SoftUni Foundation

android development

for beginners

Join at Slido.com with #AndroidSoftUni

Databases. Architectures. Design Patterns

- Databases
  - SQLite
  - ORMs
  - Room
- Architectures
  - (Dagger)
  - MVP
  - MVVM
  - Onion/clean arch
- Good Practises
  - Lambdas

SQLite

- Базата данни по подразбиране в Андроид
- Използва се чрез API на сравнително ниско ниво
- Това я прави опасна
- Ако използвате директно SQLite е препоръчително да използвате такава абстракция, че да се минимизират ръчните грешки
- Това е най-бързия вариант за база данни на устройството
- Трябва да се наследи класа `SQLiteOpenHelper`
- Той позволява достъп до базата за извършване на CRUD

ORMs

- Object Relational Mapping
- Библиотеки, които, използвайки абстракция, позволяват записване/извличане на Java обекти в базата данни
- ORM Lite
- SugarOrm
- ActiveAndroid
- DBFlow

ORMs - ORM Lite

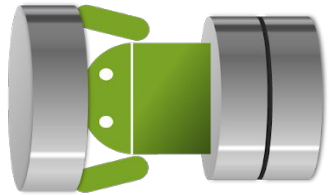
- The first Android ORM library is in fact NOT an Android ORM, it's a Java ORM with SQL databases support. It can be used anywhere where Java is used (JDBC connections, Spring framework and of course Android).

- **Pros:**
- well known library with rich documentation
- easy to setup
- fast

- **Cons:**
- left join or more complicated queries are difficult with OrmLite. The setup of relationships is a bit confusing and has its drawbacks.
- raw queries are possible but require you to parse the object by hand instead of doing it for you.
- a bit hard to use overall
- it can be a little slow at times.
- 350 kb library

ORMs - SugarOrm

- SugarOrm is an ORM built only for Android. It comes with an API which is both simple to learn and simple to remember. It creates necessary tables itself, gives you a simple methods of creating one-to-one and one-to-many relationships, and also simplifies CRUD by using only 3 functions, save(), delete() and find() (or findById())
- **Pros:**
  • extremely easy to setup
  • a real pleasure to use API
  • online documentation
  • supports migrations
- **Cons:**
  • slow. It's actually one of the slowest.
  • The library is unfinished. Currently it doesn't support naming of the columns, so your stuck with the field name of the entity class.

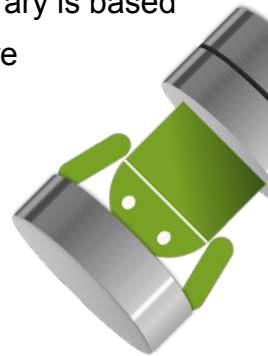


ORMs - ActiveAndroid

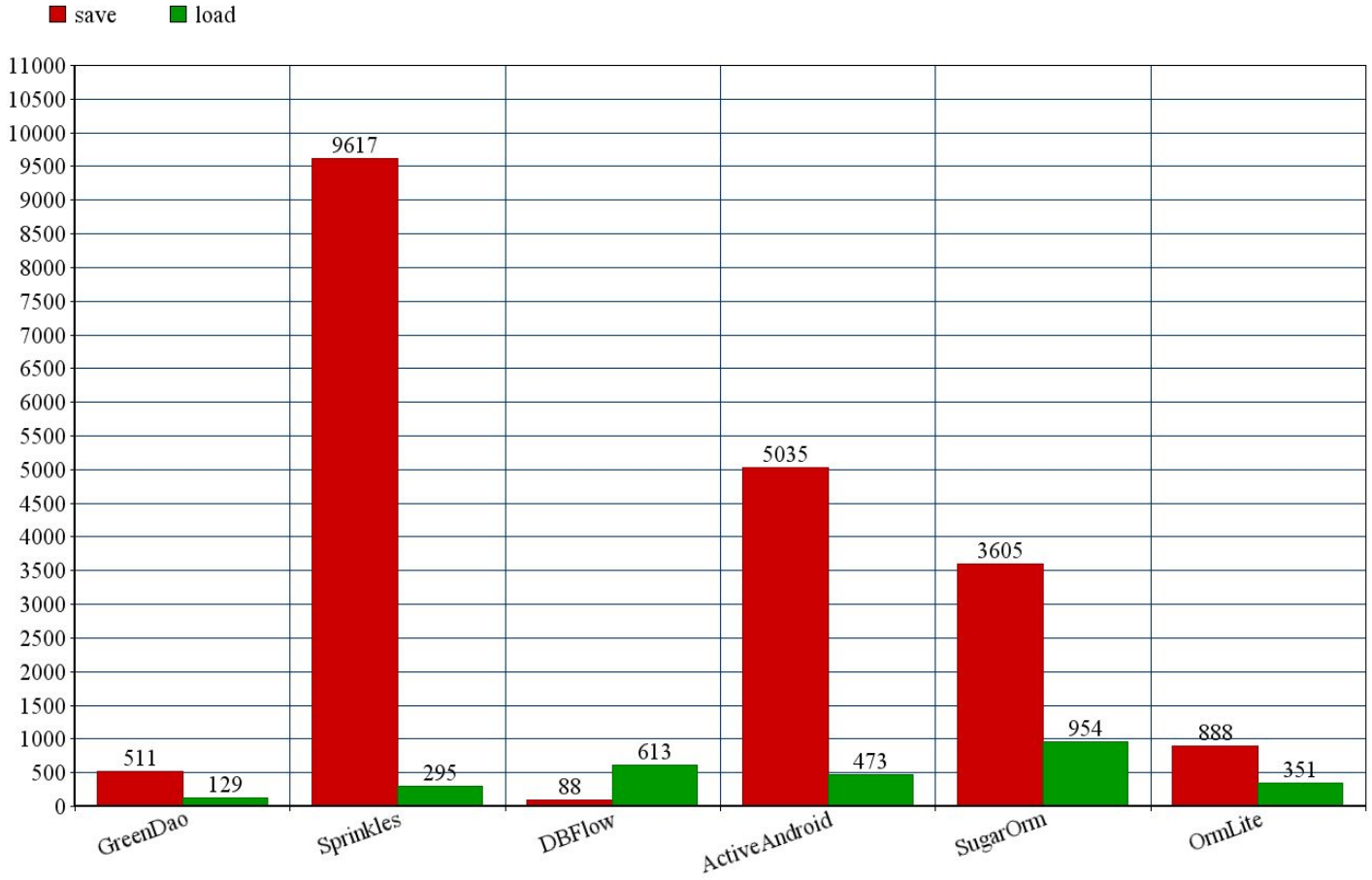
- ActiveAndroid allows you to save and retrieve SQLite database records without ever writing a single SQL statement. Each database record is wrapped neatly into a class with methods like save() and delete().
- **Pros:**
  - **extremely easy to setup**
  - **the API is almost the same as SugarOrm (awesome)**
  - **online documentation**
  - **one of the fastest ORM's**
  - **supports migrations**
- **Cons:**
  - **there maybe problems with lollipop**
  - **latest release was 2014 so maybe work on the library has stopped.**

ORMs - DBFlow

- A robust, powerful, and very simple ORM android database library with annotation processing. The library is based on Active Android, Schematic, Ollie, and Sprinkles, but takes the best of each while offering much more functionality and extensibility.
- **Pros:**
  - **easy to setup**
  - **easy API**
  - **native support for async calls to the database**
  - **online documentation**
  - **extremely fast ORM**
  - **supports migrations**
  - **uses annotation processing instead of reflection which gives this library it's speed.**
- **Cons:**
  - **Cannot write raw queries**



ORMs - Comparison



Room

- Подобно на ORM е абстракция върху SQLite
- Позволява лесно поддържане на актуална онлайн база
- В бъдеще ще позволи заменянето на SQLite с по-добра база данни
- Query-тата се проверяват по време на компилация!!
- Има три главни части
  - Database - основен дб контролер
  - Entity - данни за една таблица в базата
  - Dao - функции на една таблица в базата

Room - Entity

```
@Entity
public class User {
    @PrimaryKey
    private int uid;

    @ColumnInfo(name = "first_name")
    private String firstName;

    @ColumnInfo(name = "last_name")
    private String lastName;

    // Getters and setters are ignored for brevity,
    // but they're required for Room to work.
}
```

Room - Dao

```
@Dao
public interface UserDao {
    @Query("SELECT * FROM user") List<User> getAll();

    @Query("SELECT * FROM user WHERE uid IN (:userIds)")
    List<User> loadAllByIds(int[] userIds);

    @Query("SELECT * FROM user WHERE first_name LIKE :first AND "
           + "last_name LIKE :last LIMIT 1")
    User findByName(String first, String last);

    @Insert void insertAll(User... users);

    @Delete void delete(User user);
}
```

Room - Database

```
@Database(entities = {User.class}, version = 1)
public abstract class AppDatabase extends RoomDatabase {
    public abstract UserDao userDao();
}
```

Използване:

```
AppDatabase db = Room.databaseBuilder(getApplicationContext(),
                AppDatabase.class, "database-name").build();
db.userDao().insertAll(new User());
```

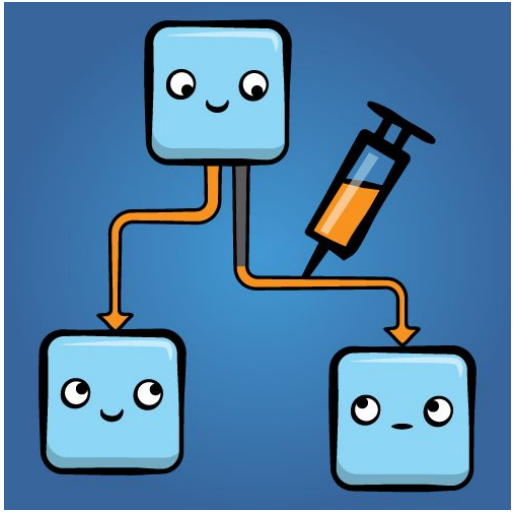
android architecture

Същност

- Архитектурата на едно приложение е зависимостта между отделните му компоненти и как те са разпределени в пакети и модули
- Всяко приложение има архитектура, дори да не сте помислили за нея.
- Има различни готови шаблони, които може да използвате, нагласяйки ги спрямо вашите нужди
  - MVP
  - MVVM
  - Clean Architecture

Dependency Injection

- Начин за създаване на обекти
- Външно тяло създава обектите и ги дава на класовете, които искат да ги използват
- Позволява по-голямо ниво на абстракция и имплементиране на различни архитектури
- В андроид се използва библиотеката Dagger



MVP

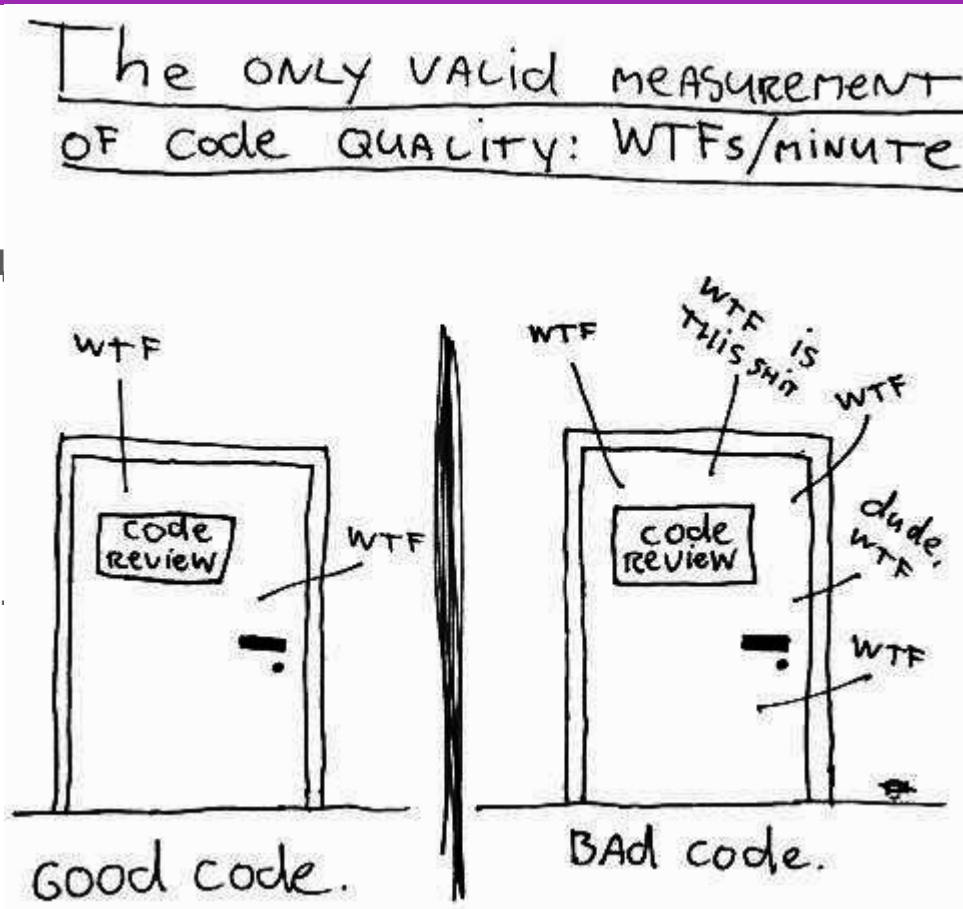
- Състои се от три основни части - Model View Presenter
- Разделя логиката от презентационния слой
- Model - слоя с данните (от интернет, от локалната база, от блутуут у-во)
- View - обикновено е Activity, Fragment или View
- Presenter - клас, който взима данните от различни модели и контролира едно View
- Във View-то не трябва да има никакъв код, който да променя състоянието му, всичко това се решава и прави от Presenter-а

MVVM

- Състои се от три основни части - Model View ViewModel
- Основната разлика с MVP е че докато при MVP презентъра директно казва на View-то какво да прави, при MVVM View-то само се абонира да слуша ViewModel-а. Т.е. ViewModel-а не знае за View-то.
- Това намалява нуждата от използване на интерфейси за комуникация между слоевете.
- При MVVM потребителя на данните знае за създателя им, но обратното не е вярно
- ViewModel-а позволява View-та да се абонират не само за евентите му, но и за състояния

Clean Architecture

- Не толкова архитектура, колкото поредица правила, които ако се спазват водят до чист и подреден код
- Основната черта на архитектурата е съществуването на Usecases - малки класове с бизнес логика, правещи само 1 нещо
- Логиката за един екран се държи от т. нар. Интерактори, които пък вътрешно използват множество usecases за да постигнат желания резултат



patterns and good practises

Singleton

- Най-простия design pattern
- Позволява създаването само на един обект от класа в който е имплементиран
- Трябва ви
  - Статична инстанция на класа в самия клас
  - Да напарвите конструктора частен
  - Да създадете статичен метод, който връща тази статична инстанция

lambdas

Защо се използват?

Ламбдата е анонимна функция. Малка, стегната, анонимна функция.

Позволява ни да не пишем излишен код и да навързваме няколко операции една след друга четимо.

Пример с итерация

```
List<Integer> list = Arrays.asList(1,2,3,4,5,6,7,8,9);

//External iterator

//External iterator

//Internal iterator

//Internal iterator with lambdas
```

Подобрения

Type Inference:

- не трябва да дефинирате типа
- скобите около параметъра са задължителни само при повече от 1
- при просто предаване на параметъра, той може да се пропусне, но точката се замества с четири точки

Забележка

- Дръжте ламбдите си кратки
- Те могат да бъдат блокове код, но не трябва да бъдат
- Те са просто връзката между кода
- 1 ред ламбда е повече от достатъчен

Ресурси

https://developer.android.com/training/data-storage/sqlite.html

https://developer.android.com/training/data-storage/room/index.html

https://github.com/googlesamples/android-architecture-components/tree/master/GithubBrowserSample

https://medium.com/upday-devs/android-architecture-patterns-part-3-model-view-viewmodel-e7eeee76b73b

https://antonioleiva.com/mvp-android/

https://android.jlelse.eu/fundamentals-of-dependency-injection-and-popular-libraries-in-android-c17cf48b5253

https://developer.android.com/studio/write/java8-support.html

https://www.youtube.com/watch?v=1OpAgZvYXLQ

Homework #1

Направете приложение, което има едно поле за въвеждане на код на баркод и един бутон. При натискане на бутона се прави заявка до https://world.openfoodfacts.org/api/v0/product/737628064502.json На потребителя се показва името на продукта и какво съдържа. Информацията се записва в локалната база.
При повторно търсене на същия баркод, информацията се търси първо в локалната база, и чак ако я няма се правя заявка до апи-то.

Този модел би трябвало да работи с данните от АПИ-то:

```
Product {
	String code; //the code may have a leading zero
	String product_name;
	String ingredients_text;
}
```

Може да прочетете повече за АПИ-то на https://world.openfoodfacts.org/data
Може да надградите приложението, като вместо да се въвежда, баркода се сканира с камерата на телефона. Библиотека за това е https://github.com/zxing/zxing