**DatabaseOpenHelper (класс для создания БД)**

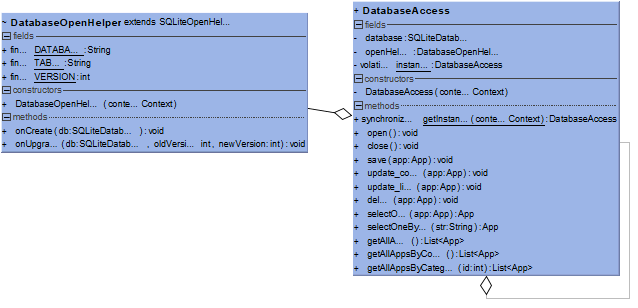
package Database;  
  
import android.content.Context;  
import android.database.sqlite.SQLiteDatabase;  
import android.database.sqlite.SQLiteOpenHelper;  
  
class DatabaseOpenHelper extends SQLiteOpenHelper {  
 public static final String *DATABASE* = "apps.db";  
 public static final String *TABLE* = "app";  
 public static final int *VERSION* = 1;  
  
 public DatabaseOpenHelper(Context context) {  
 super(context, *DATABASE*, null, *VERSION*);  
 }  
  
 @Override  
 public void onCreate(SQLiteDatabase db) {  
 db.execSQL("CREATE TABLE app(name TEXT PRIMARY KEY, cat TEXT, cat\_num INTEGER,limit\_h INTEGER,limit\_m INTEGER,count INTEGER,pack TEXT);");  
 }  
  
 @Override  
 public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {  
  
 }  
}

Библиотека Android содержит абстрактный класс **SQLiteOpenHelper**, с помощью которого можно создавать, открывать и обновлять базы данных. Это основной класс, с которым вам придётся работать в своих проектах. При реализации этого вспомогательного класса от вас скрывается логика, на основе которой принимается решение о создании или обновлении базы данных перед ее открытием. Класс **SQLiteOpenHelper** содержит два обязательных абстрактных метода:

* **onCreate()** — вызывается при первом создании базы данных
* **onUpgrade()** — вызывается при модификации базы данных

В методе **onCreate()** реализовывается логика создания таблиц при помощи SQL-команды.

**DatabaseAccess (класс для работы с бд)**

****

package Database;  
  
import android.content.ContentValues;  
import android.content.Context;  
import android.database.Cursor;  
import android.database.sqlite.SQLiteDatabase;  
import android.util.Log;  
  
import java.util.ArrayList;  
import java.util.List;  
  
import fivemoreminutes.a5moreminutes.Entity.App;  
  
public class DatabaseAccess {  
 private SQLiteDatabase database;  
 private DatabaseOpenHelper openHelper;  
 private static volatile DatabaseAccess *instance*;  
  
 private DatabaseAccess(Context context) {  
 this.openHelper = new DatabaseOpenHelper(context);  
 }  
  
 public static synchronized DatabaseAccess getInstance(Context context) {  
 if (*instance* == null) {  
 *instance* = new DatabaseAccess(context);  
 }  
 return *instance*;  
 }  
  
 public void open() {  
 this.database = openHelper.getWritableDatabase();  
 }  
  
 public void close() {  
 if (database != null) {  
 this.database.close();  
 }

Для управления базой данных SQLite существует класс **SQLiteDatabase**. В классе **SQLiteDatabase** определены методы **query()**, **insert()**, **delete()** и **update()** для чтения, добавления, удаления, изменения данных. Кроме того, метод **execSQL()** позволяет выполнять любой допустимый код на языке SQL применимо к таблицам базы данных, если вы хотите провести эти (или любые другие) операции вручную.

private static volatile DatabaseAccess *instance*;

данная переменная содержит дескриптор БД.

Модификатор volatile накладывает некоторые дополнительные условия на чтение/запись переменной. Важно понять две вещи о volatile переменных:

1. Операции чтения/записи volatile переменной являются атомарными.
2. Результат операции записи значения в volatile переменную одним потоком, становится виден всем другим потокам, которые используют эту переменную для чтения из нее значения.

public void open() {  
 this.database = openHelper.getWritableDatabase();  
}

**getWritableDatabase()**, чтобы открыть и вернуть экземпляр базы данных, с которой мы имеем дело (он будет доступен как для чтения, так и для записи)