Лекция №8

https://github.com/IFMO-Android-2016/lesson8

Хранение данных

File System

Shared Preferences

SQLite

https://developer.android.com/guide/topics/data/data-storage.html

Файловая система

Internal Storage

External Storage

Файловая система

- Internal Storage

  `/data/data/<package ID>/`

  – Постоянный доступ к файлам
  – По умолчанию файлы доступны только вашему приложению*
  – При удалении приложения все файлы удаляются

  *или root

Файловая система

- External Storage

  `/sdcard/` – корневая директория*
  - Доступно не всегда (отключено пользователем)
  - Файлы доступны **всем**, включая пользователя
  - После удаления приложения, файлы могут остаться

*имя корневой директории может меняться, поэтому его нельзя сохранять. Нужно пользоваться

`Environment.getExternalStorageDirectory()`

Internal Storage

Стандартная структура директорий приложения:

`/data/data/<package ID>/`

- `files/` – любые файлы
- `cache/` – кэш (может быть очищен пользователем в настройках)
- `databases/` – базы данных, доступ через SQLiteOpenHelper
- `shared_prefs/` – настройки, доступ через SharedPreferences

Internal Storage

- Получение доступа к стандартным директориям:

```
/data/data/<package ID>/

   files/          - Context.getFilesDir()
   cache/          - Context.getCacheDir()

   databases/
      - Context.getDatabasePath(String dbName)

   shared_prefs/
      - Context.getSharedPreferences(
                           String prefsName, int mode)
```

External Storage

Файлы приложений:

```
/sdcard/
   Android/data/ «скрытые файлы»*
      <package ID#1>/    приложение №1
         files/
         cache/
      <package ID#2>/    приложение №2
```

*стандартные приложение (Галерея) не показывают эти файлы, но доступ к ним никак не ограничен.

External Storage

Общие файлы:

```
/sdcard/
    Music/
    Pictures/
    Ringtones/
    Download/
    …

Context.getExternalFilesDir(
        Environment.DIRECTORY_XXX)
```

External Storage

Правила использования:

1. Прописываем пермиссии в манифесте

```
<manifest >

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

https://developer.android.com/reference/android/Manifest.permission.html

External Storage

Правила использования:

2. Получаем корневые директории через API,
   а храним только относительные пути

```
File filesDir = context.getExternalFilesDir(null);
File cacheDir = context.getExternalCacheDir();

String relativePath = "mydir/filename.ext";
File file = new File(filesDir, relativePath);

// Сохраняем относительный путь в настройки
sharedPrefs.put("file_path", relativePath);
```

External Storage

Правила использования:

3. Всегда быть готовым к пропаже или порче файлов, а также к ситуации недоступности

```
if (!Environment.MEDIA_MOUNTED.equals(
      Environment.getExternalStorageState())) {
      ...
      // External Storage временно недоступен
}
```

Чтение / запись в файлы

Обязательный к применению паттерн кодирования: освобождать все ресурсы в блоке finally

```
InputStream in = null;

try {
    in = new FileInputStream(file);
    ...

} finally {
    if (in != null) {
        try {
            in.close();
        } catch (IOException e) {…}
    }
}
```

```
InputStream in = null;
OutputStream out = null;

try {
    in = new FileInputStream(srcFile);
    out = new FileOutputStream(destFile);
    ...
} finally {
    if (in != null) {
        try { in.close(); } catch (IOException e) {…}
    }
    if (out != null) {
        try { out.close(); } catch (IOException e) {…}
    }
}
```

Не только файлы, но и всё, что закрывается…

```
Cursor cursor = null;

try {
    cursor = db.execSQL("SELECT...");
    ...

} finally {
    if (cursor != null) {
        cursor.close();
    }
}
```

… или отключается

```
HttpUrlConnection conn = null;

try {
    conn = url.openConnection();
    ...

} finally {
    if (conn != null) {
        conn.disconnect();
    }
}
```

Хранение настроек

Shared Preferences

Shared Preferences

- Стандартный механизм для хранения примитивных пар ключ-значение в XML файлах

  `/data/data/<packageID>/shared_prefs/<prefs_name>.xml`

```
<?xml version='1.0' encoding='utf-8'?>
<map>
  <boolean name="cache_enabled" value="true" />
  <long name="deadline_ts" value="1450040399000" />
  <string name="login_name">dmitry.trunin</string>
  <int name="launch_count" value="12" />
</map>
```

Shared Preferences

1. Получаем объект SharedPreferences

```
// Получаем настройки, которые хранятся в файле
// <package dir>/shared_prefs/my_prefs.xml
// и доступны только нашему приложению
SharedPreferences prefs =
      context.getSharedPreferences(
            "my_prefs",
            Context.MODE_PRIVATE);
```

Shared Preferences

2. Сохраняем значения в настройки:

```
SharedPreferences.Editor editor = prefs.edit();
editor.putString("login_name", "dmitry.trunin");
editor.putBoolean("cache_enabled", true);
editor.putInt("launch_count", 12);
editor.remove("deadline_ts");
editor.apply(); // editor.commit()
```

apply() – асинхронная запись в файл

commit() – синхронная запись в файл (нельзя вызывать в UI потоке)

Shared Preferences

- Читаем значения из настроек

```
String login = prefs.getString("login_name", null);
boolean cacheEnabled =
        prefs.getBoolean("cache_enabled", false);
long deadlineTs = prefs.getLong("deadline_ts", 0);
Map<String, ?> map = prefs.getAll();

int launchCount =
        (int) prefs.getLong("launch_count", 0);
// ClassCastException
```

Shared Preferences

1. При первом вызове любого из методов вида `SharedPreferences.getXXX` происходит чтение всех значений из файла (нельзя вызывать в UI потоке)
2. Все последующие вызовы `getXXX` не обращаются к файлу (можно вызывать в UI потоке)
3. Изменения сразу становятся видны другим частям приложения в том же процессе (не обязательно использовать один объект `SharedPreferences`)

База данных SQLite

https://www.sqlite.org/

CREATE TABLE

CREATE INDEX

INSERT

SELECT

База данных SQLite

SQLite – компактная встраиваемая* (embedded) реляционная база данных.

Де-факто стандарт в мобильных приложениях

- Компактная: одна база данных – один файл в `/data/data/<package ID>/databases/`
- Встраиваемая – код SQLite выполняется прямо в процессе приложения.
- Реляционная – таблицы, язык SQL

Таблицы

rowid	city_id	name	country	latitude ▲	longitude
186	6295642	Letten	CH	47.32811	8.53458
183	6295429	Fahrweid (s...	CH	47.408138	8.41367
185	6291739	Bärenbohl	CH	47.434929	8.51832
146	5815135	Washington	US	47.500118	-120.501472
184	7286907	Riehen	CH	47.579418	7.65117
203	6556314	Haar	DE	48.099998	11.7333
156	7871309	Wels(Stadt)	AT	48.16082	14.02164
212	6555717	Baiersbronn	DE	48.515499	8.3495
194	7602475	Pfaffenhof...	DE	48.528431	11.502
101	6255148	Europe		48.69096	9.140625
216	2856307	Ostfildern	DE	48.7267	9.25143

Создание таблицы

Пример:

```
CREATE TABLE cities (
      city_id INTEGER PRIMARY KEY,
      name TEXT NOT NULL,
      country TEXT,
      lat REAL,
      lon REAL
)
```

Полное описание синтаксиса:
https://www.sqlite.org/lang_createtable.html

Типы данных SQLite

- NULL
- INTEGER – целое со знаком от 1 до 8 байт
- REAL – действительное число, 8 байт
- TEXT – текст (String), по умолчанию в UTF-8
- BLOB – бинарные данные (byte[])

https://www.sqlite.org/datatype3.html

Создание таблицы

```
city_id INTEGER PRIMARY KEY,
name TEXT NOT NULL,
```

Колонка `PRIMARY KEY` определяет ключ, по которому можно быстро найти строчку в таблице:

- Поиск по `city_id` работает быстро – O(log n)
- Поиск по `name` работает медленно – O(n)
- При попытке вставить в `name` значение NULL возникнет ошибка.
- При попытке вставить в `city_id` уже существующее значение возникнет ошибка

Запись данных в таблицу

Выражение `INSERT` для вставки новых* строк:

```
INSERT INTO cities (
    city_id, name, country,
    latitude, longitude
) VALUES (
    102908598, 'Пушкин', 'RU',
    59.715, 30.417
);
```

https://www.sqlite.org/lang_insert.html

Изменение данных

Выражение `UPDATE` для изменения данных

```
// Заменить ru на RU в колонке country
UPDATE cities SET country='RU'
              WHERE country='ru';

// Все значения name в верхний регистр
UPDATE cities SET name=upper(name)
            WHERE name NOT NULL;
```

https://www.sqlite.org/lang_update.html

Выборка данных

Выражение `SELECT` – выборка по условию

```
SELECT
    city_id,name,country
FROM
    cities
WHERE
    longitude BETWEEN 29.5 AND 30.5
    AND
    latitude BETWEEN 59.5 AND 60.5;
```

// выборка городов в окресностях СПб

```
// Выборка всех городов, название которых
// начинается на 'Saint'
// (* -- запрашиваются все колонки)
SELECT * FROM cities
         WHERE name LIKE 'Saint%';

// Количество городов за северным
// полярным кругом
SELECT count(*) FROM cities
         WHERE latitude > 66.5622;
```

SQLite

- Полное описание языка:

https://www.sqlite.org/lang.html

SQLite в Android

SQLiteDatabase

SQLiteOpenHelper

Cursor

SQLiteDatabase

Низкоуровневый API для прямой работы с БД

```
// Открыть БД из файла
// (создать новый файл базы данных, если надо)
SQLiteDatabase db = SQLiteDatabase
    .openOrCreate(file.getPath(), ... );

// Выполнить SQL выражение (кроме SELECT)
db.execSQL("CREATE TABLE ...");

// Выполнить SELECT
Cursor cursor = db.rawQuery("SELECT * FROM ...");

db.close(); // Закрыть БД
```

SQLiteDatabase

- Должен быть только один объект `SQLiteDatabase` для одной базы данных
- Методы `SQLiteDatabase` можно вызывать только в фоновых потоках
- Thread-safe – можно работать с одним объектом `SQLiteDatabase` (одной БД) из разных потоков.
- Закрывать БД (`close()`) надо очень осторожно (либо не закрывать совсем).

SQLiteOpenHelper

Базовый абстрактный класс, значительно упрощает процесс создания базы данных и апгрейда схемы БД при установке новой версии приложения.

Каждой БД – свой класс `SQLiteOpenHelper`:

1. В конструкторе указать номер версии
2. Реализовать паттерн **singleton**
3. Реализовать метод `onCreate` – создание всех таблиц (а также индексов, триггеров и пр.)
4. Реализовать метод `onUpgrade` – изменение схемы БД при увеличении номера версии

INSERT

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

ContentValues values = new ContentValues();
values.put("name", "Пушкин");
values.put("country", "RU");

// Возвращает rowid или -1 в случае ошибки
long rowId = db.insert("cities", null, values);

// То же самое на языке SQL
db.execSQL("INSERT INTO cities (name, country)"
      + " VALUES ('Пушкин', 'RU')");
```

SELECT

Обязательный к применению паттерн

```
Cursor cursor = null;

try {
    cursor = db.query(...);

} finally {
    if (cursor != null) {
        try { cursor.close(); }
        catch (Exception e) { ... }
    }
}
```

SELECT

- Объект `Cursor` позволяет оперировать с результатом выполнения выражения SELECT строка за строкой.
- Когда вы получаете `Cursor`, выражение только подготовлено, но еще не выполнено.
- При вызове `cursor.moveToNext()` вычисляется следующая строчка результата.
- `cursor.getCount()` приводит к чтению всего результата выражения (может быть долго) – лучше не использовать.

Выполнение SQL запроса

```
cursor = db.query(
    // table
    "cities",
    // projection
    new String[] { "name", "country" },
    // selection
    "country=? AND name LIKE ?",
    // selection args
    new String[] { "RU", "Saint%" }
);

SELECT name, country FROM cities
    WHERE country='RU' AND name LIKE 'Saint%';
```

Чтение данных из курсора

```
// Проверяем, что результат есть и не пустой
if (cursor != null && cursor.moveToFirst()) {
    // Итерируем строка-за-строкой
    for (; !cursor.isAfterLast();
                        cursor.moveToNext()) {
        // Получаем значения колонок по индексу
        // из projection
        String name = cursor.getString(0);
        String country = cursor.getString(1);
        ...
    }
}
```

Демо приложение

https://github.com/IFMO-Android-2016/lesson8

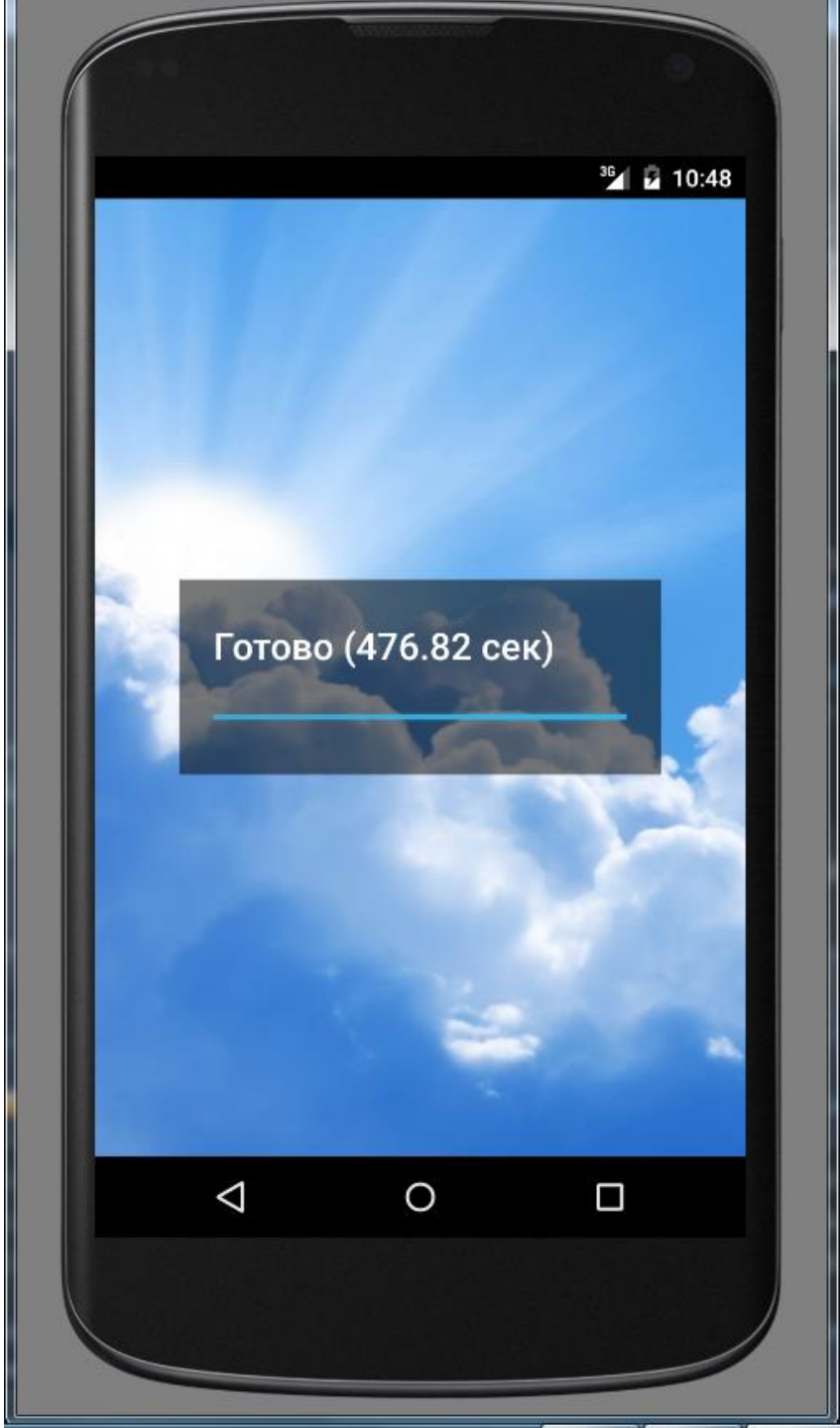
Демо приложение

1. Скачивает файл с данными городов в формате JSON и сохраняет в файл – `DownloadActivity`
2. Создает базу данных городов при помощи `CityDBHelper`
3. Импортирует данные из JSON в базу данных – `CityFileImporter`
4. Хранит настройки в Shared Preferences - `WorldcamPreferences`

Запись данных в БД

- GZIP 3.8 Мб
- JSON 18 Мб
- ~200k городов

8 минут на эмуляторе (Core i5, 2.8 GHz)



Оптимизация INSERT

Транзакция – группа последовательных операций с БД, которая представляет собой логически единое действие.

Все операции в транзакции выполняются успешно, либо ни одна операция не выполняется.

Пример (банковский перевод):

1. Уменьшить баланс счета №1 на 10 рублей
2. Увеличить баланс счета №2 на 10 рублей

Оптимизация INSERT

Для одиночной операции SQLite открывает транзакцию, выполняет операцию и закрывает транзакцию – добавляются расходы на открытие/закрытие транзакции.

```
for (...) {
    ContentValues values = ...;
    // Транзакция открывается
    db.insert("cities", null, values);
    // Транзакция закрывается
}
```

Оптимизация INSERT

Явное выполнении всех операций в одной транзакции даёт значительное ускорение:

```
db.beginTransaction();
try {
    for (...) {
        ContentValues values = ...;
        db.insert("cities", null, values);
    }
    db.setTransactionSuccessful();
} finally {
    db.endTransaction();
}
```

Оптимизация INSERT

`SQLiteStatement` – подготовленное заранее выражение, которое можно выполнить много раз, не компилируя его каждый раз заново.

```
// SQL выражение каждый раз компилируется
db.execSQL("INSERT INTO cities ...");

// Здесь тоже происходит неявная компиляция
db.insert("cities", null, values);
```

Оптимизация INSERT

```
SQLiteStatement insert = db.compileStatement(
    "INSERT INTO cities (city_id, name)"
        + " VALUES (?, ?)";

for (...) {
    insert.bindLong(1, cityId);
    insert.bindString(2, name);

    insert.executeInsert();
}
```

SQLiteStatement тоже надо закрывать!

```
SQLiteStatement insert = null;

try {
    insert = db.compileStatement("...");

} finally {
    if (insert != null) {
        try { insert.close(); }
        catch (Exception e) { ... }
    }
}
```

Конец лекции №8

https://github.com/IFMO-Android-2016/lesson8