SHOUT · OUR · PASSION · TOGETHER

ODo IT SOPT O

안드로이드 4차 세미나

SHOUT OUR PASSION TOGETHER
SOPT

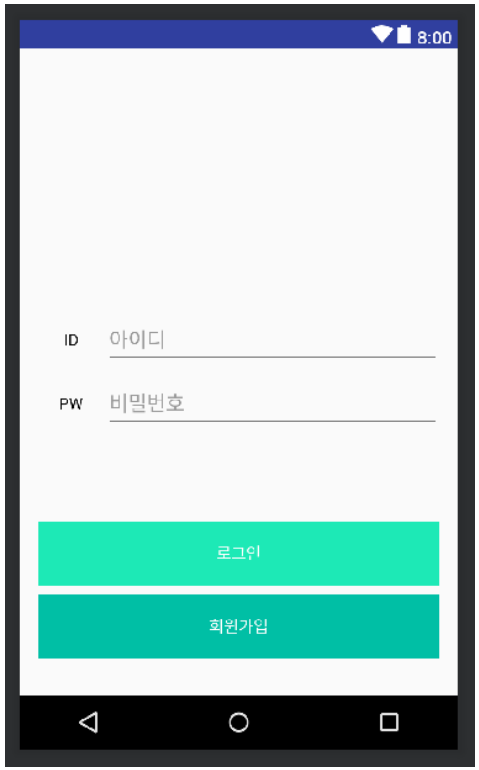
Contents

01

실습 사전 준비

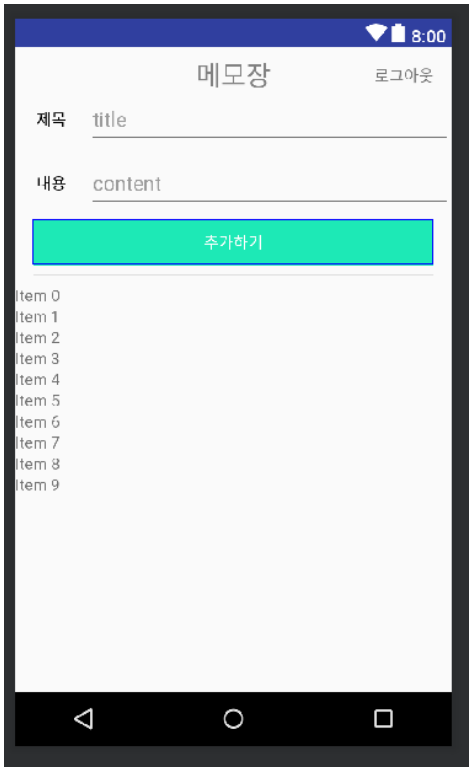
오늘 할 세미나 내용 실습에 의의를 두고 시간 절약을 위해

아래와 같이 투박한.. View를 이용해서 실습해봅시다!!! (2개의 .kt과 .xml 필요)



LoginActivity.kt

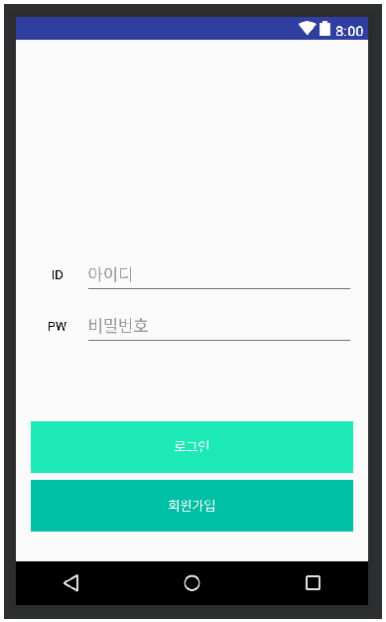
activity_login.xml



MemoActivity.kt

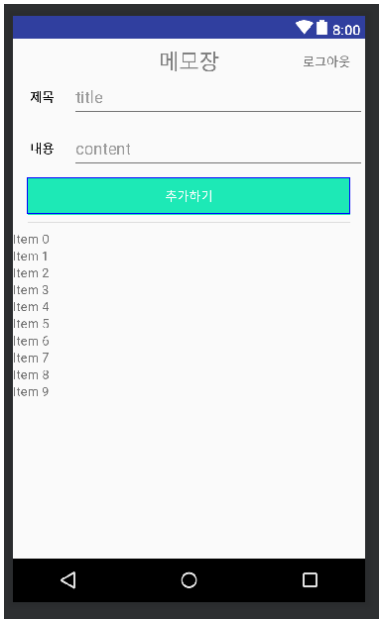
activity_memo.xml

1. LoginActivity와 MemoActivity를 만들어주세요!(메뉴를 통해서 빈 엑티비티로)



LoginActivity.kt

activity_login.xml



MemoActivity.kt

activity_memo.xml

2. 뷰는 단톡방에 올려드리겠습니다! 복붙을 통해 해당 xml파일에 붙여넣기를 해주세요!

02

내부 데이터베이스 소개

02 내부 데이터베이스 소개

- **SharedPreference**
  - ① 간단한 데이터를 읽고 쓰기에 유용
  - ② Key-Value 쌍으로 데이터 저장
  - ③ context.getSharedPreference(키이름, 모드)을 통해, 단일 인스턴스에 접근
  - ④ 앱 초기 설정 값(KakaoTalk로 예를 들면 알림 끄기 같은(?)), 자동 로그인 구현 등에 이용될 수 있음
  - ⑤ 간단하게 사용하기 쉽다.

- **SQLite**
  - ① Android OS에서는 기본적으로 SQLite라는 데이터베이스를 제공, 가장 많이 쓰긴합니다!
  - ② 일반적인 RDBMS 성능보다는 낮지만, 중소 규모의 데이터 양이라면 속도 괜찮다!
  - ③ MySQL등 RDBMS를 써본 적이 있다면 익히는데 쉽다.

- **Realm**
  - ① 모바일에 최적화된 내부 데이터베이스 라이브러리!
  - ② 쿼리를 이용해 테이블의 칼럼에 값을 저장하는 SQLite와 달리, 데이터를 객체의 형태로 저장
  - ③ ORM(Object-relation mapping)방식이 아닌 데이터 컨테이너 모델을 사용해 객체를 직접 데이터베이스에 저장!
    ➔ 그래서 빠른 속도를 낸다!!
  - ④ SQLite보다 속도가 빠르며 사용하기에도 편해서 각광 받는다!

03

SharedPreference

(실습) SharedPreference를 이용한 자동 로그인 구현

1. “db” 패키지를 만들고 내부에 SharedPreferenceController.kr 만들기

2. object : JAVA로 치면 static 객체를 만들 때 사용

```
object SharedPreferenceController {

}
```

3. USER ID와 USER PW를 저장하는 코드 get/set

```
import android.content.Context
import android.content.SharedPreferences

object SharedPreferenceController {
    private val USER_NAME: String = "user_name"

    private val USER_ID: String = "user_id"
    private val USER_PW: String = "user_pw"

    //ID 집어 넣기
    fun setUserID(ctx: Context, input_id: String) {
        val preference: SharedPreferences = ctx.getSharedPreferences(USER_NAME, Context.MODE_PRIVATE)
        val editor: SharedPreferences.Editor = preference.edit()
        editor.putString(USER_ID, input_id)
        editor.commit()
    }

    //PW 집어 넣기
    fun setUserPW(ctx: Context, input_pw: String) {
        val preference: SharedPreferences = ctx.getSharedPreferences(USER_NAME, Context.MODE_PRIVATE)
        val editor: SharedPreferences.Editor = preference.edit()
        editor.putString(USER_PW, input_pw)
        editor.commit()
    }

    //ID 꺼내기
    fun getUserID(ctx: Context): String {
        val preference: SharedPreferences = ctx.getSharedPreferences(USER_NAME, Context.MODE_PRIVATE)
        return preference.getString(USER_ID, "") // (키 명, 든게 없을때 리턴할 값)
    }

    //PW 꺼내기
    fun getUserPW(ctx: Context): String {
        val preference: SharedPreferences = ctx.getSharedPreferences(USER_NAME, Context.MODE_PRIVATE)
        return preference.getString(USER_PW, "") // (키 명, 든게 없을때 리턴할 값)
    }

    //user_name 모든 데이터 제거
    fun clearUserSharedPreferences(ctx: Context) {
        val preference: SharedPreferences = ctx.getSharedPreferences(USER_NAME, Context.MODE_PRIVATE)
        val editor: SharedPreferences.Editor = preference.edit()
        editor.clear()
        editor.commit()
    }
}
```

Context.MODE_PRIVATE를 mode에 써주면, 해당 어플리케이션 외 다른 곳에서는 이 데이터를 접근 할 수 없다.

이 함수를 쓰면 “user_name”에 저장된 모든 데이터를 지울 수 있습니다!

상세 설명..! 이라고 하기는 단순한 로직!

```
    //ID 집어 넣기
    fun setUserID(ctx: Context, input_id: String) {
        val preference: SharedPreferences = ctx.getSharedPreferences(USER_NAME, Context.MODE_PRIVATE)
        val editor: SharedPreferences.Editor = preference.edit()
        editor.putString(USER_ID, input_id)
        editor.commit()
    }



    //ID 꺼내기
    fun getUserID(ctx: Context): String {
        val preference: SharedPreferences = ctx.getSharedPreferences(USER_NAME, Context.MODE_PRIVATE)
        return preference.getString(USER_ID, "") // (키 명, 든게 없을때 리턴할 값)
    }
}
```

- **값을 넣을 때(Setter)**
  1. getSharedPreference를 통해 앱에 존재하는 단일 인스턴스를 가져옵니다!
  2. editor을 엽니다!
  3. 값을 넣습니다
  4. commit을 통해 마침을 알립니다!

- **값을 꺼낼 때(Getter)**
  1. getSharedPreference를 통해 앱에 존재하는 단일 인스턴스를 가져옵니다!
  2. **키**를 통해꺼냅니다.
  3. 끝!

4. 이제 SharedPreference를 사용해보자!

- 우리는 현재 서버가 없으므로, 그냥 ID와 PW에 아무 값만 넣으면 로그인 되도록 하겠습니다!

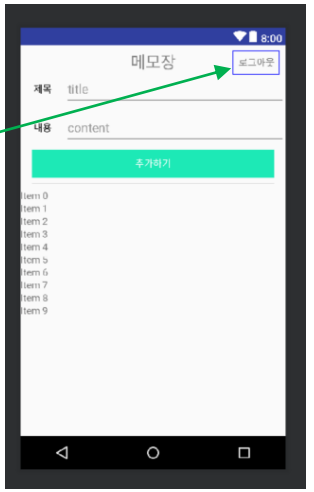
```
class LoginActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_login)
        setOnBtnClickListener()

        if (SharedPreferenceController.getUserID(this).isNotEmpty()){
            startActivity<MemoActivity>()
            finish()
        }
    }
    private fun setOnBtnClickListener(){
        //로그인 버튼
        btn_login_act_sign_in.setOnClickListener {
            val input_id : String = et_login_act_id.text.toString()
            val input_pw : String = et_login_act_pw.text.toString()

            //만약 서버가 존재한다면 여기서 로그인 성공 여부를 체크하겠죠?!
            //지금은 서버가 없으므로 공백인지 아닌지만 체크해줍니다!
            if (input_id.isNotEmpty() && input_pw.isNotEmpty()){

                SharedPreferenceController.setUserID(this, input_id)
                SharedPreferenceController.setUserPW(this, input_pw)

                startActivity<MemoActivity>()
                finish()
            }
        }
    }
}
```

2. 이후 앱을 껐다 켰을 때, 만약 저장된 ID가 있다면 로그인 과정 없이 바로 MemoActivity로 넘어가는 로직을 넣었습니다!

1. 서버가 없으므로!!
ID와 PW가 일치하는지 체크 과정 없이 공백만 아니라면 무조건 로그인 성공으로 가정하고! SharedPreference에 값을 저장 시킵니다!!!

<LoginActivity.kt>

5. MemoActivity에 로그아웃 하는 로직을 넣어 보아요!




```kotlin
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_memo)

    btn_memo_act_logout.setOnClickListener {
        SharedPreferenceController.clearUserSharedPreferences(this)
        startActivity<LoginActivity>()
        finish()
    }

}
```

<MemoActivity.kt>

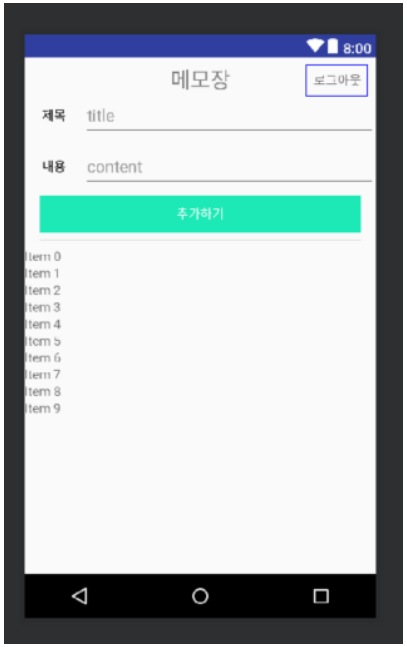
- clearUserSharedPreferences를 통해 로그인된 정보 데이터를 제거하고 다시 로그인 시키도록 하는 로직을 넣습니다!

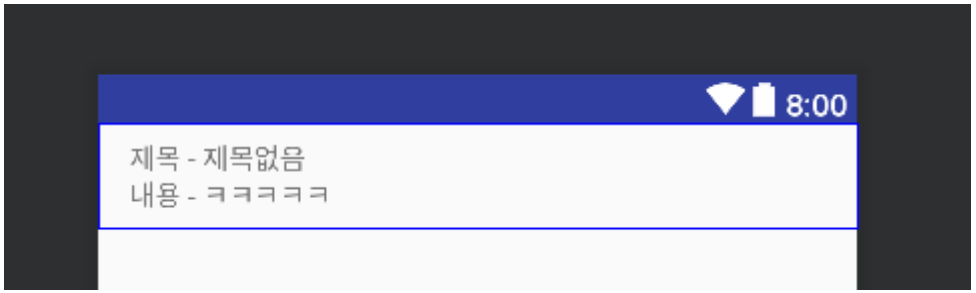
04

RecyclerView 더 알아보자

1. (복습) MemoActivity에 RecyclerView 구성해보기

① DataClass 만들기
② Adapter 만들기
③ item view 만들기
④ MemoActivity에 RecyclerView 연결 시켜 보기





뷰가 투박해도 괜찮습니다! item view을 스스로 만들어보세요.

2. RecyclerView의 아이템을 변동시키는 실습을 해보겠습니다! 일단 데이터 추가부터!!

```
private fun addItem(){
    if (et_memo_act_title.text.toString().isNotEmpty() && et_memo_act_content.text.toString().isNotEmpty()){
        val position = recycMemoRecyclerViewAdapter.itemCount
        recycMemoRecyclerViewAdapter.dataList.add(MemoData(et_memo_act_title.text.toString(), et_memo_act_content.text.toString()))
        //recycMemoRecyclerViewAdapter.notifyDataSetChanged()
        recycMemoRecyclerViewAdapter.notifyItemInserted(position)
    }
}
```

<MemoActivity.kr>

아이템 변경 사항 알리기

- notifyDataSetChanged() : 데이터 전체가 바뀌었을 때 호출해주자!
  (이거 하나면 다른 메소드 신경 안 써도 되긴 하는데…! 불필요한 Cost 방지를 위해! 최대한 아끼기)
- notifyItemInserted() : 특정 데이터 하나가 들어갔을 때 호출하면 좋은 것!
- notifyItemRangeInserted() : 특정 범위 포지션에 데이터를 추가했을 때 호출하면 좋은 것!
  예를 들면, 5~8번 포지션에 새로운 데이터를 넣은 경우!
- notifyRemoved() : 특정 데이터 하나를 없앴을 때 호출하면 좋은 것!
- notifyRangeRemoved() : 특정 범위 포지션에 데이터를 없앴을 때 호출하면 좋은 것!
- notifyItemMoved() : 특정 아이템의 위치를 바꿀 때, 3번 포지션 → 5번 포지션
- notifyItemRangeChanged() : 특정 범위 포지션이 바뀌었을 때

3. 다음은 아이템 클릭 시, 제거되는 로직을 구현해봅시다!

- 일단, item view의 가장 root group view에 id를 줍니다!!

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/btn_rv_item_memo_whole_box">

            <!== 아이템 뷰 내용들~~~~~ ==>
</RelativeLayout>
```

이런 식으로!!!

- 다음 해당 RecyclerView Adapter Holder에서 findViewId를 해주고!

```
val whole_box : RelativeLayout = itemView.findViewById(R.id.btn_rv_item_memo_whole_box) as RelativeLayout
```

- onBindViewHolder에서 item root view에 클릭 리스너를 달아주세요!

```
holder.whole_box.setOnClickListener {
    try {
        dataList.removeAt(position)
        notifyItemRemoved(position)
        notifyItemRangeChanged(position, dataList.size)
    } catch (e : IndexOutOfBoundsException){
        Log.e("Index error", e.toString())
    }
}
```

TMI) 클릭 시 Ripple Effect 주기, 클릭할 맛 나도록!

```
android:background="?android:selectableItemBackground"
```

- 클릭 가능한 view에 위의 옵션을 줘보세요!!! 클릭 시 물방울 모양으로 퍼져요!
- 위 옵션을 이미 정의되어 있는 것이고,
  커스터마이징이 가능하니 응용하고 싶은 사람은 검색을 통해 더 알아보는 걸로!!!

- 예를 들면 이런 식으로!

```xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/btn_rv_item_memo_whole_box"
    android:background="?android:selectableItemBackground">

    <!== 뷰 내용들~~~ ==>
</RelativeLayout>
```

4. 중첩 스크롤에 대해 알아봅시다!

- NestedScrollView를 통해 RecyclerView와 그 외 View가 함께 스크롤 되도록 만들어 보겠습니다.

함께 스크롤 될 View!!
RecyclerView의 위에 있으므로
Header라고 보면 되겠습니다!

RecyclverView

```
<android.support.v4.widget.NestedScrollView
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <LinearLayout
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <LinearLayout...>
        <android.support.v7.widget.RecyclerView...>

    </LinearLayout>
</android.support.v4.widget.NestedScrollView>
```

<activity_memo.xml>

(NestScrollView 속을 LinearLayout로 감싸고, 그 안에 함께 스크롤 될 것들을 모두 넣어주세요)

- 여기서 끝이 아닙니다!! 부드러운 스크롤을 구현하기 위해서 아래와 같은 옵션을 추가해주세요!

```
android:nestedScrollingEnabled="false"
```

```
<android.support.v7.widget.RecyclerView
    android:id="@+id/rv_memo_act_memo_list"
    android:layout_width="match_parent"
    android:layout_height="match_parent“
    android:nestedScrollingEnabled="false">

</android.support.v7.widget.RecyclerView>
```

5. 간단하게 당겨서 새로 고침 기능을 구현해 봅시다!

- 앞서 만든 NestedScrollView를SwipeRefreshLayout으로 감싸고 id를 셋팅해주세요!

```
<android.support.v4.widget.SwipeRefreshLayout
    android:id="@+id/refresh_memo_act"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <android.support.v4.widget.NestedScrollView...>

</android.support.v4.widget.SwipeRefreshLayout>
```

<activity_memo.xml>

- 그 다음, 아래와 같은 리스너를 셋팅해주세요!! 그럼 끄읏!~!

```
refresh_memo_act.setOnRefreshListener {
    toast("새로 고침!")
    //이곳에 서버 통신과 같은 로직을 구현해주시면 됩니다!!
    refresh_memo_act.isRefreshing = false // 뺑뺑이 멈추기!
}
```

<MemoActivity.kr>

SHOUT · OUR · PASSION · TOGETHER

ODo IT SOPT O

THANK U

SHOUT OUR PASSION TOGETHER
SOPT

PPT 자료 안드로이드 남윤환