

# Android 애플리케이션 프로그래밍 - 내부 데이터 관리

SQLite, 파일 저장소, SharedPreferences



부산대학교 정보·의생명공학대학  
정보컴퓨터공학부



# 이론

# 강의 목표와 구성

## ❖ 내부 데이터 관리

- SQLite
- 파일 저장소
- SharedPreferences

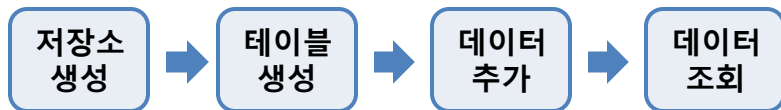
## ❖ 활용 단계

- 저장소 생성
- 테이블 생성
- 데이터 추가
- 데이터 조회

# 내부 데이터 관리 – SQLite

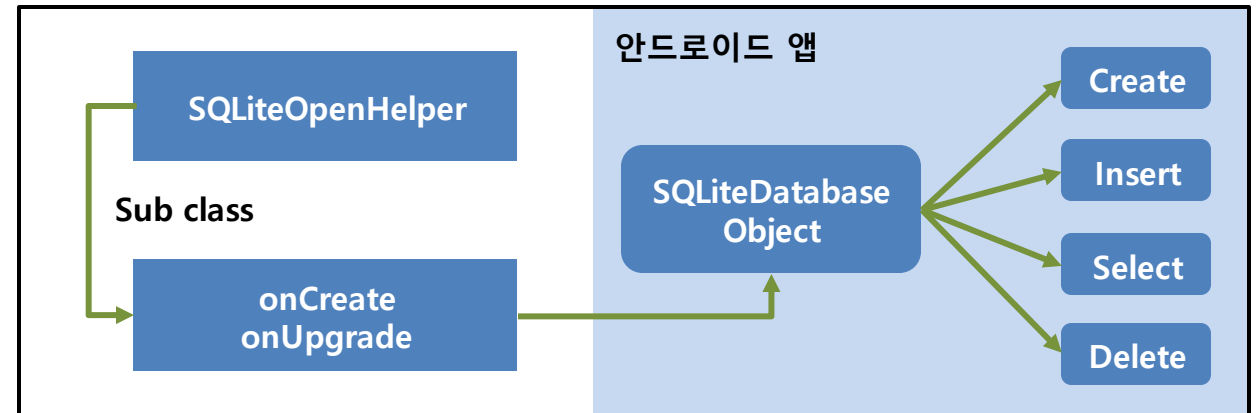
## ❖ SQLite 란?

- 앱의 데이터 관리를 위한 내부 SQL DB 관리 시스템
  - 낮은 메모리 사용량, 빠른 속도
  - 오픈소스
- SQLite 활용 단계
  - openOrCreateDatabase(저장소 생성)
  - Create (테이블 생성)
  - Insert (데이터 추가)
  - Select (데이터 조회)



## ❖ SQLiteDatabase

- SQLite 데이터베이스를 사용할 때 가장 중요한 핵심 요소 클래스
  - **openOrCreateDatabase() 함수를 이용해 객체를 생성한다.**
  - 객체가 생성된 후 아래 함수를 이용해 **질의문을 실행**할 수 있다.
    - **execSQL**: 데이터 조회 이외의 기능 실행할 때 사용
      - » Create, alter, drop, insert, update, delete 문을 실행하는 함수
    - **rawQuery**: 데이터 조회할 때 사용
      - » Select 문을 실행하는 함수



전체 아키텍처

# 저장소 생성 및 테이블 생성

## ❖ 저장소 생성

- openOrCreateDatabase 함수
  - 저장소가 존재하면 **저장소 이름 반환** 및 **Open** 수행
  - 저장소가 존재하지 않으면 **생성** 및 **Open** 수행

```
database = openOrCreateDatabase( name: "people",  
                                MODE_PRIVATE, factory: null)
```

## ❖ 테이블 생성

- Create table SQL 형식
  - CREATE TABLE [IF NOT EXISTS] table\_name(col\_name col\_definition ...) [table option] ...
  - **SQLite 컬럼 자료형**
    - SQLite는 컬럼 자료형을 참조형으로만 사용하기 때문에 실제 데이터와 자료형이 일치하지 않아도 에러가 발생하지 않는다.
    - Ex.) text, varchar(문자열) / smallint, integer(정수) ...

SQL

```
val tableName = "student"  
database?.execSQL( sql: "create table if not exists ${tableName}" +  
                    "( id integer PRIMARY KEY autoincrement, " +  
                    "name text, " +  
                    "age integer, " +  
                    "mobile text)")
```

컬럼 명-자료형

# 데이터 추가 및 조회

## ❖ 데이터 추가

- 레코드(Record) - 테이블에 추가되는 데이터
  - 문자열 데이터
- Insert into SQL 형식
  - INSERT INTO 테이블\_이름[(속성\_리스트)]VALUES(속성값\_리스트)

```
database?.execSQL( sql: "insert into ${tableName}" +  
    "(name,age,mobile)" +  
    "values"+  
    "('$john','20','010-0000-0000')")
```

레코드

## ❖ 데이터 조회

- 데이터를 조회할 때는 **rawQuery 함수**로 실행한다.
- Select SQL 형식
  - Select 속성\_리스트 from 테이블\_리스트
- **rawQuery**
  - 조회 결과를 Cursor 객체로 반환한다.
  - 해당 객체는 테이블에서 조회된 행의 집합이라 할 수 있다.

```
val cursor = database?.rawQuery( sql: "select _id,name,age,mobile " +  
    "from ${tableName}", selectionArgs: null)
```

```
if(cursor != null){  
    for (index in 0 until cursor.count){  
        cursor.moveToNext()  
        val id = cursor.getInt( columnIndex: 0)  
        val name = cursor.getString( columnIndex: 1)  
        val age = cursor.getInt( columnIndex: 2)  
        val mobile = cursor.getString( columnIndex: 3)  
        output1.append("레코드#${index} : " +  
            "$id,$name,$age,$mobile\n ")  
    }  
    cursor.close()  
}
```

조회 데이터

레코드#0 : 1, john, 20, 010-0000-0000

데이터 조회 결과

# 내부 데이터 관리 – SQLite

## ❖ SQLiteOpenHelper

- 데이터베이스를 관리(테이블 생성, 변경, 제거)하는 코드를 추상화해서 구조적으로 작성 지원
  - SQLiteOpenHelper는 추상 클래스이므로 이를 **상속받아 하위 클래스를 작성**한다.
    - onCreate: SQLiteOpenHelper 클래스가 이용되는 순간 한번 호출
    - onUpgrade: 생성자에 지정한 DB 버전 정보가 변경될 때마다 호출
  - SQLiteDatabase 객체도 **SQLiteOpenHelper 클래스를 이용해 생성**한다.

```
class DBHelper(context: Context?) : SQLiteOpenHelper(  
    context, name: "testDB", factory: null, version: 1) {  
    override fun onCreate(db: SQLiteDatabase) {  
    }  
    override fun onUpgrade(db: SQLiteDatabase?,  
                           oldVersion: Int, newVersion: Int) {  
    }  
}
```

SQLiteOpenHelper 내장 함수

```
var database: SQLiteDatabase? =  
    DBHelper(context: this).writableDatabase
```

# 내부 데이터 관리 - 파일 보관

## ❖ 파일 저장

- Android 앱에서 파일을 다룰 때는 대부분 java.io 패키지에서 제공하는 클래스를 이용한다.
  - File**: 파일 및 디렉토리를 지칭하는 클래스
  - FileInputStream / FileOutputStream**: 파일에서 바이트 스트림으로 데이터를 읽거나 쓰는 클래스
  - FileReader / FileWriter**: 파일에서 문자열 스트림으로 데이터 읽거나 쓰는 클래스
- Android에서 파일 저장소는 내장 메모리와 외장 메모리 공간으로 구분된다.

파일스트림  
쓰기

파일스트림  
읽기

```
val file = File(filesDir, child: "test.txt")
val writeStream: OutputStreamWriter = file.writer()
writeStream.write(str: "hello world")
writeStream.flush()

val readStream: BufferedReader = file.reader().buffered()
readStream.forEachLine { it: String
    Log.d(tag: "kkang", msg: "$it")
}
```

Java.io의 File 클래스 이용

## ❖ 내장 메모리

- 내장 메모리는 앱이 설치되면 **시스템에서 자동으로 할당**하는 공간
- Android 시스템은 앱에서 파일을 이용하지 않더라도 **앱의 패키지 명으로 디렉토리를 생성**한다.

Context  
저장

Context  
읽기

```
openFileOutput(name: "test.txt",
    Context.MODE_PRIVATE).use { it: FileOutputStream!
    it.write("hello world!!".toByteArray())
}

openFileInput(name: "test.txt")
    .bufferedReader().forEachLine { it: String
    Log.d(tag: "kkang", msg: "$it")
}
```

Context 객체 제공 함수 이용



# 내부 데이터 관리 - 파일 보관

외장 메모리 사용 가능 여부 판단

## ❖ 외장 메모리

- SD카드와 같은 외부 저장 장치를 의미함. 하지만 어떤 기기는 내부 저장소의 파티션을 나누어 제공할 수 있다.
- 모든 기기가 외장 메모리를 제공한다고 보장할 수 없으므로 **외장 메모리를 사용할 수 있는지 확인**해야 한다.
  - `getExternalStorageState()` 함수를 통해 얻은 값이 `MEDIA_MOUNTED`일 경우 외장 메모리 사용이 가능하다.
- 외장 메모리를 사용하기 위해서는 **파일을 읽고 쓰기 위한 권한이 필요**하다.
  - 메니페스트 파일에 `WRITE_EXTERNAL_STORAGE`, `READ_EXTERNAL_STORAGE`와 같은 퍼미션을 설정

```
if(Environment.getExternalStorageState()  
    == Environment.MEDIA_MOUNTED){  
    Log.d( tag: "kkang", msg: "ExternalStorageState Mounted")  
}else{  
    Log.d( tag: "kkang", msg: "ExternalStorageState UnMounted")  
}
```

D/kkang: ExternalStorageState Mounted

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

# 내부 데이터 관리 – 파일 보관

앱 저장소 위치 함수

## ❖ 외장 메모리

- 다른 앱에서도 이 저장소에 접근할 수 있게 하려면 **file provider**를 이용한다.
- 외장 메모리의 앱별 저장소 위치는 **getExternalFilesDir()** 함수를 이용한다.
  - getExternalFilesDir(null) 함수가 반환하는 위치는 다음과 같다.
    - `./storage/emulated/0/Android/data/패키지명/files`
    - 함수의 매개변수는 파일의 종류를 나타내며 null이 아닌 다음 값을 전달할 수 있다.
      - » DIRECTORY\_PICTURES, DIRECTORY\_DOCUMENTS, DIRECTORY\_MOVIES ...

```
val file: File = File(getExternalFilesDir(type: null),  
    child: "test.txt")  
val writeStream: OutputStreamWriter = file.writer()  
writeStream.write(str: "hello world")  
writeStream.flush()  
  
val readStream: BufferedReader =  
    file.reader().buffered()  
readStream.forEachLine { it: String  
    Log.d(tag: "kkang", msg: "$it")  
}
```

D/kkang: hello world

# SharedPreferences

## ❖ SharedPreferences

- 플랫폼 API에서 제공하는 클래스
- 데이터를 **키-값 형태**로 저장
- 내부적으로 내장 메모리의 앱 폴더에 XML 파일로 데이터 저장
- 객체를 얻는 방법은 2가지가 있다.
  - `Activity.getSharedPreferences(int mode)`: 액티비티 단위로 저장
  - `Context.getSharedPreferences(String mode, int mode)`: 앱 전체 데이터 저장

## ❖ 데이터 저장 및 불러오기

- 데이터를 저장하기 위해서는 `SharedPreferences.Editor` 클래스의 **함수**를 이용한다.
  - `putBoolean`, `putInt`, `putFloat`, `putLong`, `putString` ...
- **put~함수**를 이용해 데이터를 담고 **commit()** 호출 순간 저장된다.
- 저장된 데이터를 가져오려면 `SharedPreferences`의 **get~함수**를 이용한다.
  - `getBoolean`, `getInt`, `getFloat`, `getLong`, `getString` ...



# 실습

# 실습 목표와 구성

## 1. 기초(따라하기)

- SQLite를 이용한 데이터 관리
- 다양한 안드로이드 데이터 관리

## 2. 응용(로직구현)

- 영화 예매 정보 저장(SQLite)
- 알람 저장 기능 구현

# 기초(따라하기) – 예제 1

## ❖ SQLite를 이용한 데이터 관리

1. Gradle Script / XML 작성
2. 객체 및 리스너 작성
3. 저장소 생성
4. 테이블 생성
5. 데이터 추가
6. 데이터 갱신
7. 데이터 조회
8. 데이터 삭제



# 기초(따라하기) – 예제 1

## 1. Gradle Script / XML 작성

### ■ Gradle Script

- (4 line) :findViewById를 생략하기 해주는 플러그인

Build.gradle (:app)

```
1 plugins {  
2     id 'com.android.application'  
3     id 'kotlin-android'  
4     id 'kotlin-android-extensions'  
5 }
```

추가하기

### ■ XML

- 최상위 태그로 <LinearLayout>를 선언한 뒤, 다음 제공 코드를 선언(<LinearLayout>, <LinearLayout>, <ScrollView>)

activity\_main.xml

```
11 <LinearLayout  
12     android:layout_width="match_parent"  
13     android:layout_height="52dp">  
14     <Button  
15         android:id="@+id/doButton1"  
16         android:layout_width="wrap_content"  
17         android:layout_height="wrap_content"  
18         android:layout_weight="1"  
19         android:text="DB생성" />  
20     <Button  
21         android:id="@+id/doButton2"  
22         android:layout_width="wrap_content"  
23         android:layout_height="wrap_content"  
24         android:layout_weight="1"  
25         android:text="TB 생성" />  
26     <Button  
27         android:id="@+id/doButton3"  
28         android:layout_width="wrap_content"  
29         android:layout_height="wrap_content"  
30         android:layout_weight="1"  
31         android:text="Data 추가" />  
32 </LinearLayout>
```

```
33 <LinearLayout  
34     android:layout_width="match_parent"  
35     android:layout_height="52dp">  
36     <Button  
37         android:id="@+id/doButton4"  
38         android:layout_width="wrap_content"  
39         android:layout_height="wrap_content"  
40         android:layout_weight="1"  
41         android:text="Data 갱신" />  
42     <Button  
43         android:id="@+id/doButton5"  
44         android:layout_width="wrap_content"  
45         android:layout_height="wrap_content"  
46         android:layout_weight="1"  
47         android:text="Data 조회" />  
48     <Button  
49         android:id="@+id/doButton6"  
50         android:layout_width="wrap_content"  
51         android:layout_height="wrap_content"  
52         android:layout_weight="1"  
53         android:text="Data 제거" />  
54 </LinearLayout>
```

```
55 <ScrollView  
56     android:layout_width="match_parent"  
57     android:layout_height="match_parent"  
58     android:background="#03A9F4">  
59     <LinearLayout  
60         android:layout_width="match_parent"  
61         android:layout_height="wrap_content"  
62         android:orientation="vertical">  
63         <TextView  
64             android:id="@+id/output1"  
65             android:layout_width="match_parent"  
66             android:layout_height="wrap_content" />  
67     </LinearLayout>  
68 </ScrollView>
```

# 기초(따라하기) – 예제 1

## 2. 객체 및 리스너 작성

- (line 11) : SQLiteDatabase 객체 생성
- (line 18-35) : setOnClickListener 구현

MainActivity.kt

```
9 class MainActivity : AppCompatActivity() {
10     val databaseName = "people"
11     var database: SQLiteDatabase? = null
12     val tableName = "student"
13
14     override fun onCreate(savedInstanceState: Bundle?) {
15         super.onCreate(savedInstanceState)
16         setContentView(R.layout.activity_main)
17
18         doButton1.setOnClickListener { it: View!
19             createDatabase()
20         }
21         doButton2.setOnClickListener { it: View!
22             createTable()
23         }
24         doButton3.setOnClickListener { it: View!
25             addData()
26         }
27         doButton4.setOnClickListener { it: View!
28             updateData()
29         }
30         doButton5.setOnClickListener { it: View!
31             queryData()
32         }
33         doButton6.setOnClickListener { it: View!
34             deleteData()
35         }
36     }
```



# 기초(따라하기) – 예제 1

## 3. 저장소 생성

- (42 line) : 데이터베이스 생성 혹은 연결
- (44 line) : 데이터베이스 객체 생성 유무 확인
- (52 line) : 데이터베이스 객체 연결 해제

MainActivity.kt

```
38 fun createDatabase(){
39     database = openOrCreateDatabase(databaseName, MODE_PRIVATE, factory: null)
40     output1.append("데이터베이스 생성 또는 오픈 함\n")
41 }
42
43 fun checkDatabase() : Boolean {
44     if (database == null){
45         output1.append("데이터베이스를 먼저 오픈하세요.\n")
46         return true
47     }
48     return false
49 }
50
51 fun closeDatabase(){
52     database?.close()
53 }
```

## 4. 테이블 생성

- (56 line) : 데이터베이스 객체 생성 유무 확인
- (57 line) : 테이블 존재 시, 테이블 삭제 (실습 편의)
- (58 line) : 테이블 생성 sql 문 구성
- (63 line) : SQL 문 실행
- (64 line) : GUI의 Textview 내용 추가하기

MainActivity.kt

```
55 fun createTable(){
56     if(checkDatabase()) return
57     database?.execSQL( sql: "DROP TABLE IF EXISTS ${tableName}")
58     val sql = "create table if not exists ${tableName}" +
59         "(_id integer PRIMARY KEY autoincrement, " +
60         "name text, " +
61         "age integer, " +
62         "mobile text)"
63     database?.execSQL(sql)
64     output1.append("테이블 생성함\n")
65 }
```

# 기초(따라하기) – 예제 1

## 5. 데이터 추가

- (70 line) : sql insert 문 작성
- (73 line) : sql 문 실행

MainActivity.kt

```
67 fun addData(){
68     if(checkDatabase()) return
69
70     val sql = "insert into ${tableName}(name,age,mobile)" +
71         "values"+
72         "('john', '20', '010-0000-0000')"
73     database?.execSQL(sql)
74     output1.append("데이터 추가\n")
75 }
```

## 6. 데이터 갱신

- (80-83 line) : 갱신할 데이터 내용 구성
- (85 line) : sql update 문 수행
  - name 가 arr 인 데이터 튜플에 대해서 value로 내용을 변경함

MainActivity.kt

```
77 fun updateData(){
78     if(checkDatabase()) return
79
80     val values = ContentValues()
81     values.put("name", "mike")
82     values.put("age", "24")
83     values.put("mobile", "010-4000-4000")
84     val arr : Array<String> = arrayOf("john")
85     database?.update(tableName, values, whereClause: "name=?", arr)
86     output1.append("데이터 갱신\n")
87 }
```

# 기초(따라하기) – 예제 1

## 7. 데이터 조회

- (92 line) : sql select 문 구성
- (93 line) : sql query 수행
- (94 line) : 수행 결과가 존재하는지 유무 판단
- (95 – 103 line) : For 구문을 이용해 moveToNext 메서드 호출을 레코드 숫자만큼 반복

MainActivity.kt

```
89 fun queryData(){
90     if(checkDatabase()) return
91
92     val sql = "select _id,name,age,mobile from ${tableName}"
93     val cursor = database?.rawQuery(sql, selectionArgs: null)
94     if(cursor != null){
95         for(index in 0 until cursor.count){
96             cursor.moveToNext()
97             val id = cursor.getInt(0)
98             val name = cursor.getString(1)
99             val age = cursor.getInt(2)
100             val mobile = cursor.getString(3)
101             output1.append("레코드#${index} : " +
102                 "$id,$name,$age,$mobile\n")
103         }
104         cursor.close()
105     }
106
107     output1.append("데이터 조회 결과\n")
108 }
```

# 기초(따라하기) – 예제 1

## 8. 데이터 삭제

- (120 line) : sql delete 문 구성
- (121 line) : sql 문 실행

MainActivity.kt

```
110 fun deleteData(){
111     if(checkDatabase()) return
112
113     val sql = "select _id,name,age,mobile from ${tableName}"
114     val cursor = database?.rawQuery(sql, selectionArgs: null)
115     if(cursor != null){
116         cursor.count
117         val count = cursor.count
118         cursor.close()
119
120         val delete = "delete from ${tableName} where _id = ${count}"
121         database?.execSQL(delete)
122         output1.append("데이터 삭제\n")
123     }
124 }
```

# 기초(따라하기) – 예제 2

## ❖ 다양한 안드로이드 데이터 관리

1. 권한 및 레이아웃 작성
2. `setOnClickListener` 구성
3. 파일 클래스 활용
4. 파일 클래스 활용(외부)
5. 컨트랙트 함수 활용
6. Preferences 활용



# 기초(따라하기) – 예제 2

## 1. 권한 및 레이아웃 작성

- (line 3): 외부 저장소 읽기 권한 부여
- (line 4): 외부 저장소 쓰기 권한 부여
- (line 7): 레거시 외부 저장소 요청 허용

AndroidManifest.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3      package="com.example.week10_2">
4      <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
5      <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
6      <application
7          android:requestLegacyExternalStorage="true"
8          android:allowBackup="true"
```

추가하기

# 기초(따라하기) – 예제 2

## 2. setOnClickListener 구성

- (18-29 line) : setOnClickListener 구현

MainActivity.kt

```
13 class MainActivity : AppCompatActivity() {  
14     override fun onCreate(savedInstanceState: Bundle?) {  
15         super.onCreate(savedInstanceState)  
16         setContentView(R.layout.activity_main)  
17  
18         doButton1.setOnClickListener{ it: View!  
19             fileClass()  
20         }  
21         doButton2.setOnClickListener{ it: View!  
22             ExternelIF()  
23         }  
24         doButton3.setOnClickListener{ it: View!  
25             contextFunction()  
26         }  
27         doButton4.setOnClickListener{ it: View!  
28             sharedPreferences()  
29         }  
30     }
```

# 기초(따라하기) – 예제 2

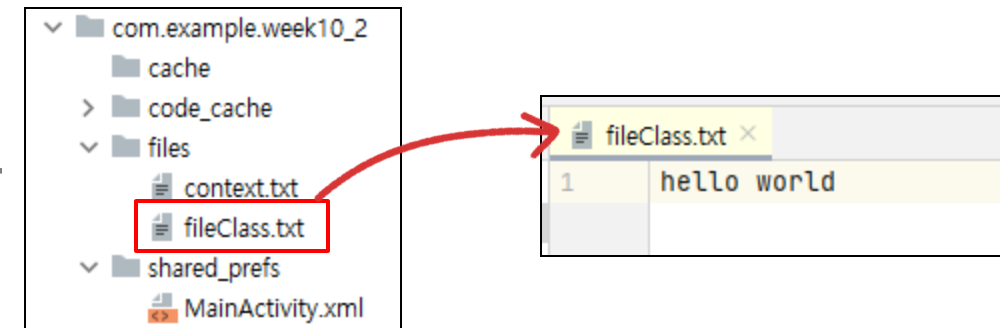
## 3. 파일 클래스 활용

- Java.IO 를 활용한 파일 입출력
- (34 line) : File 클래스 생성자 호출(위치, 파일명)
- (36 line) : OutputStreamWriter 생성
- (43 line) : BufferedReader 생성

MainActivity.kt

```
32 fun fileClass()
33 {
34     val file = File(filesDir, child: "fileClass.txt")
35     val str = "hello world"
36     val writeStream: OutputStreamWriter = file.writer()
37     output1.append("<파일 클래스>\n")
38
39     writeStream.write(str)
40     writeStream.flush()
41     output1.append("파일 쓰기 : $str\n")
42
43     val readStream: BufferedReader = file.reader().buffered()
44     output1.append("파일 읽기 : \n")
45     readStream.forEachLine { it: String
46         output1.append(" $it \n")
47     }
48     return
49 }
```

생성 결과



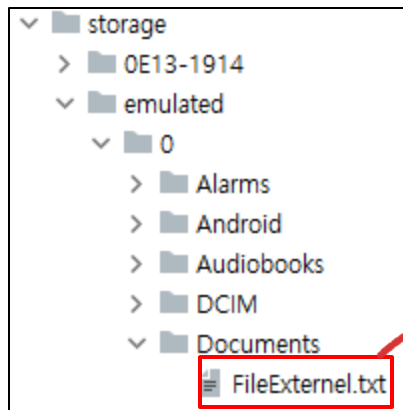


# 기초(따라하기) – 예제 2

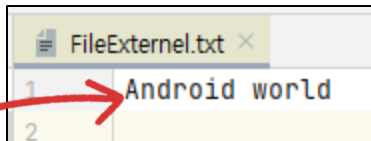
MainActivity.kt

## 4. 파일 클래스 활용(외부)

- (53 line) : 앱의 외부 저장소 권한 확인
- (63-65 line) : 파일 생성자 호출
  - Environment.DIRECTORY\_DOCUMENTS: 외부저장위치



생성 결과



```
51 fun ExternalIF()
52 {
53     if(Environment.getExternalStorageState() == Environment.MEDIA_MOUNTED){
54         output1.append("ExternalStorageState Mounted\n")
55         ExternalFile()
56     }
57     else{
58         output1.append("ExternalStorageState UnMounted\n")
59     }
60 }
61 fun ExternalFile()
62 {
63     val file: File = File(
64         Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOCUMENTS),
65         child: "FileExternal.txt")
66     val str = "Android world"
67     val writeStream: OutputStreamWriter = file.writer()
68     output1.append("<파일 클래스 외부>\n")
69
70     writeStream.write( str: "$str\n")
71     writeStream.flush()
72     output1.append("파일 쓰기 : \n")
73
74     val readStream: BufferedReader = file.reader().buffered()
75     readStream.forEachLine { it: String
76         output1.append(" $it\n")
77     }
78 }
```

# 기초(따라하기) – 예제 2

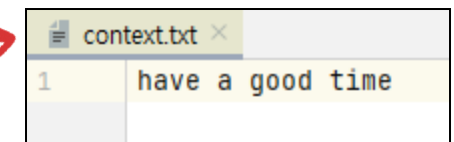
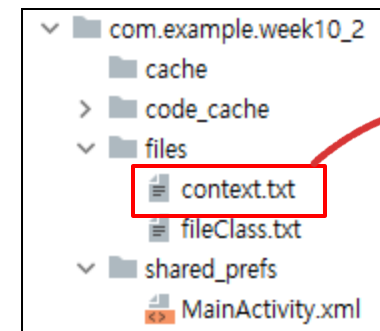
## 5. 컨트랙트 함수 활용

- 스트림을 이용한 파일 입출력
- (85 line) : openFileOutput() 함수 사용
  - 생성 파일을 비공개 설정
- (90 line) : openFileInput() 함수 사용

MainActivity.kt

```
80 fun contextFunction()
81 {
82     val str = "have a good time"
83     output1.append("<컨텍스트 함수>\n")
84
85     openFileOutput( name: "context.txt", Context.MODE_PRIVATE)
86         .use{it.write(str.toByteArray())}
87     output1.append("파일 쓰기 : $str\n")
88
89     output1.append("파일 읽기 : \n")
90     openFileInput( name: "context.txt")
91         .bufferedReader().forEachLine { it: String
92             output1.append(" $it\n")
93         }
94 }
```

생성 결과



# 기초(따라하기) – 예제 2

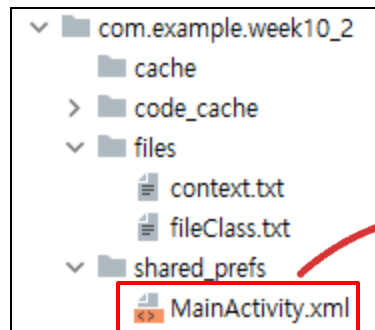
MainActivity.kt

## 6. Preferences 활용

- 앱 내부 데이터 유지 방법(key-value 구조)
  - 호출한 액티비티 클래스명으로 xml 파일 자동저장
- (102 line) : getPreferences 생성
- (103 line) : Preference 편집 수행 영역
- (104-105 line) : 입력 데이터 설정
- (106 line) : 데이터 저장 수행
- (110-111 line) : 데이터 가져오기

```
96 fun sharedPreferences()  
97 {  
98     val name = "Jack"  
99     val score = 100  
100    output1.append("<sharedPreferences>\n")  
101  
102    val sharedPref = getPreferences(Context.MODE_PRIVATE)  
103    sharedPref.edit().run { this: SharedPreferences.Editor!  
104        putString("name", name)  
105        putInt("score", score)  
106        commit() ^run  
107    }  
108    output1.append("키-값 저장 : $name, $score\n")  
109  
110    val data1 = sharedPref.getString("name", "none")  
111    val data2 = sharedPref.getInt("score", 0)  
112    output1.append("키-값 읽기 : $data1, $data2\n")  
113 }  
114 }
```

생성 결과



```
activity_main.xml x MainActivity.xml x AndroidManifest.xml x MainActivity.kt x  
1 <?xml version='1.0' encoding='utf-8' standalone='yes' ?>  
2 <map>  
3     <int name="score" value="100" />  
4     <string name="name">Jack</string>  
5 </map>  
6
```

# 응용(로직구현) – 예제 3

## ❖ SQLiteDatabase 통합 구현

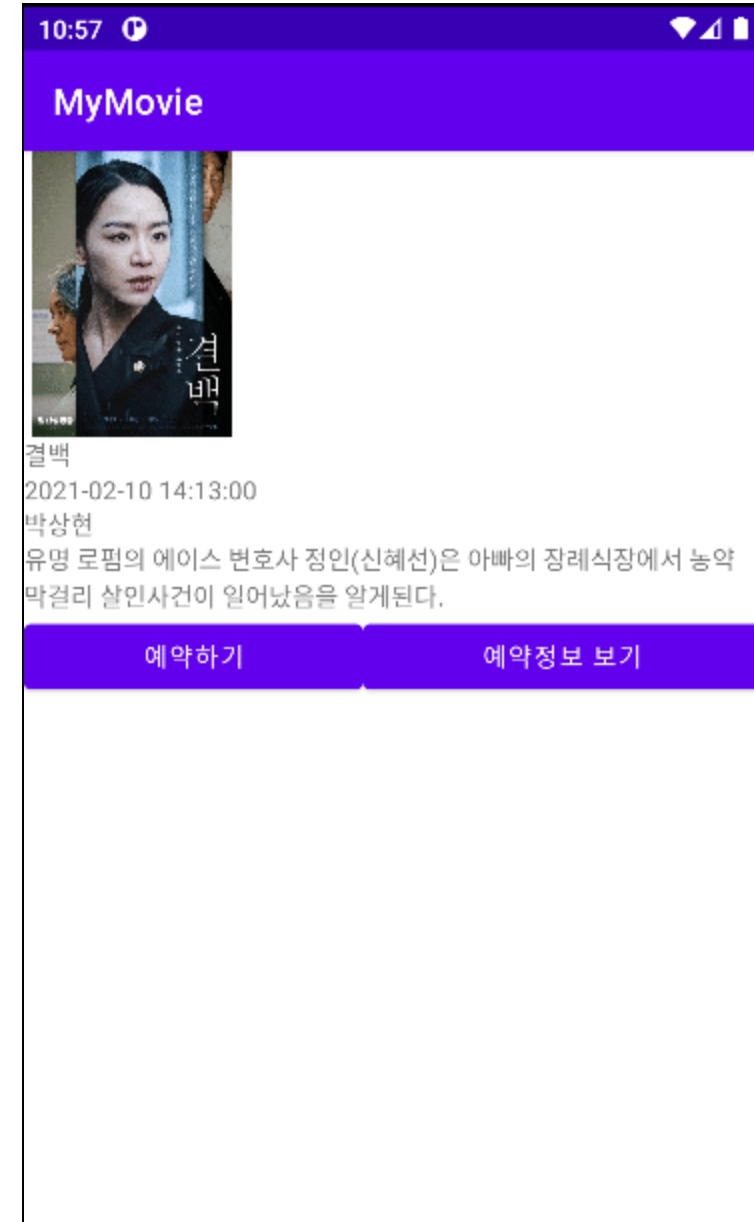
- SQLiteDatabase를 이용해 데이터를 추가하고 저장된 데이터를 확인할 수 있다.
- 본인이 희망하는 영화로 할 것!

1. 위젯 배치
2. 저장소 및 테이블 생성
3. 이미지 저장
4. 데이터 저장
5. 데이터 조회
6. 새로운 activity 생성

Build.gradle (:app)

```
1 plugins {  
2     id 'com.android.application'  
3     id 'kotlin-android'  
4     id 'kotlin-android-extensions'  
5 }
```

추가하기

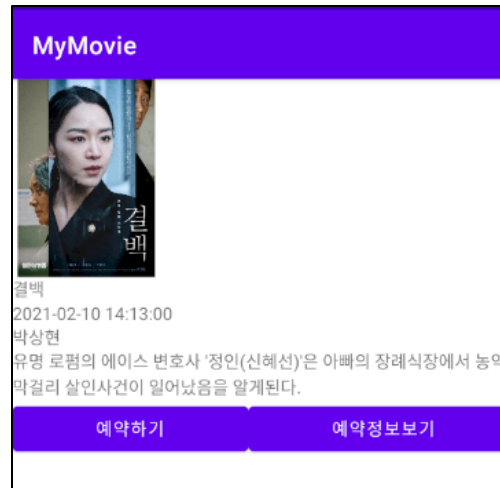


# 응용(로직구현) – 예제 3

activity\_main.xml

## 1. 위젯 배치

- 최상위 레이아웃 : LinearLayout
- (10 line) : 영화 이미지
- (14 line) : 이미지 로드 text
  - app/res/drawable/ 위치에 이미지 넣기
- (16 line) : 영화 제목 text
- (22 line) : 영화 예매 날짜 text
- (28 line) : 감독 text
- (34 line) : 줄거리 text
- (45 line) : 예약하기 버튼
- (52 line) : 예약정보보기 버튼



```
10 <ImageView
11     android:id="@+id/posterImageView"
12     android:layout_width="120dp"
13     android:layout_height="160dp"
14     android:src="@drawable/gg" />
15
16 <TextView...>
21
22 <TextView...>
27
28 <TextView...>
33
34 <TextView...>
39
40 <LinearLayout
41     android:layout_width="match_parent"
42     android:layout_height="match_parent"
43     android:orientation="horizontal">
44
45     <Button...>
50
51     <Button...>
56 </LinearLayout>
```

# 응용(로직구현) – 예제 3

## 2. 새로운 XML 및 activity 생성

- 새로운 XML을 “activity\_reserved”로 생성
  - 오른쪽 이미지와 동일하게 생성할 것
    - “activity\_main.xml”을 참조하여 전체 코드 구성
    - “예약된 영화”라는 제목을 위한 Textview 추가
    - 기존 Textview의 내용 삭제
    - 안드로이드 기본 아이콘으로 변경
    - 닫기 버튼 생성
- 새로운 액티비티를 “ReservedActivity” 로 생성
  - New – Activity – Empty activity
- (15-22 line) 객체 생성



ReservedActivity.kt

```
15 data class ReservedMovie(  
16     val _id:Int?,  
17     val name:String?,  
18     val poster_image:String?,  
19     val director: String?,  
20     val synopsis: String?,  
21     val reserved_time: String?  
22 ): Serializable
```

# 응용(로직구현) – 예제 3

## 2. 새로운 XML 및 activity 생성

- (36 line) : Serializable 자료형으로 가져와서 as 연산자를 이용해 ArrayList<ReservedMovie>? 로 형변환
- (40-43 line) : textview의 값 부여하기

ReservedActivity.kt

```
24 class ReservedActivity : AppCompatActivity() {
25     override fun onCreate(savedInstanceState: Bundle?) {
26         super.onCreate(savedInstanceState)
27         setContentView(R.layout.activity_reserved)
28
29         processIntent(intent)
30         btnClose.setOnClickListener { it: View!
31             finish()
32         }
33     }
34
35     fun processIntent(intent: Intent?) {
36         val movies = intent?.getSerializableExtra( name: "movies") as ArrayList<ReservedMovie>?
37         val movie = movies?.get(0)
38         if(movie!=null) {
39             posterImageView.setImageURI(Uri.parse(movie.poster_image))
40             input1.setText(movie.name)
41
42             

Textview 값 부여


43
44         }
45     }
46 }
```

# 응용(로직구현) – 예제 3

## 3. 저장소 및 테이블 생성

- MainActivity 기본 요소 구현
  - 객체 요소 변수, onCreate(), setOnClickListener 등등...
  - 액티비티 초기화시 테이블 삭제

Mainactivity.kt

```
17 class MainActivity : AppCompatActivity() {  
18     val databaseName = "movie"  
19     var database: SQLiteDatabase? = null  
20     val tableName = "movie_reserved"  
21     override fun onCreate(savedInstanceState: Bundle?) {...}
```

- (37-39 line) : 데이터베이스 저장소 생성
- (42 line) : 테이블 생성 SQL문 추가하기
  - 테이블이 존재하지 않는 경우에 생성하게끔 하기
  - 컬럼은 \_id, name, poster\_image, director, synopsis, reserved\_time
  - \_id 컬럼만 integer 나머지는 text 타입
  - \_id는 PRIMARY KEY 지정 및 autoincrement 속성 추가
- (50 line) : SQL문 실행 함수 추가하기

Mainactivity.kt

```
37 fun createDatabase() {  
38     database = openOrCreateDatabase(databaseName, MODE_PRIVATE, factory: null)  
39 }  
40  
41 fun createTable() {  
42     val sql  
43  
44  
45  
46  
47  
48  
49     if(database == null) return  
50  
51 }
```

SQL 문 구성

SQL 문 실행



# 응용(로직구현) – 예제 3

## 3. 데이터 저장

- (53-63 line): 위젯 값 받아오기

Mainactivity.kt

```
53 fun saveMovie() {  
54     val posterImageUri = savePosterToFile(R.drawable.gg)  
55  
56     val name = input1.text.toString()  
57     val reserved_time = input2.text.toString()  
58     val director = input3.text.toString()  
59     val synopsis = input4.text.toString()  
60     val poster_image = posterImageUri.toString()  
61  
62     addData(name, poster_image, director, synopsis, reserved_time)  
63 }
```

# 응용(로직구현) – 예제 3

## 4. 이미지 저장

- 이미지는 보통 파일로 저장하고 파일의 경로만 데이터베이스에 저장
- (65 line) : getDrawable 함수를 호출하면 파라미터로 전달한 id를 이용해 객체 생성
- (66 line) : ContextWrapper 객체 생성 후 getDir 함수 호출하면 단말 내부의 저장소에 접근 가능
- (68-70 line) : images 폴더 참조하도록 한 후, file 객체 생성
- (72-80 line) : 비트맵 객체의 메서드 호출하면서 이미지 파일 저장 후 파일 경로 Uri 객체로 반환

Mainactivity.kt

```
64 fun savePosterToFile(drawable:Int): Uri {
65     val drawable = ContextCompat.getDrawable(applicationContext,drawable)
66     val bitmap = (drawable as BitmapDrawable).bitmap
67
68     val wrapper = ContextWrapper(applicationContext)
69     val imagesFolder = wrapper.getDir( name: "images", Context.MODE_PRIVATE)
70     val file = File(imagesFolder, child: "gg.jpg")
71
72     try{
73         val stream: OutputStream = FileOutputStream(file)
74         bitmap.compress(Bitmap.CompressFormat.JPEG, quality: 100, stream)
75         stream.flush()
76         stream.close()
77     }catch (e: IOException){
78         e.printStackTrace()
79     }
80     return Uri.parse(file.absolutePath)
81 }
```

# 응용(로직구현) – 예제 3

## 5. 데이터 저장

- (82 line) : 데이터 추가
- (85-87 line) : 데이터 저장소가 없을 때
- (89 line) : 값들 array 객체로 전달해서 SQL문 실행

Mainactivity.kt

```
82 fun addData(name:String, poster_image:String, director:String, synopsis:String, reserved_time:String){
83     val sql = Insert SQL 문 구성
84
85     if(database==null){
86         println("데이터베이스를 먼저 오픈하세요\n")
87         return
88     }
89     database?.execSQL(sql, arrayOf(name,poster_image,director,synopsis,reserved_time))
90     println("데이터 추가함\n")
91 }
```

# 응용(로직구현) – 예제 3

## 6. 데이터 조회

- (94 line) : select SQL문을 이용해 데이터 조회
- (96 line) : intent 생성
- (97-98 line) : 데이터 전달 및 화면 전환

Mainactivity.kt

```
93 fun loadMovie() {  
94     val movies = queryData()  
95  
96     val intent = Intent( packageContext: this, ReservedActivity::class.java)  
97     intent.putExtra( name: "movies", movies)  
98     startActivity(intent)  
99 }
```

# 응용(로직구현) – 예제 3

## 6. 데이터 조회

- (101 line) : 데이터 조회하는 select SQL문 추가하기
- (107 line) : 조회한 칼럼들을 하나의 객체로 만들어 반환하기 위해 ReservedMovie(데이터 클래스) 사용
- (108 line) : SQL문 실행

Mainactivity.kt

```
100 fun queryData():ArrayList<ReservedMovie>?{
101     val sql = "select _id,name,poster_image,director,synopsis,reserved_time from ${tableName}"
102
103     if(database == null){
104         println("데이터베이스를 먼저 오픈하세요.\n")
105         return null
106     }
107     val list= arrayListOf<ReservedMovie>()
108     val cursor = database?.rawQuery(sql, selectionArgs: null)
```

# 응용(로직구현) – 예제 3

## 6. 데이터 조회

- (109-133line) : moveToNext 함수를 이용해 칼럼 값 조회
  - For문을 이용해 객체의 count 속성에서 가리키는 레코드 숫자만큼 반복
- (123-127line) : SQL 테이블의 데이터가 없는 경우에도 “예약정보보기” 클릭 시 앱 비정상 종료하지 않게끔 조치

Mainactivity.kt

```
109     if(cursor!=null){
110         for (index in 0 until cursor.count){
111             cursor.moveToNext()
112             val _id = cursor.getInt(0)
113             val name = cursor.getString(1)
114             val poster_image = cursor.getString(2)
115             val director =
116             val synopsis =
117             val reserved_time = cursor.getString(5)
118             println("레코드# ${index}: $_id, $name, $poster_image, $director, $synopsis, $reserved_time\n")
119
120             val movie = ReservedMovie(_id,name,poster_image,director,synopsis,reserved_time)
121             list.add(movie)
122         }
123         if (cursor.count == 0 )
124             {...}
128         cursor.close()
129     }
130     println("데이터 조회함\n")
131     return list
132 }
133 }
```

**칼럼 값 가져오기**

# 응용(로직구현) – 예제 4

## ❖ 알람 저장 기능 구현

- SharedPreferences를 이용해 내부 데이터 저장을 할 수 있다.

1. 위젯 배치
2. 데이터 저장
3. 새로운 activity 생성
4. 데이터 불러오기

Build.gradle (:app)

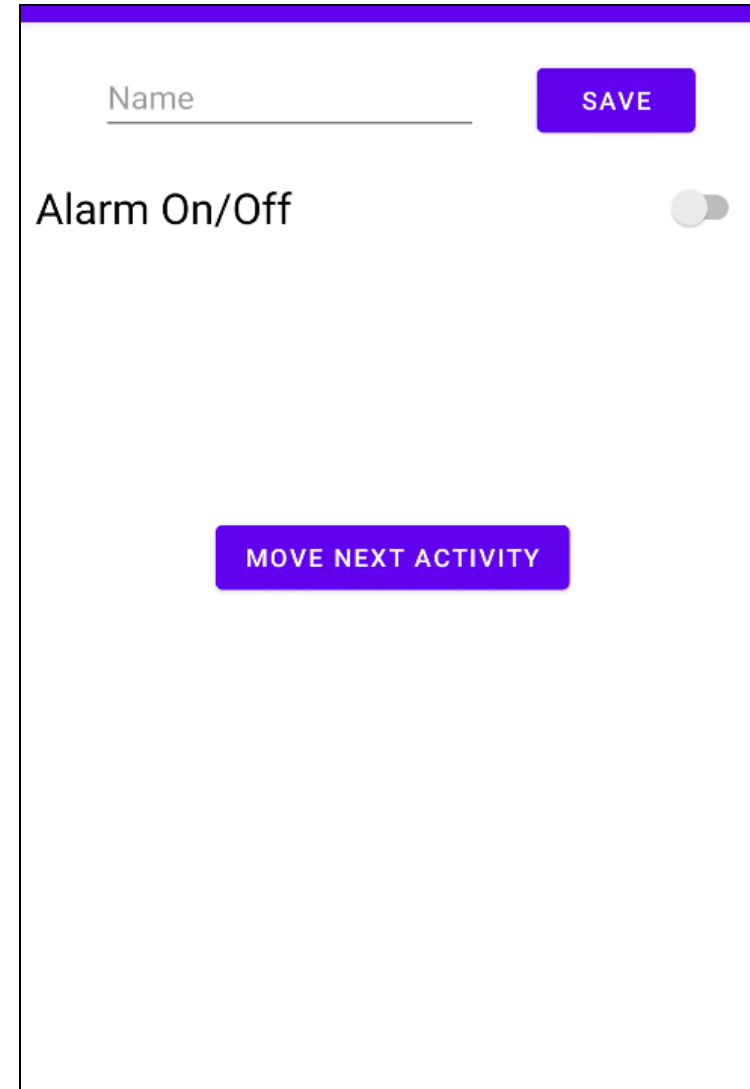
```
1 plugins {  
2     id 'com.android.application'  
3     id 'kotlin-android'  
4     id 'kotlin-android-extensions'  
5 }
```

추가하기

AndroidManifest.xml

```
12 <activity  
13     android:name=".MainActivity2"  
14     android:exported="false" />  
15 <activity  
16     android:name=".MainActivity"  
17     android:exported="true">
```

자동 추가  
확인하기



# 응용(로직구현) – 예제 4

Activity\_main.xml

## 1. 위젯 배치

- (3 line) : Constraintlayout 사용
  - 태그에 app:layout\_constraint~~속성을 사용할 것!
- (10 line) : 화면 전환 button
  - Button 이름 : Move Next Activity
- (20 line) : alarm switch
- (32 line) : 사용자 입력 edit text
- (44 line) : 입력 저장 button
  - Button 이름 : Save

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <androidx.constraintlayout.widget.ConstraintLayout
3     xmlns:android="http://schemas.android.com/apk/res/android"
4     xmlns:app="http://schemas.android.com/apk/res-auto"
5     xmlns:tools="http://schemas.android.com/tools"
6     android:layout_width="match_parent"
7     android:layout_height="match_parent"
8     tools:context=".MainActivity">
9
10    <Button...>
19
20    <Switch
21        android:id="@+id/vSwitchAlarm"
22        android:layout_width="0dp"
23        android:layout_height="65dp"
24        android:layout_marginTop="72dp"
25        android:text="Alarm On/Off"
26        android:textSize="24sp"
27        android:padding="8dp"
28        app:layout_constraintEnd_toEndOf="parent"
29        app:layout_constraintStart_toStartOf="parent"
30        app:layout_constraintTop_toTopOf="parent" />
31
32    <EditText...>
43
44    <Button...>
54
55 </androidx.constraintlayout.widget.ConstraintLayout>
```



# 응용(로직구현) – 예제 4

## 2. 데이터 저장

- (12-13 line) : getSharedPreferences 을 이용한 초기화
- (19-20 line) : name, mode 지정
- (22-28 line) : editText 값 저장하고 공백으로 만든 후 toast 메시지 출력
- (30-34 line) : switch 값이 바뀔 때, option에 저장

MainActivity.kt

```
11 class MainActivity : AppCompatActivity() {
12     lateinit var option: SharedPreferences
13     lateinit var userInfo: SharedPreferences
14
15     override fun onCreate(savedInstanceState: Bundle?) {
16         super.onCreate(savedInstanceState)
17         setContentView(R.layout.activity_main)
18
19         option = getSharedPreferences( name: "option", MODE_PRIVATE)
20         userInfo = getSharedPreferences( name: "user_info", MODE_PRIVATE)
21
22         vBtnSave.setOnClickListener { it: View!
23             userInfo.edit { this: SharedPreferences.Editor
24                 EditText값 저장하고 공백으로 만들기
25             }
26             저장 되었다는 Toast 메시지 출력
27         }
28     }
29
30     vSwitchAlarm.setOnCheckedChangeListener { compoundButton, b ->
31         option.edit { this: SharedPreferences.Editor
32             putBoolean("alarm", vSwitchAlarm.isChecked)
33         }
34     }
```

# 응용(로직구현) – 예제 4

## 3. 새로운 activity 생성

- MainActivity2.xml 생성
- Constraintlayout, TextView 태그 생성
- MainActivity.kt 코드 추가하기
- (36-40 line) : Move Next Activity 버튼을 클릭했을 때, 화면 전환이 이뤄지도록 intent를 사용하기
- (37-38 line) : intent 객체 생성 후 MainActivity2 화면 전환 추가하기

activity\_main2.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <androidx.constraintlayout.widget.ConstraintLayout
3      xmlns:android="http://schemas.android.com/apk/res/android"
4      xmlns:app="http://schemas.android.com/apk/res-auto"
5      xmlns:tools="http://schemas.android.com/tools"
6      android:layout_width="match_parent"
7      android:layout_height="match_parent"
8      tools:context=".MainActivity2">
9
10     <TextView...>
20 </androidx.constraintlayout.widget.ConstraintLayout>
```

TextView

MainActivity.kt

```
36  vBtnMoveActivity.setOnClickListener { it: View!
37
38      화면 전환 추가하기
39  }
40  }
```

# 응용(로직구현) – 예제 4

## 4. 데이터 불러오기

- MainActivity2.ky 생성
- (16-17 line) : getSharedPreferences 함수 이용 추가하기
- (19-22 line) : SharedPreferences를 이용해 저장된 값 가져오기
- MainActivity.ky 코드 추가하기
- (47-51 line) : 앱을 종료 후 실행했을 경우에도 데이터를 가져 오도록 MainActivity에서 onStart()함수 이용하기

MainActivity.kt

```
42 override fun onStart() {  
43     saveValueLoad() // Value Load  
44     super.onStart()  
45 }  
46  
47 private fun saveValueLoad() {  
48     vEditName.setText(userInfo.getString("name", ""))  
49  
50     vSwitchAlarm.isChecked = option.getBoolean("alarm", false)  
51 }
```

MainActivity2.kt

```
8 class MainActivity2 : AppCompatActivity() {  
9     lateinit var option: SharedPreferences  
10    lateinit var userInfo: SharedPreferences  
11  
12    override fun onCreate(savedInstanceState: Bundle?) {  
13        super.onCreate(savedInstanceState)  
14        setContentView(R.layout.activity_main2)  
15  
16        option =  
17        userInfo = getSharedPreferences 추가하기  
18  
19        var str = "" "User Name: ${userInfo.getString("name", "NULL")}  
20        |  
21        |Alarm On/Off: ${boolToText(option.getBoolean("alarm", false))}  
22        """.trimMargin()  
23  
24        vTextResult.text = str  
25    }  
26  
27    fun boolToText(check: Boolean): String {  
28        return when(check) {  
29            true -> "On"  
30            else -> "Off"  
31        }  
32    }  
33 }
```