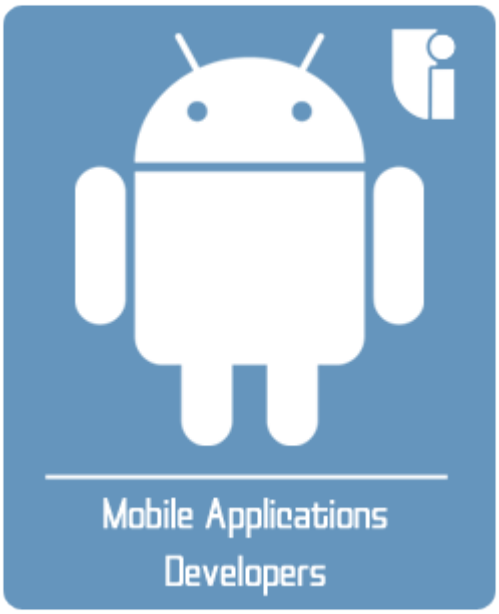
akademia androida



Składowanie danych

część VI

agenda

1. SharedPreferences.
2. Pamięć wewnętrzna i karta SD.
3. Pliki w katalogach /res/raw i /res/xml.
4. Baza danych SQLite.
5. Zadanie.

1. SharedPreferences.

Jednym ze sposobów magazynowania danych w Androidzie jest mechanizm zapewniony przez API o nazwie `SharedPreferences`. Przechowuje on dane w formie {*klucz,wartość*} w utworzonych automatycznie plikach XML znajdujących się w katalogu */data/data/NAZWA_PAKIETU/shared_pref/*. Najczęściej używa się go (jak sama nazwa wskazuje) do przechowywania ustawień aplikacji współdzielonych pomiędzy aktywnościami. Plik XML może mieć nazwę zdefiniowaną przez nas lub domyślną (nazwa klasy bieżącej aktywności). Dostęp do `SharedPreferences` uzyskujemy poprzez `Context` i wywołaniu odpowiedniej metody.

- `getPreferences(int mode)`
- `getSharedPreferences(String name, int mode)`

Tryby dostępu:

- `Context.MODE_PRIVATE`
- `Context.MODE_WORLD_READABLE  //wycofane w API 17`
- `Context.MODE_WORLD_WRITEABLE //wycofane w API 17`

Przykład zapisu danych do `SharedPreferences`:

```
SharedPreferences sharedPref = getPreferences(Context.MODE_PRIVATE);
SharedPreferences.Editor sharedPreferencesEditor = sharedPref.edit();
sharedPreferencesEditor.putString("user_nickname", "Nick");
sharedPreferencesEditor.commit();
```

Przykład odczytu danych z `SharedPreferences`:

```
sharedPref.getString("user_nickname", "default value")
```

2. Pamięć wewnętrzna i karta SD.

Zwykłe pliki możemy zapisać w pamięci wewnętrznej telefonu (dokładniej mówiąc w katalogu */data/data/NAZWA_PAKIETU/files/)* lub na karcie pamięci. Mechanizm jest podobny do tego w Java SE lub Java EE.

Za pomocą poniższych metod otwieramy plik do zapisu/odczytu w pamięci wewnętrznej telefonu.

```
FileOutputStream fos = openFileOutput("plik.txt",Context.MODE_PRIVATE)
FileInputStream fis = openFileInput("plik.txt")
```

Aby zapisać na kartę pamięci najpierw musimy pobrać ścieżkę za pomocą metody `getExternalStorageDirectory()`. Przykład poniżej.

```
File sdCardDirectory = Environment.getExternalStorageDirectory();
   if (sdCardDirectory.exists() && sdCardDirectory.canWrite()) {
       File file = new File(sdCardDirectory.getAbsolutePath() + "/" + "plik.txt");
          try {
                    file.createNewFile();
                    if (file.exists() && file.canWrite()) {
                          FileOutputStream fos = new FileOutputStream(file);
                    }
              }catch (IOException e) {}
    }
```

W przypadku zapisu na kartę należy pamiętać o dodaniu do `AndroidManifest.xml` uprawnień.

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

3. Pliki w katalogach /res/raw i /res/xml.

Istnieje możliwość umieszczenia w naszym projekcie plików dowolnego formatu w folderze */res/raw,* pozostaną one tam w niezmienionej formie, może to być tekst, film, obraz etc. Po umieszczeniu ich w projekcie stają się dostępne jako zasoby w aplikacji. Przykład odwołania się do tego typu pliku:

```
InputStream is = getResources().openRawResource(R.raw.movie);
```

Drugim typem pliku, który krótko omówię jest plik XML w folderze */res/xml* . Są to zwykłe pliki XML, które zostają automatycznie skompilowane do zasobów aplikacji. Poniżej przykład z dokumentacji *XmlPullParser* parsowania pliku XML:

```
try {
	XmlPullParser xpp = getResources().getXml(R.xml.people);
	int eventType = xpp.getEventType();

	while (eventType != XmlPullParser.END_DOCUMENT) {
		if (eventType == XmlPullParser.START_DOCUMENT) {
			Log.d(TAG, "Start document");
		}
		else if (eventType == XmlPullParser.START_TAG) {
			Log.d(TAG, "Start tag " + xpp.getName());
		}
		else if (eventType == XmlPullParser.END_TAG) {
			Log.d(TAG, "End tag " + xpp.getName());
		}
		else if (eventType == XmlPullParser.TEXT) {
			Log.d(TAG, "Text " + xpp.getText());
		}
		eventType = xpp.next();
	}
	Log.d(TAG, "End document");
}
catch (XmlPullParserException e) {}
catch (IOException e) {}
```

W ten sposób można parsować również pliki pobrane z serwera lub innego dowolnego miejsca.

4. Baza danych SQLite.

System Android został wyposażony w lokalną bazę danych *SQLite*, która znakomicie sprawdza się jako magazyn dla danych naszej aplikacji. API udostępnia łatwy sposób pracy z tego typu bazą. Najbardziej zalecanym rozwiązaniem jest zgrupowanie operacji na bazie w jednej klasie, która dziedziczy po `SQLiteOpenHelper`. Taka klasa musi zawierać co najmniej jeden konstruktor `DatabaseHandler(Context context)` i dwie metody nadpisujące metody z klasy bazowej:

- `onCreate(SQLiteDatabase db)`
- `onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)`

W metodzie `onCreate` za pomocą metody `db.execSQL(String)` tworzymy tabele (jako argument przekazując zapytanie SQL). W metodzie `onUpgrade` powinniśmy usunąć tabele (jeżeli istnieją) i ponownie wywołać metodę `onCreate(SQLiteDatabase db)`.

Przykładowa klasa:

```
public class DatabaseHandler extends SQLiteOpenHelper {


	private static final int VERSION = 1;

	private static final String DATABASE_NAME = "mad_database";
	private static final String TABLE_NAME = "zwierzeta";

	private static final String ID_COLUMN = "id";
	private static final String NAME_COL = "nazwa";
	private static final String TYPE_COL = "rodzaj";

	public DatabaseHandler(Context context) {
		super(context, DATABASE_NAME, null, VERSION);
	}
```

```
	@Override
	public void onCreate(SQLiteDatabase db) {
		String query = "CREATE TABLE " + TABLE_NAME + "(" + ID_COLUMN +
                                                "INTEGER PRIMARY KEY," + NAME_COL +
                                                " TEXT," + TYPE_COL + " TEXT" + ")";
		db.execSQL(query);

	}

	@Override
	public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
		db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);
		onCreate(db);
	}
}
```

Oczywiście klasa powinna poza tym zawierać metody do pobierania, aktualizowania i usuwania danych. Przykład zapisu do bazy z wnętrza klasy:

```
SQLiteDatabase db = this.getWritableDatabase();

ContentValues newRowValues = new ContentValues();
newRowValues.put(NAME_COL, "Wąż");
newRowValues.put(TYPE_COL, "Dusiciel");

db.insert(TABLE_NAME, null, newRowValues);
db.close();
```

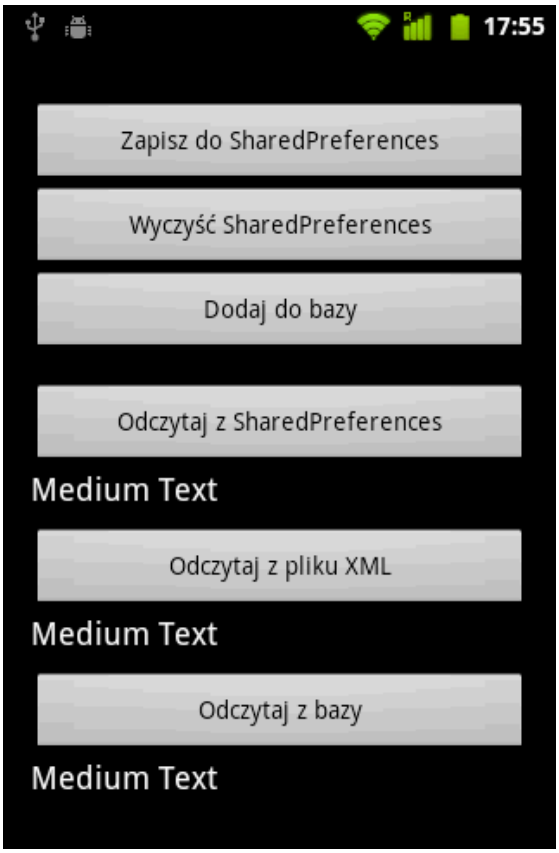
Przykład pobrania danych za pomocą SELECT:

```
SQLiteDatabase db = this.getWritableDatabase();
Cursor cursor = db.rawQuery( "SELECT  * FROM " + TABLE_NAME , null);

if (cursor.moveToFirst()) {
  do {
        int id = cursor.getInt(0);
        String nazwa = cursor.getString(1);
        String rodzaj= cursor.getString(2);
     }while (cursor.moveToNext());
}
```

5. Zadanie.

- Stworzyć aktywność z sześcioma przyciskami i trzema polami tekstowymi .
- Przycisk **Zapisz do SharedPreferences** zgodnie z nazwą zapisuje dane:
  - *name* → Jan Kowalski
  - *age* → 32 .
- Przycisk **Wyczyść SharedPreferences** usuwa wszystkie dane.
- Przycisk **Dodaj do bazy** zgodnie z nazwą doda do tabeli w bazie przykładowe dane osób *{name,age}*.
- **Pozostałe trzy przyciski** służą do odczytania zapisanych wcześniej danych i wyświetlenia ich w dowolnej formie, np. poprzez pola tekstowe.
- Przycisk „**Odczytaj z XML**” odczytuje dane z pliku *people.xml* w katalogu */res/xml*.



Struktura pliku:

```
<people>
    <person>
      <name>Jan Kowalski</name>
        <age>32</age>
    </person>
     ...
</people>
```

Dziękuję za uwagę !

Kontakt:

- sswierczek@wi.zut.edu.pl
- mad@zut.edu.pl