세상의 속도를
따라잡고 싶다면

Do it!

# 깡샘의 안드로이드 앱 프로그래밍 with 코틀린

이지스퍼블리싱(주)

17

# 저장소에 데이터 보관하기



이지스퍼블리싱(주)

# 17-1 데이터베이스에 보관하기

- 안드로이드폰에서 이용하는 데이터베이스 관리 시스템은 오픈소스로 만들어진 SQLite(sqlite.org)

## 질의문 작성하기

- SQLite를 사용하려면 SQLiteDatabase라는 API를 이용
- SQLiteDatabase 객체는 openOrCreateDatabase() 함수를 호출해서 얻습니다.

• 데이터베이스 객체 생성

```
val db = openOrCreateDatabase("testdb", Context.MODE_PRIVATE, null)
```

# 17-1 데이터베이스에 보관하기

- SQLiteDatabase 객체에 정의된 다음 함수를 이용하면 질의문을 실행
  - public void execSQL(String sql, Object[] bindArgs)
  - public Cursor rawQuery(String sql, String[] selectionArgs)

• 테이블 생성(create 문)

```
db.execSQL("create table USER_TB (" +
           "_id integer primary key autoincrement," +
           "name not null," +
           "phone)")
```

• 데이터 삽입(insert 문)

```
db.execSQL("insert into USER_TB (name, phone) values (?,?)",
    arrayOf<String>("kkang", "0101111"))
```

• 데이터 조회(select 문)

```
val cursor= db.rawQuery("select * from USER_TB", null)
```

# 17-1 데이터베이스에 보관하기

- rawQuery() 함수의 반환값은 Cursor
- Cursor 객체로 행을 선택
  - public abstract boolean moveToFirst(): 첫 번째 행을 선택합니다.
  - public abstract boolean moveToLast(): 마지막 행을 선택합니다.
  - public abstract boolean moveToNext(): 다음 행을 선택합니다.
  - public abstract boolean moveToPosition(int position): 매개변수로 지정한 위치의 행을 선택합 니다.
  - public abstract boolean moveToPrevious(): 이전 행을 선택합니다.
- 행의 열 데이터를 가져오려면 타입에 따라 함수를 이용
  - public abstract String getString(int columnIndex)
  - public abstract int getInt(int columnIndex)
  - public abstract double getDouble(int columnIndex)

• 선택한 행의 값 가져오기

```
while (cursor.moveToNext()) {
    val name = cursor.getString(0)
    val phone = cursor.getString(1)
}
```

# 17-1 데이터베이스에 보관하기

- insert(), update(), delete(), query() 함수를 사용
  - public long insert(String table, String nullColumnHack, ContentValues values)
  - public int update(String table, ContentValues values, String whereClause, String[] whereArgs)
  - public int delete(String table, String whereClause, String[] whereArgs)
  - public Cursor query(String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy)

• insert() 함수 사용

```
val values = ContentValues()
values.put("name", "kkang")
values.put("phone", "0101112")
db.insert("USER_TB", null, values)
```

# 17-1 데이터베이스에 보관하기

- query() 함수의 각 매개변수
  - table: 조회할 테이블명입니다.
  - columns: 가져올 값이 담긴 열 이름을 배열로 지정합니다.
  - selection: select 문의 where 절 뒤에 들어갈 문자열입니다.
  - selectionArgs: 질의문에서 ?에 들어갈 데이터 배열입니다.
  - groupBy: select 문의 group by 절 뒤에 들어갈 문자열입니다.
  - having: select 문의 having 조건입니다.
  - orderBy: select 문의 order by 조건입니다.

• query() 함수 사용

```
val cursor = db.query("USER_TB", arrayOf<String>("name", "phone"), "phone=?",
    arrayOf<String>("0101112"), null, null, null)
```

# 17-1 데이터베이스에 보관하기

## 데이터베이스 관리하기

- SQLiteOpenHelper 클래스를 이용하면 데이터베이스 프로그램을 좀 더 구조적으로 작성할 수 있습니다.
- SQLiteOpenHelper는 추상 클래스이므로 이를 상속받아 하위 클래스를 작성
  - onCreate(): 앱이 설치된 후 SQLiteOpenHelper 클래스가 이용되는 순간 한 번 호출합니다.
  - onUpgrade(): 생성자에 지정한 DB 버전 정보가 변경될 때마다 호출합니다.

• SQLiteOpenHelper의 하위 클래스 작성

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

# 17-1 데이터베이스에 보관하기

- SQLiteDatabase 객체도 SQLite OpenHelper 클래스를 이용해 생성

• 데이터베이스 객체 생성

```
val db: SQLiteDatabase = DBHelper(this).writableDatabase
```

# 17-2 파일에 보관하기

- 안드로이드 앱에서 파일을 다룰 때는 대부분 java.io 패키지에서 제공하는 클래스를 이용
  - File: 파일 및 디렉터리를 지칭하는 클래스입니다.
  - FileInputStream / FileOutputStream: 파일에서 바이트 스트림으로 데이터를 읽거나 쓰는 클래스입니다.
  - FileReader / FileWriter: 파일에서 문자열 스트림으로 데이터를 읽거나 쓰는 클래스입니다.

# 17-2 파일에 보관하기

## 내장 메모리의 파일 이용하기

- 앱의 패키지명으로 디렉터리를 만들어 주는데, 이 디렉터리가 바로 앱의 내장 메모리 공간
- 파일을 내장 메모리에 저장하려면 java.io의 File 클래스를 이용

• 파일 객체 생성 후 데이터 쓰기

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

• 파일의 데이터 읽기

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

• Context 객체의 함수 사용

```
openFileOutput("test.txt", Context.MODE_PRIVATE).use {
    it.write("hello world!!".toByteArray())
}
openFileInput("test.txt").bufferedReader().forEachLine {
    Log.d("kkang", "$it")
}
```

# 17-2 파일에 보관하기

## 외장 메모리의 파일 이용하기

- Environment.getExternalStorageState() 함수로 외장 메모리를 사용할 수 있는지부터 확인

• 외장 메모리 사용 가능 여부 판단

```
if (Environment.getExternalStorageState() == Environment.MEDIA_MOUNTED) {
    Log.d("kkang", "ExternalStorageState MOUNTED")
} else {
    Log.d("kkang", "ExternalStorageState UNMOUNTED")
}
```

# 17-2 파일에 보관하기

- 매니페스트 설정

• 매니페스트에 외장 메모리 사용 설정

```
<manifest ... 생략 ... >
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <application
        (... 생략 ...)
        android:requestLegacyExternalStorage="true">
        (... 생략 ...)
    </application>
</manifest>
```

# 17-2 파일에 보관하기

- 앱별 저장소 이용
  - 외장 메모리 공간은 앱별 저장소와 공용 저장소로 구분
  - 앱별 저장소는 개별 앱에 할당된 공간
  - 앱별 저장소의 파일을 외부 앱에서 접근하게 하려면 파일 프로바이더로 공개해야 합니다.
  - 외장 메모리의 앱별 저장소 위치는 getExternalFilesDir() 함수로 구합니다.

• 앱별 저장소에 접근

```
val file: File? = getExternalFilesDir(null)
Log.d("kkang", "${file?.absolutePath}")
```

# 17-2 파일에 보관하기

- getExternalFilesDir() 함수를 이용할 때 매개변수는 파일의 종류를 나타내며 null이 아닌 다음과 같은 Environment의 상수를 전달
  - Environment.DIRECTORY_PICTURES
  - Environment.DIRECTORY_DOCUMENTS
  - Environment.DIRECTORY_MUSIC
  - Environment.DIRECTORY_MOVIES

• 앱별 저장소에 파일 쓰기와 읽기

```
// 파일 쓰기
val file: File = File(getExternalFilesDir(null), "test.txt")
val writeStream: OutputStreamWriter = file.writer()
writeStream.write("hello world")
writeStream.flush()
// 파일 읽기
val readStream: BufferedReader = file.reader().buffered()
readStream.forEachLine {
    Log.d("kkang", "$it")
}
```

# 17-2 파일에 보관하기

- 다른 앱에서도 이 저장소에 접근할 수 있게 하려면 파일 프로바이더를 이용

• 외장 메모리의 앱별 저장소 파일을 다른 앱에서 접근

```
// 파일 생성
val timeStamp: String = SimpleDateFormat("yyyyMMdd_HHmmss").format(Date())
val storageDir: File? = getExternalFilesDir(Environment.DIRECTORY_PICTURES)
val file = File.createTempFile(
    "JPEG_${timeStamp}_",
    ".jpg",
    storageDir
)
filePath = file.absolutePath
// 파일 Uri 획득
val photoURI: Uri = FileProvider.getUriForFile(
    this,
    "com.example.test17.fileprovider", file
)
// 카메라 앱 실행
val intent = Intent(MediaStore.ACTION_IMAGE_CAPTURE)
intent.putExtra(MediaStore.EXTRA_OUTPUT, photoURI);
requestCameraFileLauncher.launch(intent)
```

```
java.lang.IllegalArgumentException: Couldn't find meta-data for provider with authority
com.example.test17_provider.fileprovider
```

# 17-2 파일에 보관하기

- 프로젝트의 res/xml 디렉터리에 XML 파일을 만들고 이 파일에서 외부 앱에 공개할 경로를 지정

• 외부에 공유할 경로 설정

```
<paths xmlns:android="http://schemas.android.com/apk/res/android">
    <external-path name="myfiles"
        path="Android/data/com.example.test17/files/Pictures" />
</paths>
```

- XML 파일 정보를 매니페스트에 <provider> 태그로 등록

• 매니페스트에 파일 프로바이더 등록

```
<provider
    android:name="androidx.core.content.FileProvider"
    android:authorities="com.example.test17.fileprovider"
    android:exported="false"
    android:grantUriPermissions="true">
    <meta-data
        android:name="android.support.FILE_PROVIDER_PATHS"
        android:resource="@xml/file_paths"></meta-data>
</provider>
```

# 17-2 파일에 보관하기

- 공용 저장소 이용
  - 공용 저장소는 안드로이드 시스템에서 파일 종류에 따라 지정한 폴더
  - Uri값을 지정할 때 MediaStore.Images는 안드로이드폰의 이미지 파일이 저장되는 공용 저장소인 DCIM과 Pictures 디렉터리
  - MediaStore.Video는 DCIM, Movies, Pictures 디렉터리
  - Media Store.Audio는 Alarms, Audiobooks, Music, Notifications, Podcasts, Ringtones 디렉터리

• 공용 저장소에 접근

```
val projection = arrayOf(
    MediaStore.Images.Media._ID,
    MediaStore.Images.Media.DISPLAY_NAME
)
val cursor = contentResolver.query(
    MediaStore.Images.Media.EXTERNAL_CONTENT_URI,
    projection,
    null,
    null,
    null
)
cursor?.let {
    while (cursor.moveToNext()) {
        Log.d("kkang", "_id : ${cursor.getLong(0)}, name : ${cursor.getString(1)}")
    }
}
```

# 17-2 파일에 보관하기

- 이미지 데이터를 가져와 화면에 출력하는 코드

• 이미지 파일의 Uri값 가져오기

```
val contentUri: Uri = ContentUris.withAppendedId(
    MediaStore.Images.Media.EXTERNAL_CONTENT_URI,
    cursor.getLong(0)
)
```

- Uri값으로 이미지를 읽을 수 있는 InputStream 객체

• 이미지 데이터 가져오기

```
val resolver = applicationContext.contentResolver
resolver.openInputStream(contentUri).use { stream ->
    // stream 객체에서 작업 수행
    val option = BitmapFactory.Options()
    option.inSampleSize = 10
    val bitmap = BitmapFactory.decodeStream(stream, null, option)
    binding.resultImageView.setImageBitmap(bitmap)
}
```

# 17-3 공유된 프리퍼런스에 보관하기

## 공유된 프리퍼런스 이용하기

- 공유된 프리퍼런스는 플랫폼 API에서 제공하는 클래스로, 데이터를 키-값 형태로 저장
- SharedPreferences 객체를 얻는 방법은 다음 2가지를 제공
  - Activity.getPreferences(int mode)
  - Context.getSharedPreferences(String name, int mode)
- Activity.getPreferences() 함수는 액티비티 단위로 데이터를 저장할 때 사용

• 액티비티의 데이터 저장

```
val sharedPref = getPreferences(Context.MODE_PRIVATE)
```

- 앱 전체의 데이터를 키-값 형태로 저장하려고 SharedPreferences 객체를 얻을 때는 Context.getSharedPreferences() 함수를 이용

• 앱 전체의 데이터 저장

```
val sharedPref = getSharedPreferences("my_prefs", Context.MODE_PRIVATE)
```

# 17-3 공유된 프리퍼런스에 보관하기

- 데이터를 저장하려면 다음과 같은 SharedPreferences.Editor 클래스의 함수를 이용
  - putBoolean(String key, boolean value)
  - putInt(String key, int value)
  - putFloat(String key, float value)
  - putLong(String key, long value)
  - putString(String key, String value)

• 프리퍼런스에 데이터 저장

```
sharedPref.edit().run {
    putString("data1", "hello")
    putInt("data2", 10)
    commit()
}
```

# 17-3 공유된 프리퍼런스에 보관하기

- 저장된 데이터를 가져오려면 SharedPreferences의 게터 함수를 이용
  - getBoolean(String key, boolean defValue)
  - getFloat(String key, float defValue)
  - getInt(String key, int defValue)
  - getLong(String key, long defValue)
  - getString(String key, String defValue)

• 프리퍼런스에서 데이터 가져오기

```
val data1 = sharedPref.getString("data1", "world")
val data2 = sharedPref.getInt("data2", 10)
```

# 17-3 공유된 프리퍼런스에 보관하기

## 앱 설정 화면 만들기

- 플랫폼 API에서 이처럼 앱의 설정 기능을 자동화해주는 API는 많았지만 안드로이드 10 버전(API 레벨 29)부터 모두 deprecated
- AndroidX의 Preference를 이용할 것을 권장

• AndroidX의 프리퍼런스 사용 선언

```
implementation 'androidx.preference:preference-ktx:1.1.1'
```

# 17-3 공유된 프리퍼런스에 보관하기

- 프리퍼런스 이용 방법
  - res/xml 디렉터리에 설정과 관련된 XML 파일을 만들어야 합니다.
  - 루트 태그가 <PreferenceScreen>
  - 하위에 <SwitchPreferenceCompat>, <Preference> 등의 태그를 이용해 설정 항목을 준비

• 설정 XML 파일

```
<PreferenceScreen xmlns:app="http://schemas.android.com/apk/res-auto">
    <SwitchPreferenceCompat
        app:key="notifications"
        app:title="Enable message notifications" />
    <Preference
        app:key="feedback"
        app:title="Send feedback"
        app:summary="Report technical issues or suggest new features" />
</PreferenceScreen>
```

# 17-3 공유된 프리퍼런스에 보관하기

- 설정 항목의 key 속성값이 데이터의 키
- title 속성은 설정 화면에 출력되는 문자열
- XML 파일을 코드에서 적용해야 하는데 이때 PreferenceFragmentCompat 클래스를 이용
- PreferenceFragmentCompat을 상속받은 프래그먼트 클래스는 onCreatePreferences() 함수를 재정의해서 작성

• 설정 XML 파일 적용

```
class MySettingFragment : PreferenceFragmentCompat() {
    override fun onCreatePreferences(savedInstanceState: Bundle?, rootKey: String?) {
        setPreferencesFromResource(R.xml.settings, rootKey)
    }
}
```

• 액티비티에서 프래그먼트 출력

```
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    class="com.example.test17.MySettingFragment" />
```

▶ 실행 결과

# 17-3 공유된 프리퍼런스에 보관하기

- 설정 화면 구성
  - <PreferenceCategory> 태그를 이용하면 한 화면에 보이는 항목끼리 구분 지어 출력

• 항목끼리 묶기

```
<PreferenceScreen xmlns:app="http://schemas.android.com/apk/res-auto">
    <PreferenceCategory
        app:key="a_category"
        app:title="A Setting">
        <SwitchPreferenceCompat
            app:key="a1"
            app:title="A - 1 Setting" />
        <SwitchPreferenceCompat
            app:key="a2"
            app:title="A - 2 Setting" />
    </PreferenceCategory>
    <PreferenceCategory
        app:key="B_category"
        app:title="B Setting">
        <SwitchPreferenceCompat
            app:key="b1"
            app:title="B - 1 Setting" />
    </PreferenceCategory>
</PreferenceScreen>
```

▶ 실행 결과

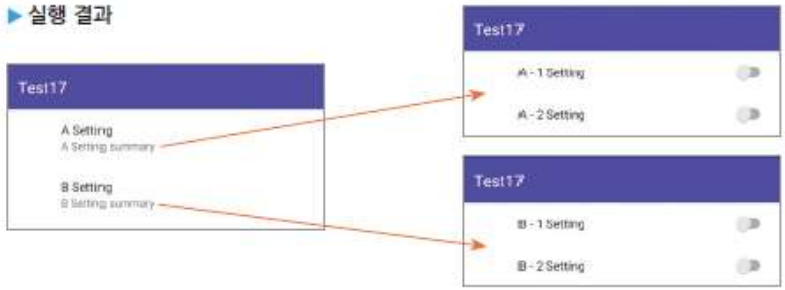
# 17-3 공유된 프리퍼런스에 보관하기

- 설정 항목이 더 많을 때는 화면을 여러 개로 분리하는 방법
- XML에서 각 설정 화면은 <Preference> 태그로 지정

• 두 화면을 포함하는 설정 화면

```
<PreferenceScreen xmlns:app="http://schemas.android.com/apk/res-auto">
    <Preference
        app:key="a"
        app:summary="A Setting summary"
        app:title="A Setting"
        app:fragment="com.example.test17.ASettingFragment" />
    <Preference
        app:key="b"
        app:summary="B Setting summary"
        app:title="B Setting"
        app:fragment="com.example.test17.BSettingFragment" />
</PreferenceScreen>
```

▶ 실행 결과



# 17-3 공유된 프리퍼런스에 보관하기

- <Preference> 태그를 이용해 설정 화면을 분할했다면 액티비티에서 PreferenceFragmentCompat.OnPreferenceStartFragmentCallback 인터페이스를 구현하고 onPreferenceStart Fragment() 함수를 재정의해서 작성
- onPreferenceStartFragment()는 설정 화면이 바뀔 때마다 호출되는 함수

• 분할 설정 화면을 보여주는 액티비티 코드

```
class SettingActivity : AppCompatActivity(),
    PreferenceFragmentCompat.OnPreferenceStartFragmentCallback {
    (... 생략 ...)
    override fun onPreferenceStartFragment(caller: PreferenceFragmentCompat,
                                           pref: Preference): Boolean {
        // 새로운 프래그먼트 인스턴스화
        val args = pref.extras
        val fragment = supportFragmentManager.fragmentFactory.instantiate(
            classLoader,
            pref.fragment)
        fragment.arguments = args
        supportFragmentManager.beginTransaction()
            .replace(R.id.setting_content, fragment)
            .addToBackStack(null)
            .commit()
        return true
    }
}
```

# 17-3 공유된 프리퍼런스에 보관하기

- 메인 설정 화면에서 인텐트를 이용해 하위 설정 화면을 띄우는 방법

• 인텐트로 설정 화면 실행

```
<Preference
    app:key="activity"
    app:title="Launch activity">
    <intent
        android:targetClass="com.example.test17.SomeActivity"
        android:targetPackage="com.example.test17" />
</Preference>
```

# 17-3 공유된 프리퍼런스에 보관하기

- <intent> 태그 하위에 <extra> 태그로 인텐트에 포함해서 전달할 엑스트라 데이터를 설정
- 명시적 인텐트 정보뿐만 아니라 암시적 인텐트 정보도 설정할 수 있습니다.

• 인텐트에 엑스트라 데이터 포함

```
<intent
    android:targetClass="com.example.test17.SomeActivity"
    android:targetPackage="com.example.test17">
    <extra
        android:name="example_key"
        android:value="example_value" />
</intent>
```

• 암시적 인텐트 사용

```
<intent
    android:action="android.intent.action.VIEW"
    android:data="http://www.google.com" />
```

# 17-3 공유된 프리퍼런스에 보관하기

- 설정 제어
  - 사용자가 설정 항목을 클릭한 순간의 이벤트를 처리하거나 설정값을 설정 항목 옆에 나타나게 하는 방법
  - 설정 항목에 해당하는 객체를 findPreference() 함수로 얻어야 합니다.

• 글을 입력받는 설정

```
<EditTextPreference
    app:key="id"
    app:title="ID 설정"
    app:isPreferenceVisible="false" />
```

• 설정값을 코드에서 바꾸기

```
override fun onCreatePreferences(savedInstanceState: Bundle?, rootKey: String?) {
    setPreferencesFromResource(R.xml.settings, rootKey)
    val idPreference: EditTextPreference? = findPreference("id")
    idPreference?.isVisible = true

    idPreference?.summary="code summary"
    idPreference?.title="code title"
}
```

# 17-3 공유된 프리퍼런스에 보관하기

- <EditTextPreference>나 제시한 목록에서 선택하는 <ListPreference>는 설정값을 summary 속성에 자동으로 지정
- SimpleSummaryProvider를 사용

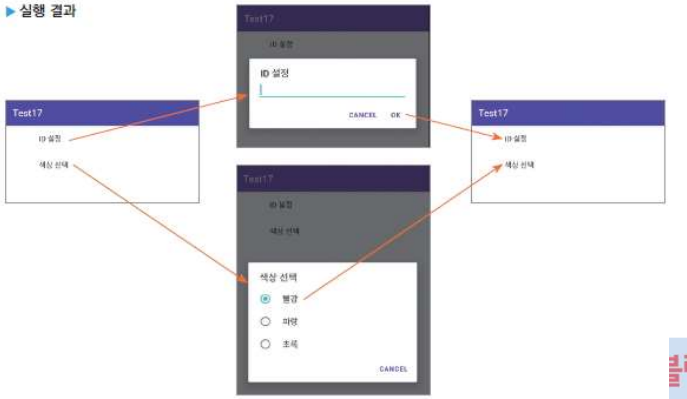
• 설정 XML 예

```
<EditTextPreference
    app:key="id"
    app:title="ID 설정"/>

<ListPreference
    app:key="color"
    app:title="색상 선택"
    android:entries="@array/my_color"
    app:entryValues="@array/my_color_values" />
```

• 설정값 자동 적용

```
val idPreference: EditTextPreference? = findPreference("id")
val colorPreference: ListPreference? = findPreference("color")

idPreference?.summaryProvider =
    EditTextPreference.SimpleSummaryProvider.getInstance()
colorPreference?.summaryProvider =
    ListPreference.SimpleSummaryProvider.getInstance()
```

▶ 실행 결과



# 17-3 공유된 프리퍼런스에 보관하기

- SummaryProvider의 하위 클래스를 만들어 코드에서 원하는 대로 summary가 지정되게 할 수도 있습니다.

• 코드에서 설정값 표시하기

```
idPreference?.summaryProvider =
    Preference.SummaryProvider<EditTextPreference> { preference ->
        val text = preference.text
        if (TextUtils.isEmpty(text)) {
            "설정이 되지 않았습니다. "
        } else {
            "설정된 ID 값은 : $text 입니다."
        }
    }
```

▶ 실행 결과

# 17-3 공유된 프리퍼런스에 보관하기

- 설정 항목에 이벤트를 추가

• 이벤트 핸들러 지정

```
idPreference?.setOnPreferenceClickListener { preference ->
    Log.d("kkang", "preference key : ${preference.key}")
    true
}
```

# 17-3 공유된 프리퍼런스에 보관하기

- 설정한 값 가져오기
  - 설정값을 가져올 때는 PreferenceManager.getDefaultSharedPreferences() 함수를 이용

• 설정값 가져오기

```
val sharedPreferences = PreferenceManager.getDefaultSharedPreferences(activity)
val id = sharedPreferences.getString("id", "")
```

# 17-3 공유된 프리퍼런스에 보관하기

- 설정 변경 순간 감지
  - Preference.OnPreference ChangeListener를 이용하는 방법 은 프리퍼런스 객체마다 이벤트 핸들러를 직접 지정하여 객체의 설정 내용이 변경되는 순간의 이벤트를 처리
  - SharedPreferences.OnSharedPrefe renceChangeListener를 이용하는 방법은 설정 객체의 변경을 하나의 이벤트 핸들러에서 처리

• 프리퍼런스를 이용한 이벤트 처리

```
idPreference?.setOnPreferenceChangeListener { preference, newValue ->
    Log.d("kkang", "preference key : ${preference.key}, newValue : $newValue")
    true
}
```

# 17-3 공유된 프리퍼런스에 보관하기

- SharedPreferences.OnSharedPreferenceChangeListener를 이용

• 공유된 프리퍼런스를 이용한 이벤트 처리

```
class MySettingFragment : PreferenceFragmentCompat(),
SharedPreferences.OnSharedPreferenceChangeListener  {
    (... 생략 ...)
    override fun onSharedPreferenceChanged(sharedPreferences:
                                           SharedPreferences?, key: String?) {
        if (key == "id") {
            Log.i("kkang", "newValue : " + sharedPreferences?.getString("id", ""))
        }
    }
    override fun onResume() {
        super.onResume()
        preferenceManager.sharedPreferences
            .registerOnSharedPreferenceChangeListener(this)
    }
    override fun onPause() {
        super.onPause()
        preferenceManager.sharedPreferences
            .unregisterOnSharedPreferenceChangeListener(this)
    }
}
```

# Do it! 실습 개선된 할 일 목록 앱 만들기

## 1단계. 모듈 생성하고 빌드 그래들 작성하기

- Ch17_Database 라는 이름으로 새로운 모듈을 만듭니다.
- 뷰 바인딩 기법을 이용하도록 설정
- AndroidX의 Preference 라이브러리를 추가

## 2단계. 액티비티를 생성하고 파일 복사하기

- AddActivity, SettingActivity라는 이름으로 빈 액티비티를 추가
- res 디렉터리의 drawable, layout, menu, values, xml 디렉터리와 코틀린 소스 파일이 있는 디렉터리의 AddActivity.kt, Main Activity.kt, MyAdapter.kt, SettingActivity.kt 파일을 현재 프로젝트에서 같은 위치에 덮어쓰기 합니다.

Do it! 실습

# 시계 앱의 스톱워치 기능 만들기

## 3단계. 설정 XML 파일 작성하기

- settings.xml 파일을 열고 앱 설정 내용을 추가

## 4단계. 설정 프래그먼트 작성하기

- MySettingFragment라는 이름의 프래그먼트를 새로 만들고 내용으로 대체

## 5단계. 설정 화면의 XML 작성하기

- activity_setting.xml 파일을 열고 화면에 출력할 프래그먼트 클래스를 작성

Do it! 실습

# 시계 앱의 스톱워치 기능 만들기

## 6단계. 설정 XML 파일 작성하기

- DBHelper라는 이름으로 새로운 코틀린 클래스 파일을 만들고 작성

## 7단계. 할 일을 저장하는 액티비티 작성하기

- AddActivity.kt 파일을 열고 작성

## 8단계. 메인 액티비티 작성하기

- MainActivity.kt 파일을 열고 작성

Do it! 실습

# 시계 앱의 스톱워치 기능 만들기

## 9단계. 앱 실행하기

# 감사합니다

단단히 마음먹고 떠난 사람은
산꼭대기에 도착할 수 있다.
산은 올라가는 사람에게만 정복된다.

윌리엄 셰익스피어
William Shakespeare

이지스퍼블리싱(주)