터를 저장
- ▶ 연락처, 일정, 문자 같이 복잡하고 크기가 큰 데이터는 데이터베이스를 활용해야 함
- ▶ 안드로이드에서는 SQLite 데이터베이스를 지원
- ▶ SQLite 데이터베이스는 강력하지만 코드양이 많아 사용하기 어렵기 때문에 프로젝트에서는 SQLite 대신 Realm 데이터베이스를 사용

▶ Realm DB 개요

- ▶ Realm은 짧은 코드로 데이터베이스를 작성할 수 있어 쉬움
- ▶ 쿼리문을 사용해 테이블의 컬럼에 값을 저장하는 SQLite와 달리, 데이터를 객체의 형태로 저장
 - ▶ SQLite는 SQL 문법을 어느 정도는 알고 있어야 하는 반면 Realm은 SQL 문법에 익숙하지 않아도 사용할 수 있음



To do 리스트 앱 만들기

▶ 데이터베이스(Database)

- ▶ 데이터베이스란 데이터를 저장하는 저장소
- ▶ 안드로이드에서는 앱 별로 격리된 데이터베이스를 가질 수 있음
- ▶ 각각의 앱마다 제공되는 데이터베이스는 고유한 **격리공간**에 있어서 외부 앱에서는 접근할 수 없음
 - ▷ 프로바이더를 통하여 앱이 가지고 있는 데이터베이스를 외부에 공개

To do 리스트 앱 만들기

▶ 데이터베이스의 예

▶ 다음은 강아지 정보를 가지고 있는 테이블

▷ 테이블은 데이터를 담는 단위

▶ 스프레드시트도 일종의 데이터베이스인데 셀의 값은 데이터이고 시트는 테이블에 해당

▶ Doglist라는 이름의 데이터베이스에는 Dog라는 테이블이 있고 id, 이름, 나이 열(필드)이 있음

▷ 한 줄씩 열별로 데이터가 저장됨.

▶ 이렇게 데이터베이스는 행렬 구조로 데이터를 저장

	A	B	C	D
1	id	이름	나이	
2	1	멍멍이	3	
3	2	바둑이	2	
4	3	진돗개	4	

To do 리스트 앱 만들기

▶ Realm 데이터베이스 사용 준비

▶ 프로젝트 수준의 build.gradle 파일을 열고 dependencies 항목에 다음과 같이 플러그인을 추가

▷ classpath "io.realm:realm-gradle-plugin:5.3.0"

```
dependencies {  
    classpath 'com.android.tools.build:gradle:3.3.2'  
    classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:$kotlin_version"  
    // NOTE: Do not place your application dependencies here; they belong  
    // in the individual module build.gradle files  
    classpath "io.realm:realm-gradle-plugin:5.3.0"  
}
```

▶ 다음으로 모듈 수준의 build.gradle 파일을 열고 상단에 다음 두 가지 플러그인을 추가

▷ 가능하면 아래 코드 두 줄의 위치와 순서를 우측 화면과 동일하게 추가

■ apply plugin: 'kotlin-kapt'

apply plugin: 'kotlin-android-extensions'

■ apply plugin: 'realm-android'

apply plugin: 'kotlin-kapt'

apply plugin: 'realm-android'

To do 리스트 앱 만들기

▶ Realm 객체로 만드는 방법(예시)

▶ Realm 사용 방법을 간단하게 알아보기

▷ 먼저 위에서 실행했던 Dog 테이블을 클래스로 작성

▷ 이와 같은 모양의 클래스를 모델 클래스라고 부름

```
class Dog (val id : Long, var name : String = "", var age:Int = 0){  
}
```

▶ Realm에서 테이블로 사용하려면 모델 클래스 앞에 open을 붙이고 RealmObject 클래스를 상속하면 됨

```
open class Dog (val id : Long, var name : String = "", var age:Int = 0) :  
    RealmObject(){  
}
```

To do 리스트 앱 만들기

▶ 데이터베이스 설계

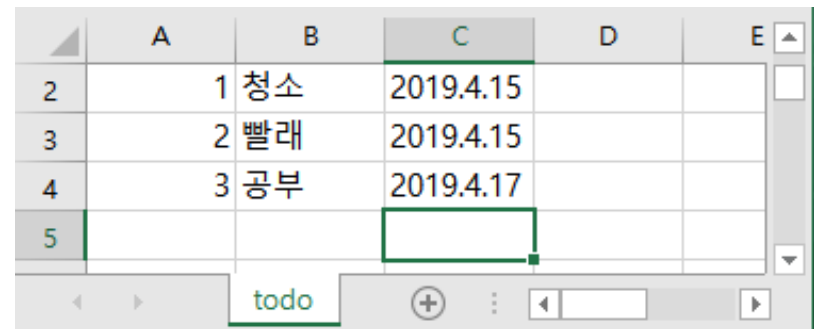
▶ 할 일 정보를 저장할 때 어떤 항목을 어떤 데이터 타입으로 저장할지 설계

▶ 다음은 할 일 정보를 스프레드 시트를 사용해 정리

▷ 테이블 명 : todo

▷ 열 :

- id : Long, 자동 증가, 고유한 값
- title : String, 할 일 내용
- date : Long, 시간



	A	B	C	D	E
2	1	청소	2019.4.15		
3	2	빨래	2019.4.15		
4	3	공부	2019.4.17		
5					

▶ 일반적으로 데이터베이스에서는 각 행마다 고유 ID 를 가짐

▷ 데이터베이스에서는 데이터를 식별하는 유일한 키 값을 기본키라고 함

To do 리스트 앱 만들기

▶ Realm 모델 클래스 작성

- ▶ 먼저 Realm에서 위와 같은 테이블 정보를 다룰 Todo 모델 클래스를 새로 작성
- ▶ 프로젝트 창에서 마우스 우클릭 또는 안드로이드 스튜디오 상단 메뉴에서
File → New → Kotlin → File/Class 를 클릭하고 Todo 라는 이름으로 새로운 클래스를 생성
- ▶ 생성한 Todo.kt 파일을 다음과 같이 작성

```
import io.realm.RealmObject
import io.realm.annotations.PrimaryKey

open class Todo(
    @PrimaryKey var id: Long = 0,
    var title: String = "",
    var date: Long = 0
) : RealmObject() {
}
```

- ▶ 코틀린에서는 Realm에서 사용하는 클래스에 open 키워드를 추가
- ▶ RealmObject 클래스를 상속받아 Realm 데이터베이스에서 다룰 수 있음
- ▶ Id는 유일한 값이 되어야 하기 때문에 기본키 제약을 주석으로 추가
- ▶ 기본키 제약은 Realm에서 제공하는 주석이며, 주석이 부여된 속성 값은 중복을 허용하지 않음

To do 리스트 앱 만들기

▶ Realm 초기화

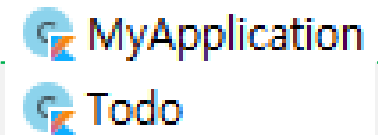
▶ 앱이 실행될 때 제일 먼저 Realm을 초기화해서 모든 액티비티가 사용하도록 할 수 있음

▶ 앱을 실행하면 가장 먼저 실행되는 애플리케이션 객체를 상속하여 Realm을 초기화하여야 함

- File → New → Kotlin File/Class 를 클릭하고 MyApplication 이라는 이름으로 클래스를 생성
- 생성한 MyApplication.kt 파일을 다음과 같이 작성

```
import android.app.Application
import io.realm.Realm

class MyApplication : Application() {
    override fun onCreate() {
        super.onCreate()
        Realm.init(this)    //Realm을 사용하기 위하여 초기화
    }
}
```



- Application 클래스를 상속받는 MyApplication 클래스를 선언
- onCreate() 메서드를 오버라이드
 - 이 메서드는 액티비티가 생성되기 전에 호출됨
- Realm.init() 메서드를 사용하여 초기화

To do 리스트 앱 만들기

▶ Realm 초기화

- ▶ 안드로이드 ApplicationClass 는 모든 컴포넌트(component)에서나 공유할 수 있는 전역 클래스
- ▶ 컴포넌트들 사이에서 공동으로 관리할 데이터가 있다면 ApplicationClass를 상속받아 만든 클래스를 사용
- ▶ 앱에서 사용하는 전체 액티비티 중에서 공통적으로 사용하는 객체를 초기화할 경우 매니페스트에 해당 클래스를 등록
 - ▶ ApplicationClass를 상속 받은 클래스를 생성 후 매니페스트에서 등록
 - application 태그 안에 name 속성을 추가
 - android:name=".MainActivity"

```
<application
    android:name = ".MyApplication"
    android:allowBackup="true"
    android:icon="@mipmap/kr_todolist"
    android:label="ToDoList"
    android:roundIcon="@mipmap/kr_todolist"
    android:supportRtl="true"
    android:theme="@style/AppTheme">
    <activity android:name=".EditActivity">
    </activity>
```

To do 리스트 앱 만들기

▶ 액티비티에서 Realm 인스턴스 객체 얻기

▶ 두 번째 화면인 EditActivity.kt 파일을 열고 Realm을 사용하기 위하여 인스턴스를 가져옴

▷ MyApplication 클래스에서 Realm을 초기화했다면 액티비티에서는 Realm.

getDefaultInstance() 메서드를 이용해 Realm 객체의 인스턴스를 얻을 수 있음

▶ 액티비티가 소멸되는 생명주기인 onDestroy()에서 렘 데이터베이스 인스턴스를 해제

```
class EditActivity : AppCompatActivity() {  
  
    val realm = Realm.getDefaultInstance()  
  
    override fun onDestroy() {  
        super.onDestroy()  
        realm.close()  
    }  
}
```

To do 리스트 앱 만들기

▶ 날짜를 다루는 Calendar 클래스 사용 방법(예시)

▶ Calendar객체를 getInstance() 메서드로 생성

▶ 객체를 생성한 오늘 날짜로 초기화됨

▶ set() 메서드에서 변경할 필드와 값을 지정하여 년, 월, 일 등을 지정하여 변경할 수 있음.
지정하는 필드는 Calendar 클래스에 상수로 정의되어 있음

▶ 사용 방법

```
//import java.util.*
//오늘 날짜로 캘린더 객체 생성
val calendar: Calendar = Calendar.getInstance()
//특정 날짜로 설정
calendar.set(Calendar.YEAR, year)           //set(field:Int, value:Int)
calendar.set(Calendar.MONTH, month)         //field : 변경할 필드, 변경할 값
calendar.set(Calendar.DAY_OF_MONTH, dayOfMonth)
//날짜를 Long형으로 반환
val time : Long = calendar.timeInMillis
```

▶ 데이터베이스에는 시간이나 날짜를 Long 형으로 저장

▶ Calendar 객체는 Long형으로 반환하는 getTimeInMillis() 메서드를 제공

▶ 코틀린에서는 timeInMillis 프로퍼티로 사용할 수 있음

To do 리스트 앱 만들기

▶ 프로퍼티(property) 사용 방법(예시)

▶ 자바나 C++의 필드(멤버 변수)와 유사하지만 필드와 Getter, Setter가 합쳐진 개념

▶ 반드시 선언과 동시에 초기화 해야함

```
class Person{
    var name : String = "kotlin"
    var age : Int = 0
    get(){
        return field
    }
    set(value){
        field = if(value >= 0) value else 0
    }
    var address : String = "Seoul"
    private set                //클래스 내부에서만 setter 사용 가능
}

fun main(args:Array<String>){
    val person = Person()
    person.name = "android"
    println(person.name)      //android
    person.age = -100
    println(person.age)       //0
    println(person.address)    //seoul
    person.address = "daegu"   //오류
}
```

To do 리스트 앱 만들기

▶ 할 일 추가

- ▶ EditActivity 파일에 할 일을 추가하는 insertTodo() 메서드를 작성
- ▶ Realm에서 데이터를 추가, 삭제, 업데이트할 때는 beginTransaction() 메서드로 트랜잭션을 시작
 - ▷ 트랜잭션이란 데이터베이스의 작업 단위
 - ▷ 데이터베이스에 추가, 삭제, 업데이트를 하려면 항상 트랜잭션을 시작하고 닫아야 함.
- ▶ beginTransaction()메서드와 commitTransaction() 메서드 사이에 작성한 코드들은 전체가 하나의 작업(트랜잭션)이기 때문에 도중에 에러가 나면 모두 취소됨

```
private fun insertTodo() {  
    realm.beginTransaction()  
    val todo : Todo = realm.createObject<Todo>(nextId())  
    todo.title = todoEditText.text.toString()  
    todo.date = calendar.timeInMillis  
    realm.commitTransaction()  
  
    alert( message: "내용이 추가되었습니다.") { this: AlertDialog.Builder<DialogInterface>  
        | yesButton { finish() }  
    }.show()  
}
```

To do 리스트 앱 만들기

▶ 할 일 추가

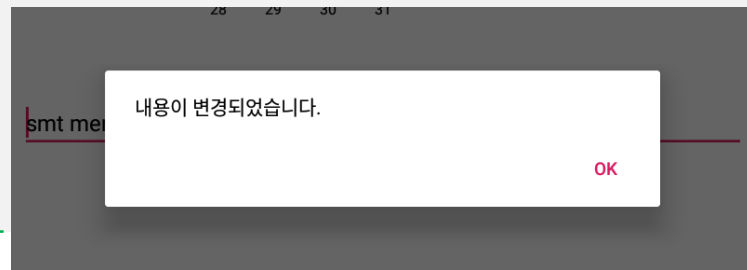
▶ EditActivity.kt에 아래 코드 추가

```
//import java.util.*
//import io.realm.kotlin.createObject
val calendar: Calendar = Calendar.getInstance()//날짜를 다루는 캘린더 객체

private fun insertTodo() {
    realm.beginTransaction()    //트랜잭션 시작
    val todo = realm.createObject<Todo>(nextId())    //새 객체 생성
    todo.title = todoEditText.text.toString()    //값 설정
    todo.date = calendar.timeInMillis
    realm.commitTransaction()    //트랜잭션 종료

    alert("내용이 추가되었습니다.") { yesButton { finish() } }.show()//다이얼로그 표시
}

private fun nextId(): Int {    //다음 id를 반환
    val maxId = realm.where<Todo>().max("id")
    if (maxId != null) {
        return maxId.toInt() + 1
    }
    return 0
}
```



To do 리스트 앱 만들기

▶ 할 일 추가(설명)

▶ createObject() 메서드로 새로운 Realm 객체를 생성

```
//새로운 T 타입의 Realm 객체를 생성
```

```
createObject<T : RealmModel>(primaryKeyValue : Any?)//primaryKeyValue : 기본키 지정
```

▶ Realm은 기본 키 자동 증가 기능을 지원하지 않기 때문에 가장 큰 id 값을 얻고 1을 더한 값을 반환하는 메서드를 추가로 작성

▶ 객체 생성 시 id 값을 입력할 때 사용

▶ Todo 테이블의 모든 값을 얻으려면 where<Todo>() 메서드를 사용

▶ 이 메서드는 RealmQuery 객체를 반환하고 다음에 이어지는 조건을 수행

▶ 여기서 max()메서드를 조건으로 달았는데 이 메서드는 현재 id 중 가장 큰 값을 얻을 때 사용

```
//fieldName 열 값 중에 가장 큰 값을 Number형으로 반환
```

```
max(fieldName : String) //fieldname 검색 범위가 되는 열 이름
```

▶ 객체를 생성했다면 할 일과 시간을 설정

▶ 할 일이 추가되면 다이얼로그를 표시. 다이얼로그 확인 버튼을 누르면 finish() 메서드를 호출하여 현재 액티비티를 종료

To do 리스트 앱 만들기

▶ 할 일 수정

▶ EditActivity.kt에 할 일을 수정하는 updateTodo() 메서드를 다음과 같이 작성

```
//import io.realm.kotlin.where
private fun updateTodo(id: Long) {
    realm.beginTransaction() //트랜잭션 시작
    val todo = realm.where<Todo>().equalTo("id", id).findFirst()!!//!!!:todo는 이후부터 null이 아님
    todo.title = todoEditText.text.toString()
    todo.date = calendar.timeInMillis
    realm.commitTransaction() //트랜잭션 종료 반영

    alert( " 내용이 변경되었습니다. " ) { //다이얼로그 표시
        yesButton { finish() }
    }.show()
}
```

▶ updateTodo() 메서드는 id를 인자로 받음

▶ Realm 객체의 where<T>() 메서드가 반환하는 T 타입 객체로부터 데이터를 얻음

▶ equalTo() 메서드로 조건을 설정. “id” 컬럼에 id 값이 있다면 findFirst() 메서드로 첫 번째 데이터를 반환

▶ 나머지 코드는 할 일 추가와 동일

To do 리스트 앱 만들기

▶ 할 일 삭제

- ▶ EditActivity.kt에 할 일을 삭제하는 deleteTodo() 메서드를 다음과 같이 작성
- ▶ 할 일을 수정하는 메서드와 거의 흡사
- ▶ 메서드로 전달받은 id로 삭제할 객체를 검색하고 찾았다면 deleteFromRealm() 메서드로 삭제

```
private fun deleteTodo(id: Long) {  
    realm.beginTransaction()  
    val todo = realm.where<Todo>().equalTo("id", id).findFirst()!!  
    todo.deleteFromRealm() // deleteFromRealm 메서드로 삭제  
    realm.commitTransaction()  
  
    alert("내용이 삭제되었습니다.") {  
        yesButton { finish() }  
    }.show()  
}
```

To do 리스트 앱 만들기

▶ 추가/수정 분기 처리(설명)

▶ 첫번째 화면에서 할일 목록의 id 받아오기

- ▶ 첫 번째 화면에서 인텐트를 이용해 id 값을 전달받았다면 데이터베이스의 id는 0부터 시작하므로 0 이상의 값이 넘어오게 됨

```
//인텐트에서 데이터를 추출하여 반환  
getLongExtra(name:String, defaultValue:Long)
```

- ▶ name : 아이템을 가리키는 key
- ▶ defaultValue : 반환되는 값이 없을 경우 기본값을 설정

To do 리스트 앱 만들기

▶ 추가/수정 분기 처리

▶ 두 번째 화면에서는 할 일을 추가하거나 수정

- ▶ 추가 모드와 수정 모드를 구분하는 규칙을 정한 후 분리하여 처리
- ▶ id를 -1로 초기화하고 수정 시에는 id를 인텐트에 포함하여 받아 옴
- ▶ id가 -1이면 추가 모드이고, 아니면 수정 모드가 되도록 함
- ▶ 추가 모드와 수정 모드를 분기하여 처리하는 코드를 다음과 같이 추가

```
override fun onCreate(savedInstanceState: Bundle?) {  
    // 업데이트 조건  
    val id = intent.getLongExtra("id",-1L)//1:Int, 1L:Long, 1.0:Double, 1.0f:Float  
    if (id == -1L) {    //초기화 값(기본값)인 -1이 그대로 넘어오면 추가모드  
        insertMode()  
    } else {           //중간에 변경되었으면 수정 모드  
        updateMode(id)  
    }  
}
```

To do 리스트 앱 만들기

▶ 추가/수정 분기 처리

- ▶ CalendarView에서 날짜를 선택하면 해당 날짜에 대한 처리는 setOnDateChangeListener() 메서드로 구현
- ▶ 변경된 년, 월, 일이 year, month, dayOfMonth로 넘어오므로 Calendar 객체에 년, 월, 일을 설정해주면 데이터베이스에 추가, 수정 시 설정한 날짜가 반영됨

```
override fun onCreate(savedInstanceState: Bundle?) {  
    super.onCreate(savedInstanceState)  
    setContentView(R.layout.activity_edit)  
  
    // 캘린더뷰의 날짜를 선택했을 때 Calendar 객체에 설정  
    // 선택할 때마다 날짜정보를 넘겨 받음  
    calendarView.setOnDateChangeListener { view, year, month, dayOfMonth ->  
        calendar.set(Calendar.YEAR, year)  
        calendar.set(Calendar.MONTH, month)  
        calendar.set(Calendar.DAY_OF_MONTH, dayOfMonth)  
    }  
}
```

To do 리스트 앱 만들기

▶ 추가/수정 분기 처리

- ▶ 추가 모드일 때는 삭제 버튼을 숨김
- ▶ 뷰를 보이거나 안 보이게 하려면 `setVisibility()` 메서드를 사용
- ▶ 코틀린에서는 `visibility` 프로퍼티로 사용할 수 있음
- ▶ `Visibility` 프로퍼티에는 다음과 같은 속성을 지정할 수 있음
 - ▷ `VISIBLE` : 보이게 함
 - ▷ `INVISIBLE` : 영역은 차지하지만 보이지 않음
 - ▷ `GONE` : 완전히 보이지 않음

```
class EditActivity : AppCompatActivity() {  
    // 추가 모드 초기화  
    private fun insertMode() {  
        deleteFab.visibility = View.GONE           // 삭제 버튼을 감추기  
        //오류 발생시 deleteFab.hide()  
        doneFab.setOnClickListener {                // 완료 버튼을 클릭하면 추가  
            insertTodo()  
        }  
    }  
}
```

To do 리스트 앱 만들기

▶ 추가/수정 분기 처리

▶ 추가 모드에서는 완료 버튼을 누르면 할 일을 추가

```
class EditActivity : AppCompatActivity() {  
    // 수정 모드 초기화  
    private fun updateMode(id: Long) {  
        // id에 해당하는 객체를 화면에 표시  
        val todo = realm.where<Todo>().equalTo("id", id).findFirst()!!  
        todoEditText.setText(todo.title)  
        calendarView.date = todo.date  
  
        // 완료 버튼을 클릭하면 수정  
        doneFab.setOnClickListener {  
            updateTodo(id)  
        }  
  
        // 삭제 버튼을 클릭하면 삭제  
        deleteFab.setOnClickListener {  
            deleteTodo(id)  
        }  
    }  
}
```

To do 리스트 앱 만들기

▶ 추가/수정 분기 처리

- ▶ MainActivity.kt 파일을 열어 FAB를 클릭했을 때 EditActivity 액티비티를 시작하도록 수정
- ▶ 앱을 실행하여 새 할 일을 추가
- ▶ 추가되었다는 다이얼로그가 표시되고 첫 화면으로 돌아오면 성공임
- ▶ 아직 첫 화면의 리스트 뷰를 작성하지 않기 때문에 목록이 표시되지는 않음

```
override fun onCreate(savedInstanceState: Bundle?) {  
    // 새 할 일 추가  
    fab.setOnClickListener {  
        startActivity<EditActivity>()  
    }  
}
```


To do 리스트 앱 만들기

▶ 리스트 뷰와 데이터베이스 연동

- ▶ 첫 번째 화면에 할 일 목록을 표시하는 리스트 뷰 구현

- ▶ Realm은 리스트 뷰에 표시할 데이터를 정리하는 **RealmBaseAdapter** 어댑터를 제공

▶ 리스트 뷰의 이해

- ▶ 스크롤 뷰를 적은 양의 아이템을 스크롤하는 데 사용했다면 많은 양의 반복되는 아이템을 표시할 때는 리스트 뷰를 사용

▷ 스크롤 뷰

- 적은 양의 아이템을 스크롤할 때 간단히 사용함
- 한 번에 모든 아이템을 메모리에 로드하여 상황에 따른 많은 메모리가 요구

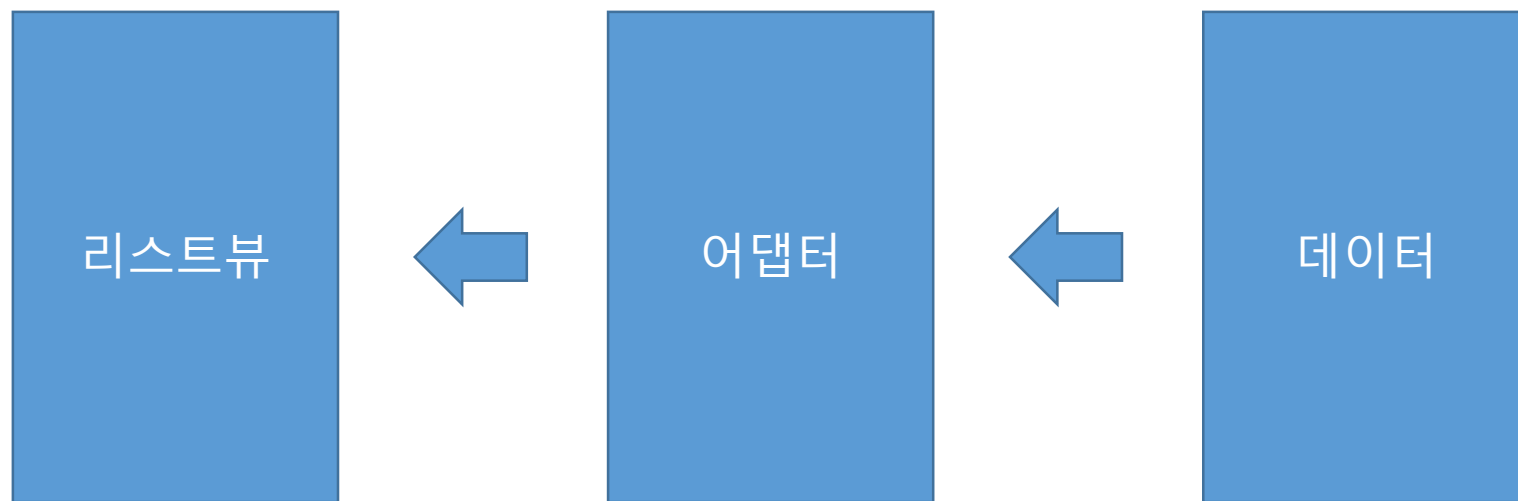
▷ 리스트 뷰

- 많은 양의 반복되는 아이템을 표시할 때 사용함
- 뷰를 재사용하므로 적은 메모리를 사용하고 화면에 보이는 것만 동적으로 로딩

To do 리스트 앱 만들기

▶ 리스트 뷰와 데이터베이스 연동

- ▶ 리스트 뷰를 사용하려면 데이터와 데이터를 표현하는 어댑터를 작성해야 함
- ▶ 어댑터란 데이터를 리스트 뷰에 어떻게 표시할지 정의하는 객체
- ▶ 어댑터를 작성하기에 따라서 리스트 뷰의 성능에도 큰 영향을 미치기 때문에 어댑터 작성은 중요



To do 리스트 앱 만들기

▶ 데이터 준비

- ▶ MainActivity.kt에 할 일 정보를 날짜순으로 모두 가져오도록 코드를 작성
- ▶ 할 일 정보를 sort() 메서드를 사용하여 날짜 순으로 내림차순 정렬하여 얻음
 - ▷ fieldName : 정렬할 열
 - ▷ sortOrder : 정렬 방법
 - DESCENDING : 내림차순
 - ASCENDING : 오름차순(기본값)

```
class MainActivity : AppCompatActivity() {
    val realm = Realm.getDefaultInstance()

    override fun onCreate(savedInstanceState: Bundle?) {
        // 전체 할 일 정보를 가져와서 날짜순으로 내림차순 정렬
        val realmResult = realm.where<Todo>().findAll().sort("date", Sort.DESCENDING)
    }
    override fun onDestroy() {
        super.onDestroy()
        realm.close()
    }
}
```

To do 리스트 앱 만들기

▶ 어댑터 작성

- ▶ 일반적으로 리스트 뷰의 어댑터는 BaseAdapter 클래스를 상속받아서 작성하지만 Realm을 사용할 때는 Realm에서 제공하는 RealmBaseAdapter 클래스를 상속받음
- ▶ RealmBaseAdapter를 사용하려면 우선 모듈 수준의 build.gradle 파일에 라이브러리 의존성을 추가하고 싱크

▶ implementation 'io.realm:android-adapters:2.1.1'

```
dependencies {
```

```
    implementation 'org.jetbrains.anko:anko-commons:0.10.5'
```

```
    implementation 'io.realm:android-adapters:2.1.1'
```

```
    implementation fileTree(dir: 'libs', include: ['*.jar'])
```

Gradle files have changed since last project sync. A project sync may be necessary for [Sync Now](#)

- ▶ 싱크 오류를 방지하기 위하여 따옴표는 직접 입력하고 의존성 주입때마다 싱크로 정상 동작 확인

To do 리스트 앱 만들기

▶ 어댑터 작성

▶ File → New → Kotlin File/Class를 선택하고 TodoListAdapter 클래스를 생성한 후 RealmBaseAdapter를 상속

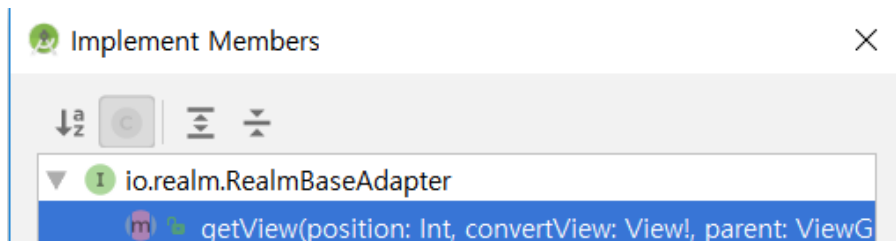
▶ RealmBaseAdapter는 OrderedRealmCollection<T>형 데이터를 받는 주 생성자를 가지고 있음

```
class TodoListAdapter(realmResult: OrderedRealmCollection<Todo>) :  
    RealmBaseAdapter<Todo>(realmResult) {  
  
}
```

To do 리스트 앱 만들기

▶ 어댑터 작성

- ▶ RealmBaseAdapter 클래스는 미구현 메서드가 포함된 추상 클래스이기 때문에 상속받은 클래스는 이를 구현해야 함
- ▶ 클래스 이름에 빨간 줄이 생기면 빨간 줄에 커서를 대고 단축키 ALT + Enter 를 누르거나 빨간 전구 아이콘을 클릭
- ▶ 제안 사항들이 표시되면 Implement members를 클릭하여 미구현 메서드를 구현



- ▶ 다음과 같이 미구현 메서드를 선택하는 화면이 나오면 getView() 메서드가 선택되었는지 확인하고 OK를 클릭
 - 추가한 후 TODO로 표시된 코드는 모두 삭제하거나 주석처리

To do 리스트 앱 만들기

▶ 어댑터 작성

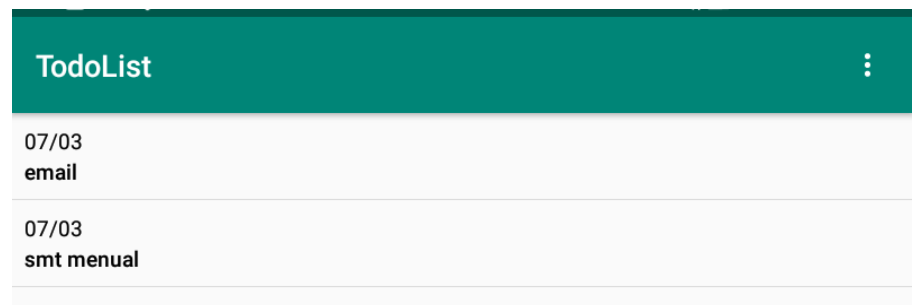
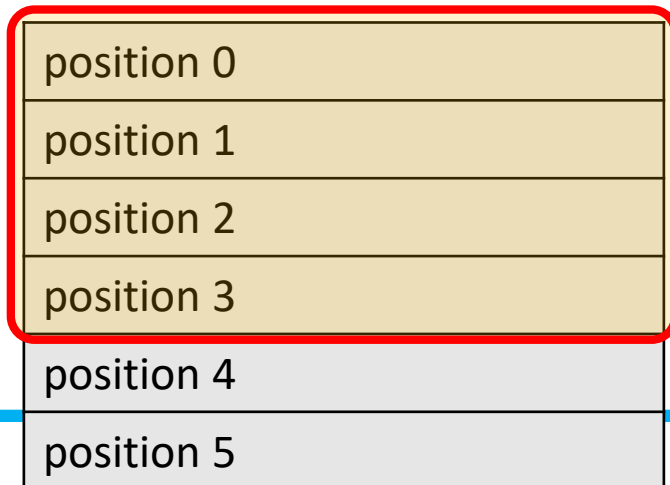
```
class TodoListAdapter(realmResult: OrderedRealmCollection<Todo>) :  
    RealmBaseAdapter<Todo>(realmResult) {  
    override fun getView(position: Int, convertView: View?, parent: ViewGroup?): View  
        {  
  
        }  
    }  
}
```

- ▶ getView() 메서드가 오버라이드 됨
- ▶ getView() 메서드에서 리스트 뷰의 각 아이템에 표시할 뷰를 구성
- ▶ 각 아이템이 화면에 보이기 전에 getView() 메서드가 한 번씩 호출 됨
 - ▷ position : 리스트 뷰의 아이템 위치
 - ▷ convertView : 재활용되는 아이템의 뷰
 - ▷ Parent : 부모 뷰, 여기서는 리스트 뷰를 참조

To do 리스트 앱 만들기

▶ 뷰 홀더 패턴 적용

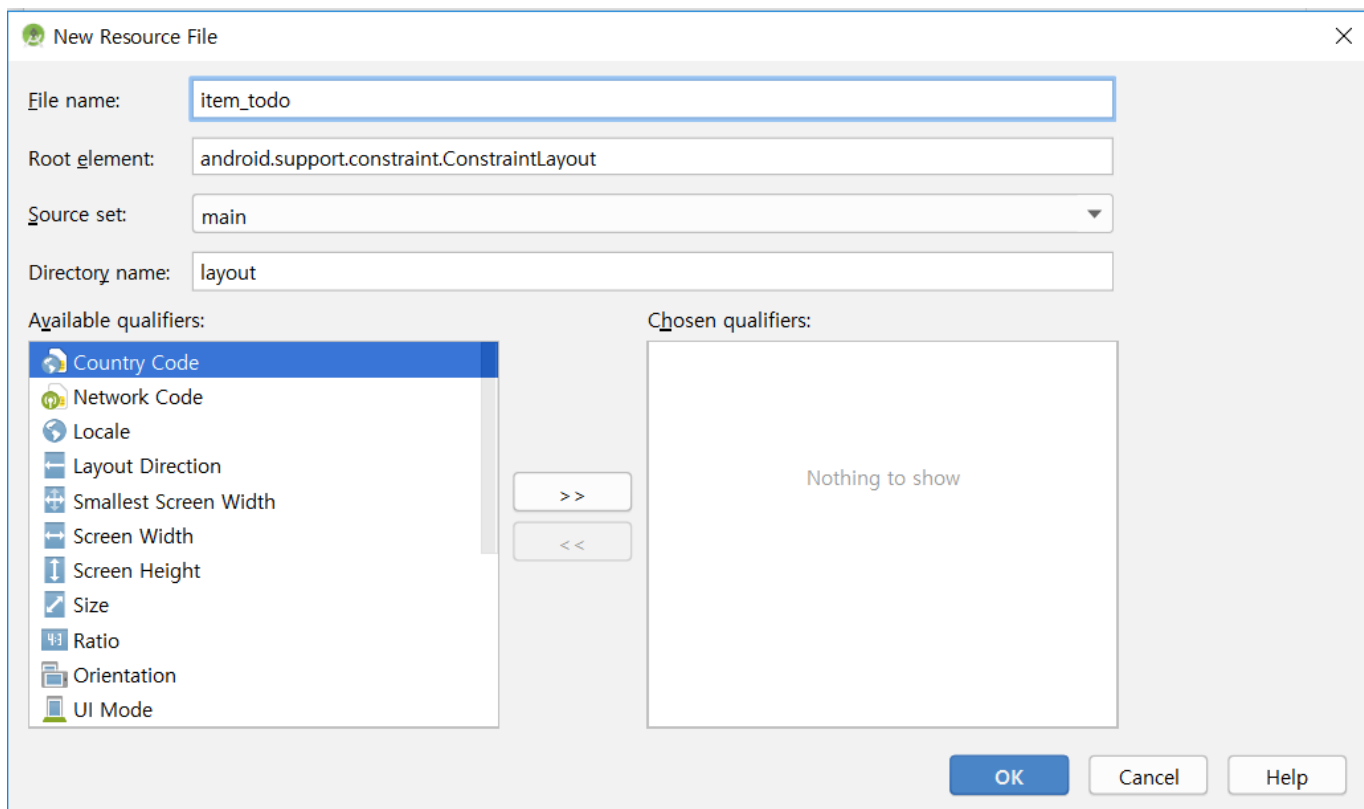
- ▶ 리스트 뷰에서는 리스트를 표시할 때 성능을 향상시킬 목적으로 일반적으로 뷰 홀더 패턴을 적용
- ▶ getView() 메서드가 아이템이 화면에 표시될 때마다 호출되므로 최대한 효율적인 코드를 작성해야 함
- ▶ 뷰 홀더 패턴은 한 번 작성한 레이아웃을 재사용하고 내용만 바꾸는 방법
- ▶ 다음 그림은 화면에 최대 4개의 아이템이 표시된다고 가정했을 때의 리스트 뷰를 나타낸 것
- ▶ 이 리스트 뷰에 뷰 홀더 패턴을 적용하면 스크롤 시 4, 5번 아이템은 0, 1번 아이템의 뷰를 재사용해 내용만 바꾸기 때문에 매번 뷰를 새로 생성할 필요가 없음
- ▶ 뷰 홀더 패턴을 한 번 만들어 둔 뷰를 최대한 재활용하여 성능을 높여주는 방법



To do 리스트 앱 만들기

▶ 아이템 레이아웃 작성

- ▶ `getView()` 메서드를 작성하기 전에 아이템에 표시할 레이아웃 리소스 파일을 작성
- ▶ File → New → layout resource file 을 클릭하면 새로운 리소스 파일을 생성하는 화면이 표시
- ▶ 파일 이름을 `item_todo`로 입력하고 OK를 클릭

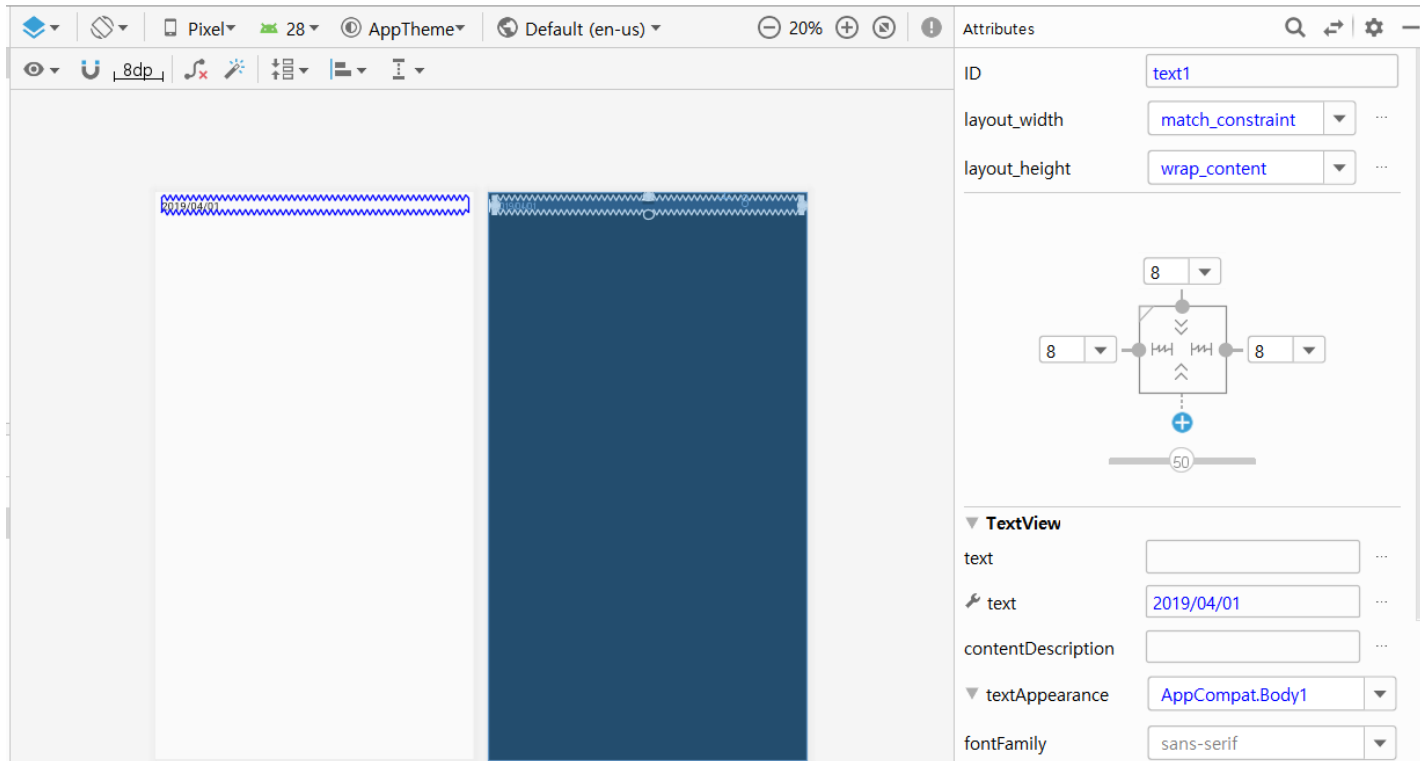


To do 리스트 앱 만들기

▶ 아이템 레이아웃 작성

▶ Item_todo.xml 파일에 표시할 텍스트 뷰를 두 개 배치

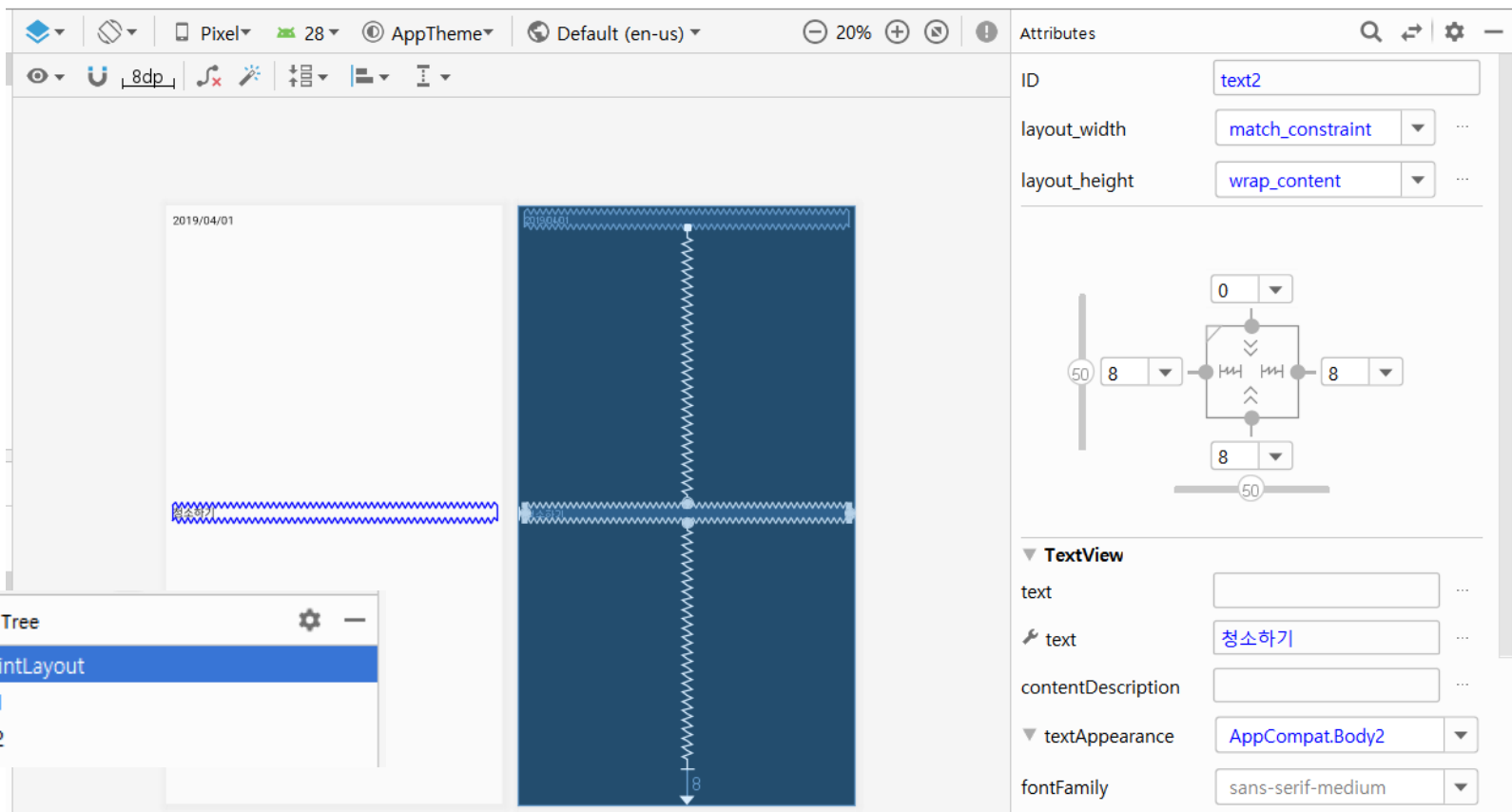
▶ 날짜를 표시할 텍스트 뷰를 배치



To do 리스트 앱 만들기

▶ 아이템 레이아웃 작성

- ▶ 할 일 내용을 표시할 두 번째 텍스트 뷰를 배치
- ▶ TextView를 드래그하여 첫 번째 텍스트 뷰 아래 적당한 위치에 배치



To do 리스트 앱 만들기

▶ 아이템 레이아웃 작성

▶ TodoListAdapter.kt 파일에 뷰 홀더 클래스를 별도의 클래스로 작성

▶ 전달 받은 뷰에서 text1과 text2 아이디를 가진 텍스트 뷰들의 참조를 저장

```
class TodoListAdapter(realmResult: OrderedRealmCollection<Todo>) :  
    RealmBaseAdapter<Todo>(realmResult) {  
  
}  
  
class ViewHolder(view: View) {  
    val dateTextView: TextView = view.findViewById(R.id.text1)  
    val textTextView: TextView = view.findViewById(R.id.text2)  
}
```

To do 리스트 앱 만들기

▶ 아이템 레이아웃 작성

▶ `TodoListAdapter.kt` 코드에 뷰 홀더 패턴을 적용하여 어댑터 코드를 작성

▷ `getView()` 메서드는 아이템이 화면에 보일 때마다 호출됨

▷ `getView()` 메서드의 두 번째 인자인 `convertView`는 아이템이 작성되기 전에는 `null`이고 한 번 작성되면 이전에 작성했던 뷰를 전달

```
override fun getView(position: Int, convertView: View?, parent: ViewGroup?): View? {  
    val vh: ViewHolder  
    val view: View  
  
    if (convertView == null) { //convertView가 null이면 레이아웃을 작성  
        view = LayoutInflater.from(parent?.context)  
            .inflate(R.layout.item_todo, parent, false)  
        //LayoutInflater 클래스는 XML 레이아웃 파일을 코드로 불러오는 기능을 제공  
        //LayoutInflater.from(parent?.context) 메서드로 객체를 얻고 inflate() 메서드로 XML 레이아웃 파일을 읽어서 뷰로 반환하여  
        //view 변수에 할당  
        vh = ViewHolder(view) //뷰 홀더 객체를 초기화. 뷰 홀더 클래스는 별도의 클래스로 먼저 작성  
        //뷰 홀더 클래스는 전달받은 view에서 text1과 text2 아이디를 가진 텍스트 뷰들의 참조(R.java)를 저장하는 역할을 함  
        view.tag = vh //뷰 홀더 객체는 tag 프로퍼티로 view에 저장됨  
        //Tag 프로퍼티는 Any 형으로 어떠한 객체도 저장할 수 있음  
    }  
}
```

To do 리스트 앱 만들기

▶ 아이템 레이아웃 작성

```
//XML 레이아웃 파일을 일어서 뷰로 반환  
inflate(resource: Int, root: ViewGroup, attachToRoot:Boolean)
```

- ▶ resource : 불러올 레이아웃 XML, 리소스 ID를 지정
- ▶ root : 불러온 레이아웃 파일이 붙을 뷰 그룹인 parent를 지정
- ▶ attachToRoot : XML 파일을 불러올 때는 false를 지정

```
override fun getView(position: Int, convertView: View?, parent: ViewGroup?): View? {  
    val vh: ViewHolder  
    val view: View  
  
    else { convertView가 null이 아니라면  
        view = convertView 이전에 작성했던 convertView를 재사용  
        vh = view.tag as ViewHolder 뷰 홀더 객체를 tag 프로퍼티로 꺼냄  
        반환되는 데이터형이 Any이므로 ViewHolder로 형변환  
    }  
    return view  
}
```

To do 리스트 앱 만들기

▶ 아이템 레이아웃 작성

▶ TodoListAdapter.kt 코드에 뷰 홀더 패턴을 적용하여 어댑터 코드를 작성

▶ RealmBaseAdapter는 adapterData 프로퍼티를 제공

▶ 프로퍼티로 저장된 데이터에 접근

```
//import android.text.format.DateFormat
override fun getView(position: Int, convertView: View?, parent: ViewGroup?): View? {
    val vh: ViewHolder
    val view: View

    if (adapterData != null) { //adapterData에 값이 있다면
        val item = adapterData!![position] //해당 위치의 데이터를 item 변수에 저장
        vh.textTextView.text = item.title //할 일 텍스트와 날짜를 각각 텍스트 뷰에 표시
        vh.dateTextView.text = DateFormat.format("MM/dd", item.date)
    } //DateFormat.format() 메서드는 지정한 형식으로 Long타입의 시간 데이터를 변환
    //DateFormat 클래스는 android.text.format.DateFormat을 импорт

    return view //완성된 view 변수를 반환
    //이 뷰는 다음에 호출되면 convertView로 재사용
}
```

To do 리스트 앱 만들기

▶아이템 레이아웃 작성

▶ TodoListAdapter.kt코드에 뷰 홀더 패턴을 적용하여 어댑터 코드를 작성

▶ getItemId() 메서드를 오버라이드

```
class TodoListAdapter(realmResult: OrderedRealmCollection<Todo>) :  
    RealmBaseAdapter<Todo>(realmResult) {  
  
    override fun getItemId(position: Int): Long {  
        //리스트 뷰를 클릭하여 이벤트를 처리할 때 인자로 position, id 등이 전달되는데 이때 전달되는 id값을 결정함  
        //데이터베이스를 다룰 때 레코드마다 고유한 아이디를 가지고 있는데 해당 정보를 반환하도록 정의  
        if (adapterData != null) {  
            return adapterData!![position].id //adapterView가 Realm 데이터를 가지고 있으므로  
                                                //요청한 해당 위치에 있는 데이터의 id 값을 반환하도록 함  
        }  
        return super.getItemId(position)  
    }  
}
```


To do 리스트 앱 만들기

▶ 할 일 목록 표시

▶ MainActivity 파일을 열고 할 일 목록이 표시되도록 코드를 추가

```
override fun onCreate(savedInstanceState: Bundle?) {  
    //TodoListAdapter 클래스에 할 일 목록인 realmResult를 전달하여 어댑터 인스턴스를 생성  
    val adapter = TodoListAdapter(realmResult)  
    //생성한 어댑터를 리스트 뷰에 설정하면 할 일 목록이 리스트 뷰에 표시됨  
    listView.adapter = adapter  
  
    // 데이터가 변경되면 어댑터에 적용 //addChangeListener를 구현하면 데이터가 변경될 때마다 어댑터에게 알림  
    realmResult.addChangeListener { _ -> adapter.notifyDataSetChanged() }  
    //어댑터에 notifyDataSetChanged() 메서드를 호출하면  
    //데이터 변경을 통지하여 리스트를 다시 표시  
    listView.setOnItemClickListener { parent, view, position, id ->  
        // 할 일 수정  
        startActivity<EditActivity>("id" to id) //EditActivity에 선택한 아이템의 id 값을 전달  
    }  
    기존 id 가 있는지 여부에 따라 새 할 일을 추가하거나 기존 할 일을 수정할 수 있음  
}
```

▶ 앱을 실행하여 추가한 할 일 목록이 첫 화면의 리스트에 표시되는지 확인.

▶ 표시된 목록을 클릭하여 수정과 삭제가 잘 동작하면 성공

연습문제

▶ 물품의 납품 스케줄과 내용을 저장하는 앱을 작성하시오.

The screenshot shows the 'ToDoList' app interface. At the top, there's a green header with the title 'ToDoList' and a menu icon. Below the header, the form contains the following text: '날짜 : 04/15', '장비명 : 노트북', '개수 : 8', and '배송지 : 포도전자'. Below this, there's a second section with '날짜 : 04/28', '장비명 : 오실로스코프', '개수 : 2', and '배송지 : 사과전자'. At the bottom, there are three buttons: '입력' (Input), '삭제' (Delete), and a greyed-out '삭제됨' (Deleted) button. The status bar at the top shows the time as 5:45 AM and battery at 67%.

The screenshot shows the 'ToDoList' app interface with a calendar view. At the top, there's a green header with the title 'ToDoList' and a menu icon. Below the header, there's a calendar for April 2019. The calendar shows the days of the week (S, M, T, W, T, F, S) and the dates. The date '6' is highlighted in a pink circle. Below the calendar, there are three input fields labeled '장비명' (Equipment Name), '개수' (Quantity), and '배송지' (Delivery Location). At the bottom, there are two buttons: a pink circular button with a trash icon and a blue button labeled '추가/수정' (Add/Modify). The status bar at the top shows the time as 5:56 AM and battery at 69%.

Q & A
