13 (13.3) Preferences KangTeacher's 방금 전

https://blog.naver.com/jyseolois/222822862607

● SharedPreferences: 앱의 데이터를 영속적으로 저장할 때 사용하는 클래스. 데이터를 테이블 구조로 저장하여 영속화하는 DBMS 방식과는 달리, 간단하게 키-값(key-value) 형태로 저장한다. SharedPreferences로 저장한 데이터 역시 XML 파일로 저장되지만, 코드를 개발자가 직접 작성하지 않고 SharedPreferences 객체를 이용한다.

▶ SharedPreferences 객체 획득: SharedPreferences를 이용하려면, 다음과 같은 함수로 SharedPreference 객체를 먼저 획득해야 한다.

Activity.getPrefernece(int mode)	별도로 파일명을 지정하지 않고 자동으로 액티비티 이름의 파일을 찾아 저장한다. 예를 들면 MainActivity에서 이 함수를 이용해서 SharedPreferences를 획득하면 자동으로 MainActivity.xml 파일에 저장한다. 하나의 액티비티만을 위한 저장 공간이 생성되는 것이며 다든 액티비티에서는 해당 데이터를 이용할 수 없다.	SharedPreferences sharedPref = getPreferences(Context.MODE_PRIVATE);
Context.getSharedPreferences (String name, int mode)	앱 전체에서 이용할 수 있는 SharedPreferences 객체를 얻을 때 사용된다. 첫 번째 매개변수에 지정한 이름의 파일에 데이터가 저장된다.	SharedPrefernece sharedPref2 = getSharedPreferences("my_prefs", Context.MODE_PRIVATE);

▶ SharedPreferences 데이터 저장: SharedPrefernece로 데이터를 저장하려면 Editor 클래스의 putter 함수를 이용하여 키-값 형태로 데이터를 저장한다. 각 데이터 타입을 위한 함수가 제공된다.

putBoolean(String key, boolean value)
putFloat(String key, float value)
putInt(String key, int value)
putLong(String key, long value)
putString(String key, String value)

이 함수를 이용하여 저장한 데이터를 최종 반영하기 위해 commit() 함수를 호출한다.

```
SharedPreferences.Editor editor = sharedPref.edit();
editor.putString("data1", "hello");
editor.putInt("data2", 100);
editor.commit();
```

▶ SharedPreference을 통해 저장된 데이터를 획득하기: SharedPreferences 클래스의 getter 함수를 이용한다.

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)

```
String data1 = sharedPref.getString("data1", "none");
int data2 = sharedPref.getInt("data2", 0);
```

● 앱 설정 자동화: 사용자의 사생활을 위해 앱의 환경설정을 위한 액티비티에서 설정 화면을 구성하고, 그 화면에서 발생하는 다양한 사용자 이벤트를 처리하여 데이터를 영속적으로 저장해야 한다. 이 과정을 대행해 주는 것이 앱 설정 자동화이다. androidx의 Preference는 앱에서 설정 기능을 제공하도록 하는 제트팩의 API이다. (DBMS 방식은 여러 건의 데이터를 구조적으로 저장하기에 좋은 바식이다. 대표적인 것이 주소록 앱으로, 많은 데이터가 저장되고 한 사람을 대상으로 여러 데이터가 저장되다 보니, 테이블을 만들어서 DBMS 방식으로 저장하는 게 좋다. 그런데 앱 내에서 간단한 데이터를 저장해야 하는 경우가 있다. 예를 들어 카카오톡이 왔을 때 진동을 울리게 설정할 수 있다. 진동이 울리게 설정했따면 그 값이 저장되어 계속 진동이 울려야 한다. 이 값은 true or false 데이터인데, 이럴 경우 이 데이터 하나를 저장하자고 테이블을 만들어서 행 하나 열 하나를 만들기보다는 SharedPreference을 이용해 키-값 형태로 저장해서 이용하는 게 낫다. 대표적으로 앱 설정 자동화 기법이 사용되는 예이다).

▶ build.gardle 파일에 dependencies 선언하기: androidx의 Preference를 이용하려면 build.gardle 파일에 다음과 같은 라이브러리를 dependencies로 선언해야 한다.

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

▶ 설정 화면을 위한 XML 파일 만들기: res 하위에 xml이라는 폴더를 만든다. 그리고 설정 XML 파일에서 다음과 같은 태그로 설정 화면을 구성한다.

<PreferenceScreen>	설정 XML의 root 태그
<PreferenceCategory>	설정 여러 개를 시각적으로 묶어서 표현
<Preference>	설정 화면 중첩

<CheckboxPreference>	체크 박스가 나오는 설정
<EditTextPreference>	글 입력을 위한 설정
<ListPreference>	항목 다이얼로그를 위한 설정
<MultiSelectListPreference>	항목 다이얼로그인데 체크박스가 자동 추가되어 여러 선택 가능
<RingtonPreference>	알람음 선택을 위한 설정
<SwitchPreferenceCompat>	스위치를 이용한 설정

☞ <CheckboxPreference>와 <SwitchPreferenceCompat>: 사용자에게 설정 응답을 boolean 값으로 전달받는 태그이다. 설정 내용 옆에 체크 박스나 스위치를 제공하여 사용자가 설정 상태를 바꿀 수 있게 한다. title 속성과 summary 속성이 화면에 출력되는 문자열이며, 상태를 변경하면 SharedPreferneces로 사용자 설정 내용을 자동으로 저장한다. 사용자가 체크박스나 스위치의 상태를 변경하면, 자동으로 SharedPreferences 객체에 true 혹은 false 값이 저장되는 것이다. 이때 키 값은 XML 속성 중 key에 지정한 값을 이용한다.

```
<PrefenceScreen xmlns:android:"http://schemas.android.com/apk/res/android">
    <CheckBoxPreference
        android:key="message"
        android:title="소리 알림"
        android:summary="알림을 소리로 받으려면 체크하세요"/>
    <SwitchPreferenceCompat
        android:key="vibrate"
        android:title="진동 알림"
        android:summary="알림을 진동 울림으로 받으려면 체크하세요."/>
</PreferenceScreen>
```

☞ <EditTextPreference>: 사용자에게 설정 응답을 간단한 글로 입력받는 태그. 대화 상자를 띄워 사용자에게 설정 응답 내용을 입력받아 자동으로 저장해 준다.

```
<EditTextPrefernece
    android:key="nickname"
    android:title="Nickname"
    android:summary="Nickname을 설정하세요"
    android:dialogTitle="Nickname 설정" />
```

☞ <ListPreference>와 <MultiSelectListPreference>: 목록을 띄워 사용자에게 설정 응답을 받는다. 목록 선택을 위한 대화상자가 자동으로 열린다. <ListPreference> 태그로 나타나는 목록에서는 하나의 목록만 선택되며, <MultiSelectListPreference> 태그로 생성한 목록에서는 여러 개를 선택할 수 있다. entries 속성으로 목록 선택 대화상자의 항목 문자열을 설정하고, entryValues 속성으로 선택한 항목을 위해 저장되어야 하는 값을 지정한다. 화면에 보이는 항목 문자열 그대로 저장되도록 하려면 entries 속성과 entryValues 속성에 같은 문자열을 주면 된다. entries 속성과 entryValues 속성의 값은 배열 리소스로 등록한다. 배열 리소스는 res/values 폴더를 이용하며, <string-array> 태그로 등록한다. R.array.array_voice를 이용하면 등록된 여러 문자열이 배열로 반환된다.

```
// res/values

<resources>
    <string-array name="array_voice">
        <item>카톡</item>
        <item>카톡왔숑</item>
        <item>카톡카톡</item>
        <item>카카오톡</item>
    </string-array>
</resources>



<ListPreference
    android:key="sound"
    android:title="알림음"
    android:summary="카톡"
    android:entries="@array/array_voice"
    android:entryValues="@array/array_voice" />
```

☞ <PreferenceCategory>: 화면에 나오는 여러 개의 설정을 하나로 묶어준다. 일종의 서브 타이틀을 생성해주는 태그다.

```
<PreferenceCategory android:title="디버깅">
    <SwtichPreferenceCompat
        android:defaultValue="false"
        android:key="debugging"
        android:title="USB 디버깅"
        android:summary="USB가 연결된 경우 디버그 모드 사용" />
    <CheckBoxPreference
        android:defaultvalue="false"
        android:dependency="debugging"
        android:key="use_app"
        android:title="USB를 통해 설치된 앱 확인"
        android:sumamry="ADB/ADT를 통해 설치된 앱의 유해한 동작이 있는지 확인" />
</PreferenceCategory>
```

생성된 설정창을 스마트폰으로 보면, 두 가지의 설정이 묶여져 있다. dependency 속성이 Preference의 결합 관계를 표현한다. dependency에 연결된 Preference가 true이면 해당 설정이 활성화되고, false면 비활성화된다. 위의 코드에서는 false이기 때문에 두 가지 설정이 구분된다. 의미 있는 결합된 설정을 생성할 때는 Preference가 모두 true여야 한다.

☞ <Preference>: 화면을 여러 개로 분리하여 더 많은 설정 항목을 생성할 수 있다. 설정 화면을 두 개로 분리하려면, 설정 XML 파일과 프래그먼트를 두 개씩 만들어야 한다. 그리고 각 설정 화면을 포함하는 메인 설정 XML을 작성한다. 메인 설정 XML파일에서 각각의 설정 화면이 <Preference> 태그로 지정된다. 그러면 메인 설정 화면에서 사용자가 화면을 클릭할 때 fragment 속성에 지정한 설정 화면으로 전환된다.

```
// 메인 설정 XML 파일
<PreferenceScreen xmlns:android:"http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">
    <Preference
        app:key="one"
        app:title="One Setting"
        app:fragment="com.example.test5_13.OneSettingFragment" />
    <Prefence
        app:key="two"
        app:title="Two Setting"
        app:fragment="com.example.test5_13.TwoSettingFragment" />
</PrefenceScreen>
```

그런 다음 액티비티에서 PreferenceFragmentCompat.OnPreferneceStartFragmentCallback 인터페이스를 구현한다. 그리고 onPreferenceStartFragment() 함수를 재정의한다. 이 함수를 정의하지 않아도 설정 화면이 분할되는 것에는 문제가 없지만 뒤로 가기 버든을 눌렀을 때 이전 화면이 출력되지 않게 된다.

```
public class SettingsActivity extends AppCompatActivity implements PreferenceFragmentCom
    // ....
    @Override
    public boolean onPreferenceStartFragment(PreferenceFragmentCompat caller, PReference
        Bundle args = pref.getExtras();
        Fragment fragment = getSupportFragmentManager().getFragmentFactory().instantiate
        fragment.setArguments(args);
        getSupportFragmentManager().beginTransaction().replace(R.id.setting_fragment, fr
        return true;
    }
}
```

복잡한 설정 화면을 구현할 때는 메인 설정 화면에서 인텐트를 이용해 하위 설정 화면을 띄우는 방법을 사용한다. 이 작업을 코드에서 할 수도 있지만 설정 XML 파일 등록만으로도 처리할 수 있다. 설정 화면의 항목을 사용자가 클릭했을 때 다른 액티비티를 실행하는 기능을 XML 파일에서 설정해주는 것만으로도 구현할 수 있는 것이다. <Prefernece> 태그 하위에 <intent> 태그로 설정 화면을 지정하여, 사용자가 이 항목을 클릭했을 때 해당 설정 화면이 실행되도록 한다.

```
<Prefernece
    app:key="activity"
    app:title="Launch activity">
    <intent
        android:targetClass="com.example.test5_13.MainActivity"
        androidtargetPackage="com.example.test5_13" />
</Preference>
```

이때 Extra 정보를 포함시킬 수도 있다. <intent> 태그 하위에서 <extra> 태그로 설정하여 전달할 Extra 데이터를 지정하면 된다.

```
<Prefernece
    app:key="activity"
    app:title="Launch activity">
    <intent
        android:targetClass="com.example.test5_13.MainActivity"
        androidtargetPackage="com.example.test5_13" />
        <extra
            android:name="example_key"
            android:value="example_value" />
</Preference>
```

위처럼 명시적 인텐트 정보뿐만 아니라 암시적 인텐트 정보도 설정할 수 있다.

```
<Prefernece
    app:key="activity"
    app:title="Launch activity">
    <intent
        android:action="android.intent.action.VIEW"
        android:data="http://www.google.com"/>
</Preference>
```

▶ 앱의 설정 화면을 구성하는 XML 파일 적용하기: PreferenceFragmentCompat을 상속받는 개발자 서브 클래스 만들기. PreferenceFragmentCompat 서브 클래스의 onCreatePreferences() 함수 내에서 setPreferencesFromResource() 함수를 호출하면 XML 파일이 적용된다. 매개변수로 설정 XML 파일 정보를 알려주면 된다.

```
public class SettingFragment extends PreferenceFragmentCompat {
    @Override
    public void onCreatePreferences(Bundle savedInstanceState, String rootKEy) {
        setPreferencesFromResource(R.xml.settings, rootKey);
    }
}
```

화면을 출력하려면 Activity 클래스가 필요한데, 위의 클래스는 Activity 클래스가 아니다. 따라서 Activity 클래스에서 설정할 수 있도록 PreferenceFragmentCompat 서브 클래스를 화면에 띄어야 하고, 이를 위해 액티비티의 레이아웃 XML 파일에서 <fragment> 태그로 PreferenceFragment 서브 클래스를 등록한다. 이때 android:name 속성 값에 전체 패키지명을 입력한다.

```
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    class="com.example.test5_13.SettingFragment"
    android:id="@+id/setting_fragment"/>
```

액티비티가 실행되면서 PreferenceFragmentCompat 클래스를 화면에 출력하고, 이 클래스에서 설정을 위한 XML을 적용하는 구조이다.

● 설정 제어 및 이벤트 처리: 사용자가 설정 항목을 클릭한 순간의 이벤트를 처리하거나 설정 값을 설정 항목 옆에 나타나게 하는 방법. 즉, 코드에서 설정을 제어하는 방법이다.

```
<EditTextPreference
    app:key="id"
    app:title="ID 설정" />
<ListPreference
    android:key="sound"
    android:title="알림음"
    android:entries="@array/array_voice"
    android:entryValues="@array/array_voice" />
```

XML 파일에 다음과 같은 태그가 작성되어 있을 때 설정한 내용을 summary에 출력하고 싶다면, summary 출력이란 android:summary 설정으로 XML 파일에 지정된 문자열이 출력되도록 하는 것이므로 사용자가 설정한 값은 그 순간의 동적인 문자열이고 이 문자열을 이용해 summary를 출력하려면 findPreference() 함수로 설정 객체를 획득한 후 제어해주어야 한다. 아래의 코드에선 사용자가 입력하거나 선택한 데이터를 그대로 Summary에 지정하기 위해 SimpleSummaryPRovider을 이용하였다.

```
EditTextPreference idPreference = findPreference("id");
ListPrefernece colorPReference = findPRefernece("sound");

idPreference.setSummaryProvider(EditTExtPrefernece.SimpleSummaryProvider.getInstance());
colorPreference.setSummaryProvider(ListPreference.SImpleSummaryProvider.getInstance());
```

만약, 사용자가 입력한 데이터를 이용해 로직을 실행한 후 그 결과를 Summary에 저장하고 싶다면, SummaryProvider을 구현한 객체를 이용한다.

```
idPreference.setSummaryProvider(new Preference.SummaryProvider() {
    @Override
    public CharSequence provideSummary(Preference preference) {
        String text = ((EditTextPrefernece)preference).getText();
        if (TextUtils.isEmpty(text)) {
            return "설정이 되지 않았습니다.";
        } else {
            return "설정된 ID 값은 : " + text + "입니다.";
        }
    }
});
```

또한 설정이 변경되는 순간을 감지해서 변경된 값을 이용하는 경우도 있다. 이때 설정 변경을 감지하는 방법에는 두 가지가 있다.

1) Preference.OnPreferenceChangeListener을 이용하는 방법: 각 Preference 객체마다 이벤트 핸들러를 직접 지정하여 설정 내용이 변경되는 순간의 이벤트를 처리한다.

```
idPreference.setOnPreferenceChangeListener(new Preference.OnPreferenceChangeListener() {
    @Override
    public boolean onPreferenceChange(Preference prefernece, Object newValue) {
        String key = prefernece.getKey();
        String value = (String) newValue;
        Log.d("kkang", Key + " changed..." + value);
        return true;
});
```

2) SharedPreferences.OnSharedPreferenceChangeListener를 이용하는 방법: 모든 설정의 객체 변경을 하나의 이벤트 핸들러에서 처리한다. 그리고 Preference.OnPreferenceChangeListener을 이용해 이벤트 콜백함수의 매개변수로 이벤트가 발생한 Prefernece 객체와 바뀐 값을 가져온다.

설정 프래그먼트 클래스에서 구현하고, 추상함수인 onSharedPreferenceChanged()를 재정의한다. 그리고 registerOnSharedPreferenceChangeListener() 함수로 이벤트 핸들러를 등록한다. 이벤트를 더 이상 감지하지 않아도 될 때 unregisterOnSharedPreferenceChangeListener() 함수를 이용해서 이벤트 등록을 해제한다.

```
public class SettingFragment extends PrefernceFragmentCompat implements SharedPreferenc
    // ....
    @Override
    public void onSharedPreferenceChanged(SHaredPReferences sharedPreferences, String s)
        if (s.equals("id")) {
            Log.i("kkang", "newValue : " + sharedPRefernece.getString("id", ""));
        }
    }

    @Override
    public void onResume() {
        super.onResume() {
            super.onResume();
            // 이벤트 핸들러 등록
            getPreferenceManager().getSharedPreferences().registerOnSharedPreferenceChar
    }

    @Override
    public void onPause() {
        super.onPause();
        // 이벤트 핸들러 등록 해제
        getPreferenceManager().getSharedPreferences().unregisterOnSharedPrefernceChange
    }

 }
```