DB et Android
SQLite
ORMLite

Cedric Dumoulin

Bibliographie

- https://openclassrooms.com/courses/creez-des-applications-pour-android/les-bases-de-donnees-5
- http://www.tutomobile.fr/comment-utiliser-sqlite-sous-android-tutoriel-android-n%C2%B019/19/10/2010/
  - Exemple Book !

Tutoriels

- http://vogella.developpez.com/tutoriels/android/utilisation-base-donnees-sqlite/
- http://www.androidbegin.com/tutorial/android-ormlite-with-sqlite-database-tutorial/
- Les exemples dans la suite sont tirés de ces tutoriaux.

SQL Lite

SQLite

- http://www.sqlite.org
- C'est une base de données SQL embarquée
  - Self-contained, compact (<500K)
  - Serverless
  - Zero-configuration
  - Transactional
  - SQL

  - Open source

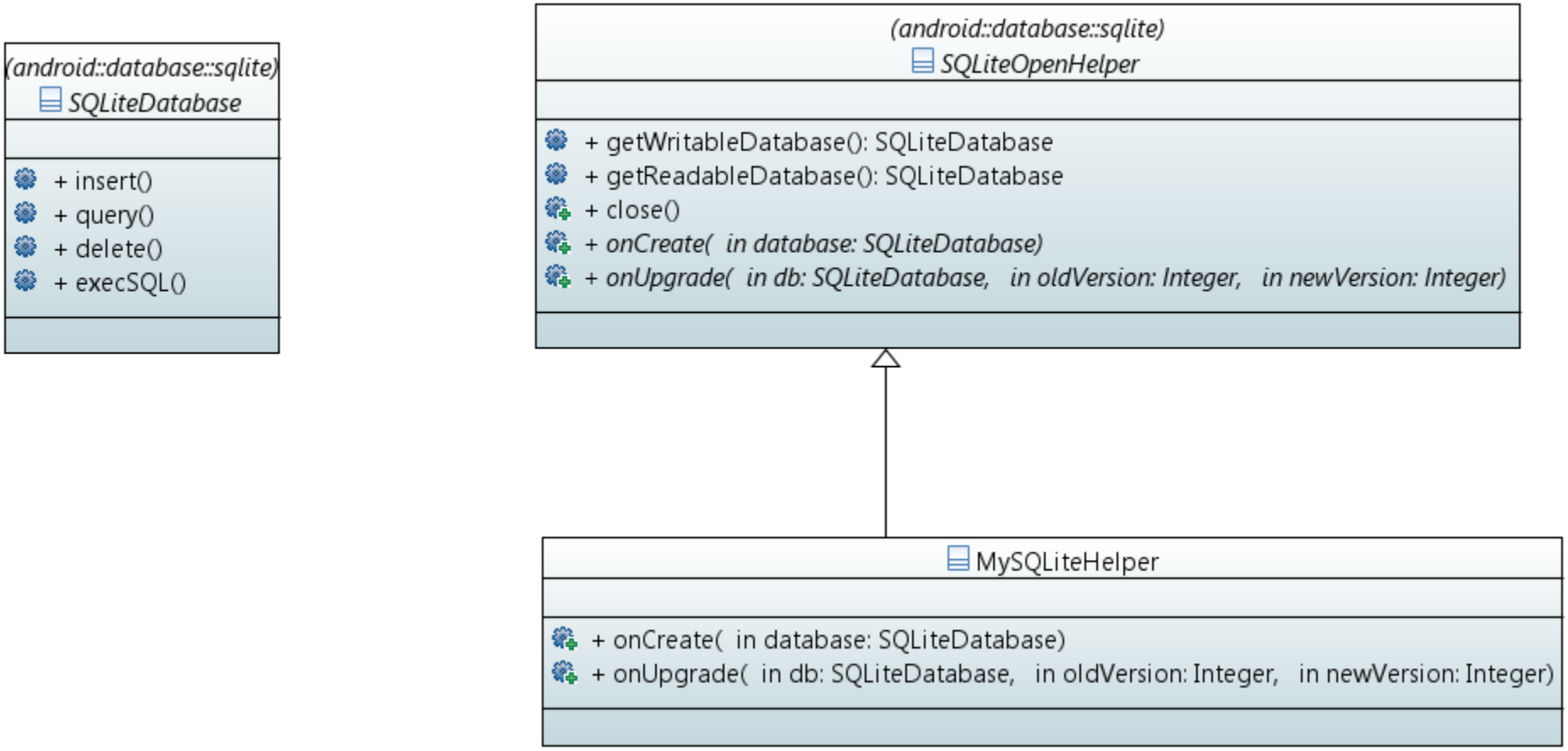
SQLite et Android

- Intégré dans chaque appareil Android
- Nécessite l'accès au système de fichier
  - La BD est sauvée dans un fichier
  - `/data/APP_NAME/databases/FILENAME`
    - `DATA` = chemin retourné par la méthode `Environment.getDataDirectory()`,
    - `APP_NAME` : nom de l'application.
    - `FILENAME` : nom de la base de données.

Imports

- `import android.database;`
  - // contient toutes les classes nécessaires pour travailler avec des bases de données.
- `import  android.database.sqlite;`
  - // contient les classes spécifiques à SQLite.

Les Classes de base



Les Classes de base

- SQLiteDatabase
  - Objet représentant la connexion à la DB
  - Permet la création des tables
  - Permet la création, la modification et la suppression de données
- SQLiteOpenHelper
  - Classe abstraite à sous-classer
  - Permet la gestion de la création de la DB
  - Permet d'obtenir SQLiteDatabase
- MySQLiteHelper
  - Classe à fournir (extends SQLiteOpenHelper)
  - Implémente les méthodes
    - pour créer les table dans la DB
    - Pour mettre à jour la DB suite à un upgrade

Cycle d'utilisation

- Ouvrir la connection à la DB
  - Cela peut entrainer la création des tables
- Accéder à la DB
- Fermer la connection à la DB

```
// Context is the calling activity
Activity context /*= this*/;

// open() - Create helper and open database
MySQLiteHelper dbHelper  = new MySQLiteHelper(context);
SQLiteDatabase database = dbHelper.getWritableDatabase();

// Access to DB HERE
//...

// close() - Close helper
dbHelper.close();
```

Créer la table
Dans MySQLiteHelper

```
public class MySQLiteHelper extends SQLiteOpenHelper {

    public static final String TABLE_COMMENTS = "comments";
    public static final String COLUMN_ID = "_id";
    public static final String COLUMN_COMMENT = "comment";

    private static final String DATABASE_NAME = "commments.db";
    private static final int DATABASE_VERSION = 1;
    // Commande sql pour la création de la base de données
    private static final String DATABASE_CREATE = "create table "
            + TABLE_COMMENTS + "(" + COLUMN_ID
            + " integer primary key autoincrement, " + COLUMN_COMMENT
            + " text not null);";

    public MySQLiteHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }
    @Override
    public void onCreate(SQLiteDatabase database) {
        database.execSQL(DATABASE_CREATE);
    }
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        Log.w(MySQLiteHelper.class.getName(),
                "Upgrading database from version " + oldVersion + " to "
                        + newVersion + ", which will destroy all old data");
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_COMMENTS);
        onCreate(db);
    }
}
```

Sauvegarder des données

- Créer une map (nomColonne, valeur)
- Insérer la map

```
static String TABLE_COMMENTS = "COMMENT_TABLE";
static String COLUMN_COMMENT = "COMMENT";
static String COLUMN_NAME = "NAME";

String comment = "UneValeur";

// Crée un objet ContentValue utilisé pour transporter des valeurs
ContentValues values = new ContentValues();
values.put(COLUMN_COMMENT, comment);
values.put(COLUMN_COMMENT, comment);
// Insert la nouvelle valeur dans la DB. Recupere l'ID en retour
long insertId = database.insert(TABLE_COMMENTS, null, values);
```

Recuperer des données

- Faire une query
- Parcourir le résultat ligne par ligne (avec le curseur)
  - Extraire les données de la ligne
- Fermer le curseur

```
public void readData() {
    Cursor cursor = database.query(TABLE_COMMENTS,
            {COLUMN_COMMENT, COLUMN_NAME},
                 null, null, null, null, null);

    cursor.moveToFirst();
    while (!cursor.isAfterLast()) {
        long id = cursor.getLong(0);
        String comment = cursor.getString(1);
        String name = cursor.getString(2);
    }
    // assurez-vous de la fermeture du curseur
    cursor.close();
}
```

Supprimer des données

```
long id = 123; // L'id de la ligne à supprimer

System.out.println("Comment deleted with id: " + id);
database.delete(TABLE_COMMENTS, COLUMN_ID + " = " + id, null);
```

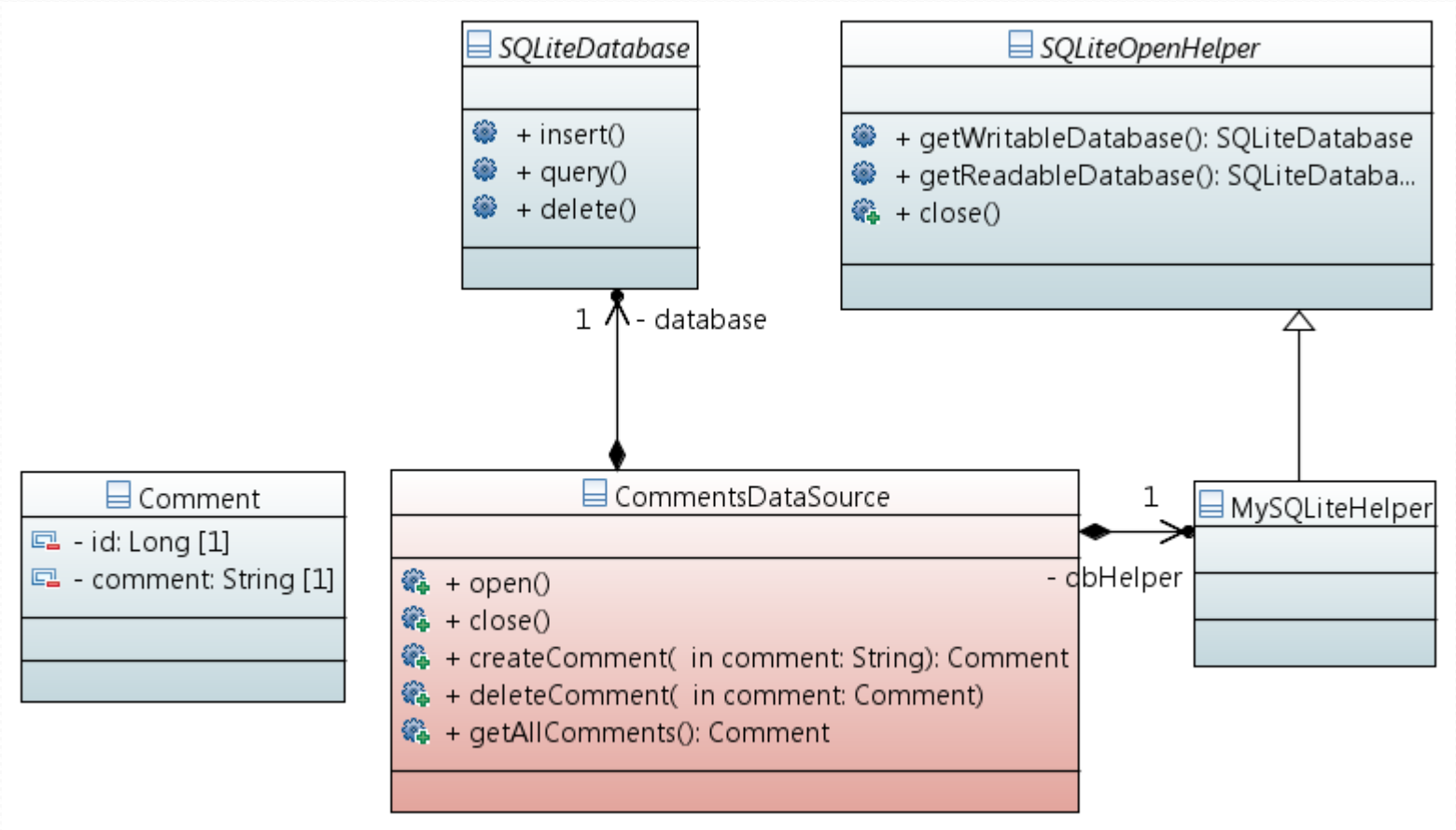
Utiliser un DAO

Data Access Object (DAO)

- Classe intermédiaire
  - entre le monde objet et la BD relationnel
- Prend en charge la traduction Objet ⇔Table
-

Exemple CommentDataSource

- Toutes les demandes passes par le dao



Cycle de vie

- Créer le DAO (dans onCreate())
- Acceder aux données (dans les méthodes)
- Fermer le DAO (dans onPause() ou onDestroy())

```
// Create DAO ()
datasource = new CommentsDataSource(context);
// Open DB connexion
datasource.open();

 // Access DB
List<Comment> values = datasource.getAllComments();

datasource.close();
```

Classe DAO

```
public class CommentsDataSource {
    // Champs de la base de données
    private SQLiteDatabase database;
    private MySQLiteHelper dbHelper;
    private String[] allColumns = {MySQLiteHelper.COLUMN_ID,
            MySQLiteHelper.COLUMN_COMMENT};

    /**
     * Constructor
     *
     */
    public CommentsDataSource(Context context) {
        dbHelper = new MySQLiteHelper(context);
    }

    public void open() throws SQLException {
        database = dbHelper.getWritableDatabase();
    }

    public void close() {
        dbHelper.close();
    }
…
```

Classe DAO (1)

```
/**
 * Crée un nouvel objet en BD
 *
 * @param comment La valeur à inséré dans le nouvel objet
 * @return Un objet du type demandé, peuplé par les valeurs de la table.
 */
public Comment createComment(String comment) {
    // Crée un objet ContentValue utilisé pour transporter des valeurs
    ContentValues values = new ContentValues();
    values.put(MySQLiteHelper.COLUMN_COMMENT, comment);
    // Insert la nouvelle valeur dans la DB. Recupere l'ID en retour
    long insertId = database.insert(MySQLiteHelper.TABLE_COMMENTS, null,
values);

    // Demande la nouvelle ligne de la table
    Cursor cursor = database.query(MySQLiteHelper.TABLE_COMMENTS,
            allColumns, MySQLiteHelper.COLUMN_ID + " = " + insertId, null,
            null, null, null);
    cursor.moveToFirst();
    // Crée un objet du type demandé à partir de la ligne.
    Comment newComment = cursorToComment(cursor);
    cursor.close();
    return newComment;
}
```

Classe DAO (3)

```
    public void deleteComment(Comment comment) {
        long id = comment.getId();
        System.out.println("Comment deleted with id: " + id);
        database.delete(MySQLiteHelper.TABLE_COMMENTS, MySQLiteHelper.COLUMN_ID
                + " = " + id, null);
    }

    public List<Comment> getAllComments() {
        List<Comment> comments = new ArrayList<Comment>();

        Cursor cursor = database.query(MySQLiteHelper.TABLE_COMMENTS,
                allColumns, null, null, null, null, null);

        cursor.moveToFirst();
        while (!cursor.isAfterLast()) {
            Comment comment = cursorToComment(cursor);
            comments.add(comment);
            cursor.moveToNext();
        }
        // assurez-vous de la fermeture du curseur
        cursor.close();
        return comments;
    }

    private Comment cursorToComment(Cursor cursor) {
        Comment comment = new Comment();
        comment.setId(cursor.getLong(0));
        comment.setComment(cursor.getString(1));
        return comment;
    }
}
```

Mapping Objet - Relationnel

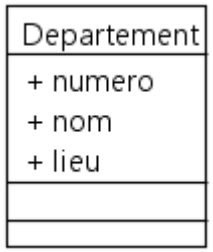
Le problème

- Comment
  - passer du monde objet au monde relationnel ?

Objet	Relationnel
Class	Table
Propriété	Colonne
Association	Identifiant
Héritage	Join

Equivalence classe - table

- En général :
  - Une classe ⇔ une table
  - Une propriété ⇔ une colonne
  - Un objet == une ligne dans la table
  - Association == clé et clé étrangère
- Exemple : la classe Département est traduite par la table
  - **DÉPARTEMENT(numéro, nom, lieu)**

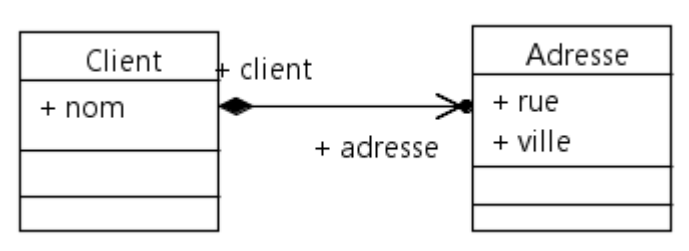


	DEPARTEMENT	
numero	nom	lieu
59	Nord	
62	Pas de Calais	
33	Gironde	
40	Landes	

objet

Exemple embarqué

- Une classe **Adresse** peut ne pas avoir de correspondance sous la forme d'une table séparée dans le modèle relationnel
- Les attributs de la classe **Adresse** sont intégrés dans la table qui représente la classe **Client**
- Les objets de la classe **Adresse** n'ont pas d'identification liée à la base de données



	Client		
id	client_nom	adr_rue	adr_ville

ORMLite

ORMLite

- http://ormlite.com/sqlite_java_android_orm.shtml

- ORM = Object Relationnal Mapping
- Fournit un ORM pour SQLite et Android
  - Support les annotations
    - Native
    - Ou JPA (standard persistance et ORM)

Principe

- Déclarer des classes ‘entity’ et les annoter
  - Générer le fichier de config correspondant
- Déclarer une classe helper
- Déclarer des DAO
  - En utilisant

- Utiliser les DAO

Ajouter les jars dans votre projet

- ORMLite necessite 2 jars:
  - ormlite-android-x.jar
  - ormlite-core-x.jar
- You may download the latest copies either from the ***libs*** directory of the attached source code or from the below URL:
  - http://sourceforge.net/projects/ormlite/files/releases/com/j256/ormlite/
- You may download both of them and put into the "**libs**" directory of the project.

Déclarer des classes ‘entity’

```
public class TeacherDetails implements Serializable {

    // Primary key defined as an auto generated integer
// If the database table column name differs than
// the Model class variable name, the way to map to use columnName
    @DatabaseField(generatedId = true, columnName = "teacher_id")
    protected int teacherId;

    // Define a String type field to hold teacher's name
    @DatabaseField(columnName = "teacher_name")
    protected String teacherName;
    // Define a String type field to hold student's address
    public String address;

    // Default constructor is needed for the SQLite, so make sure you also have it
    public TeacherDetails() {
    }

    //For our own purpose, so it's easier to create a TeacherDetails object
    public TeacherDetails(final String name, final String address) {
        this.teacherName = name;
        this.address = address;
    }
  // getters / setters
}
```

Générer le fichier de config correspondant

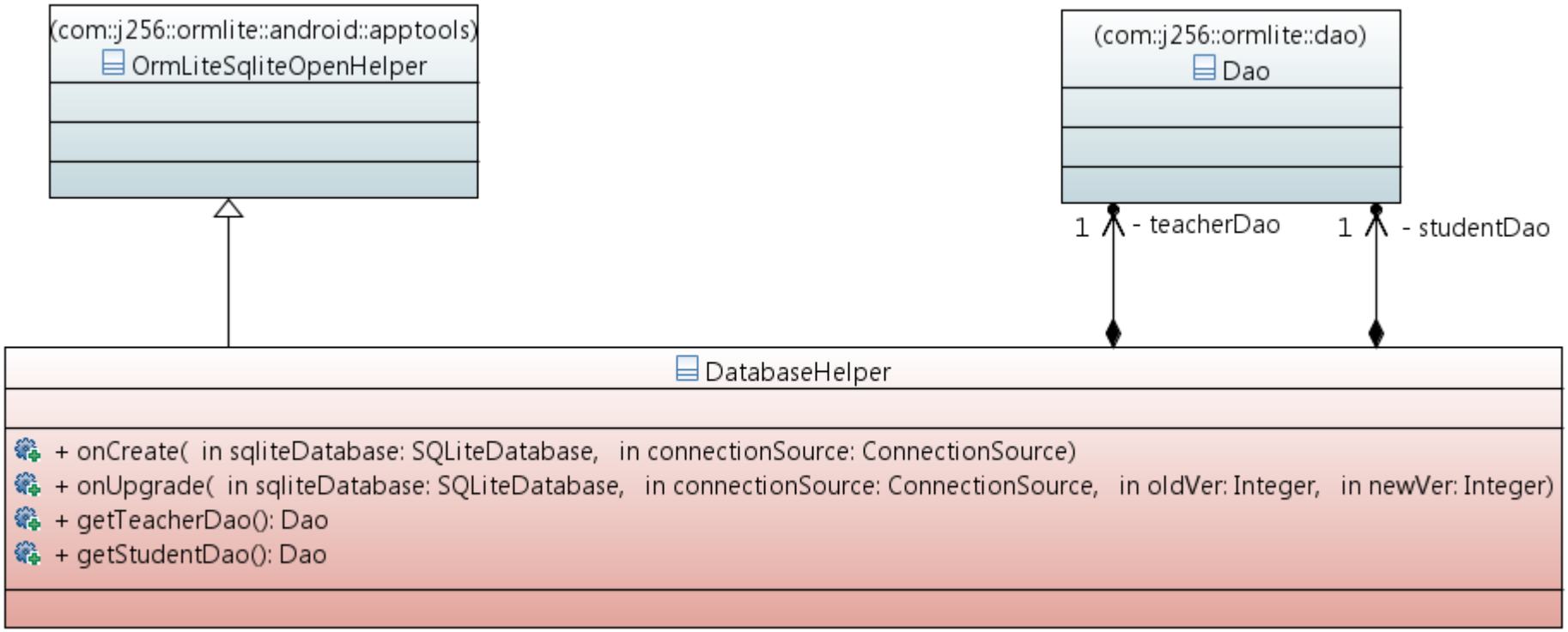
- `ormlite_config.txt`
  - Dans `res/raw`
- Soit à la main
- Soit en utilisant une méthode java
  -

```
import com.j256.ormlite.android.apptools.OrmLiteConfigUtil;


public class DatabaseConfigUtil extends OrmLiteConfigUtil {

    public static void main(String[] args) throws SQLException, IOException {
// Provide the name of .txt file which you have already created and kept in res/raw directory
        writeConfigFile("ormlite_config.txt");
    }
}
```

res/raw ormlite_config.txt

```
# --table-start--
dataClass=pje15.studentdirectory.data.TeacherDetails
tableName=teacher_details
# --table-fields-start--
# --field-start--
fieldName=teacherId
columnName=teacher_id
generatedId=true
# --field-end--
# --field-start--
fieldName=teacherName
columnName=teacher_name
# --field-end--
# --field-start--
fieldName=address
# --field-end--
# --table-fields-end--
# --table-end--
#################################
```

Déclarer une classe helper



Déclarer une classe helper

```
/**
 * Database helper which creates and upgrades the database and provides the DAOs for the app.
 */
public class DatabaseHelper extends OrmLiteSqliteOpenHelper {

    /************************************************
     * Suggested Copy/Paste code. Everything from here to the done block.
     ************************************************/

    private static final String DATABASE_NAME = "studentdir.db";
    private static final int DATABASE_VERSION = 1;

    private Dao<StudentDetails, Integer> studentDao;
    private Dao<TeacherDetails, Integer> teacherDao;

    public DatabaseHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION, R.raw.ormlite_config);
    }
```

Déclarer une classe helper (2)

```
    @Override
    public void onCreate(SQLiteDatabase sqliteDatabase, ConnectionSource connectionSource) {
        try {
// Create tables. This onCreate() method will be invoked only once of the application life time
i.e. the first time when the application starts.
            TableUtils.createTable(connectionSource, TeacherDetails.class);
            TableUtils.createTable(connectionSource, StudentDetails.class);
        } catch (SQLException e) {
            Log.e(DatabaseHelper.class.getName(), "Unable to create datbases", e);
        }
    }
    @Override
    public void onUpgrade(SQLiteDatabase sqliteDatabase, ConnectionSource connectionSource, int
oldVer, int newVer) {
        try {
// In case of change in database of next version of application, please increase the value of
DATABASE_VERSION variable, then this method will be invoked
//automatically. Developer needs to handle the upgrade logic here, i.e. create a new table or a
new column to an existing table, take the backups of the
// existing database etc.
            TableUtils.dropTable(connectionSource, TeacherDetails.class, true);
            TableUtils.dropTable(connectionSource, StudentDetails.class, true);
            onCreate(sqliteDatabase, connectionSource);
        } catch (SQLException e) {
            Log.e(DatabaseHelper.class.getName(), "Unable to upgrade database from version " +
oldVer + " to new "
                    + newVer, e);
        }
    }
```

Déclarer une classe helper (3)

```
    public Dao<TeacherDetails, Integer> getTeacherDao() throws SQLException {
        if (teacherDao == null) {
            teacherDao = getDao(TeacherDetails.class);
        }
        return teacherDao;
    }

    public Dao<StudentDetails, Integer> getStudentDao() throws SQLException {
        if (studentDao == null) {
            studentDao = getDao(StudentDetails.class);
        }
        return studentDao;
    }
}
```

Utiliser dans une activity

```
public class TeacherAddACtivity extends AppCompatActivity implements OnClickListener {

    // Reference of DatabaseHelper class to access its DAOs and other components
    private DatabaseHelper databaseHelper = null;

    // This is how, DatabaseHelper can be initialized for future use
    private DatabaseHelper getHelper() {
        if (databaseHelper == null) {
            databaseHelper = OpenHelperManager.getHelper(this,DatabaseHelper.class);
        }
        return databaseHelper;
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();

/*
* You'll need this in your class to release the helper when done.
*/
        if (databaseHelper != null) {
            OpenHelperManager.releaseHelper();
            databaseHelper = null;
        }
    }
```

Stocker un objet

```
                TeacherDetails techDetails = new TeacherDetails();

                try {
// This is how, a reference of DAO object can be done
                    final Dao<TeacherDetails, Integer> techerDao = getHelper().getTeacherDao();
//This is the way to insert data into a database table
                    techerDao.create(techDetails);

                } catch (SQLException e) {
                    e.printStackTrace();
                }
```

Récupérer tout les objets d'une classe

```
            Dao<TeacherDetails, Integer> teacherDao;
            teacherDao = getHelper().getTeacherDao();
// Query the database. We need all the records so, used queryForAll()
          teacherList = teacherDao.queryForAll();
```

Faire une query (apperçu)

```
        try {
// This is how, a reference of DAO object can be done
            studentDao =  getHelper().getStudentDao();
// Get our query builder from the DAO
            final QueryBuilder<StudentDetails, Integer> queryBuilder = studentDao.queryBuilder();
// We need only Students who are associated with the selected Teacher, so build the query by
"Where" clause
            queryBuilder.where().eq(StudentDetails.TEACHER_ID_FIELD, tDetails.teacherId);
// Prepare our SQL statement
            final PreparedQuery<StudentDetails> preparedQuery = queryBuilder.prepare();
// Fetch the list from Database by queryingit
            final Iterator<StudentDetails>  studentsIt =
                        studentDao.query(preparedQuery).iterator();
// Iterate through the StudentDetails object iterator and populate the comma separated String
            while (studentsIt.hasNext()) {
                final StudentDetails sDetails = studentsIt.next();
                String name = sDetails.studentName;
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
```

Atelier

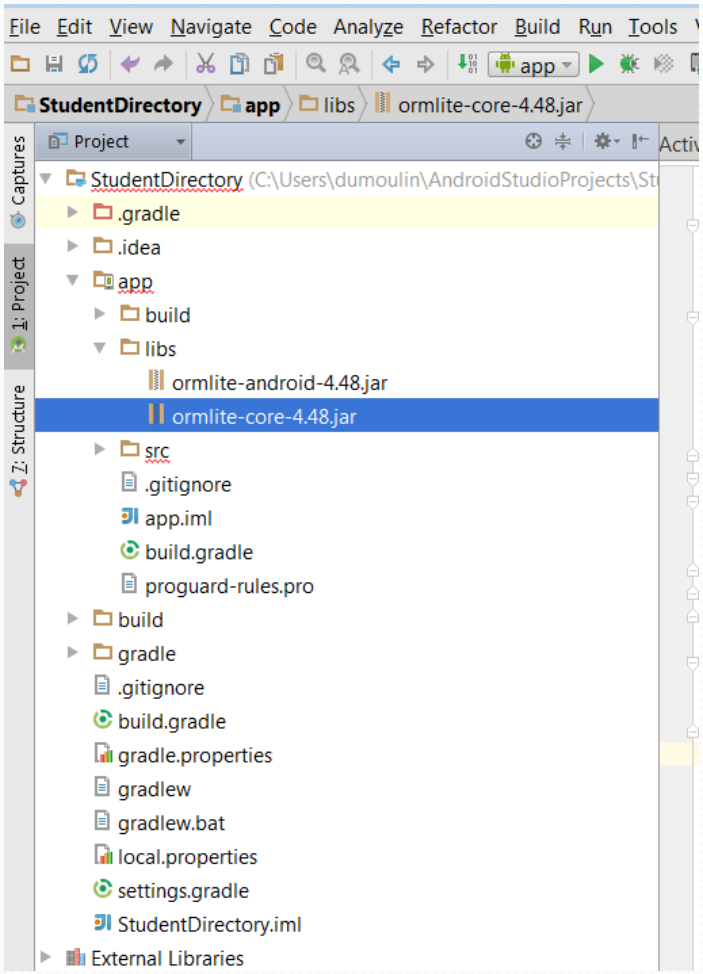
- Vous devez maintenant utiliser la BD SQLite dans votre projet.
- Il est recommandé d'utiliser aussi ORMLite
- Vous pouvez vous exercer en faisant le tutorial:
  - http://www.androidbegin.com/tutorial/android-ormlite-with-sqlite-database-tutorial/

AndroidStudio : Ajouter des Jars

AndroidStudio : Ajouter des Jar

1) Mettre les jars dans le projet

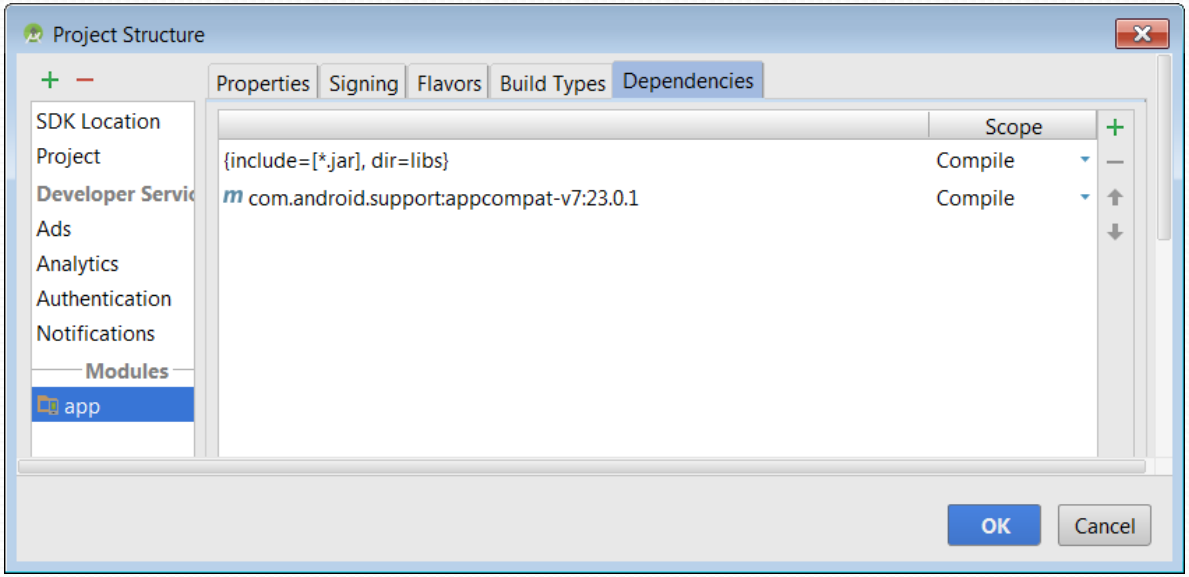
- Mettre les jars dans le projet
- Ouvrir Le projet a partir de ‘projet’
- Déposer les jar dans le répertoire app/libs



AndroidStudio : Ajouter des Jar

2) Ouvrir la fenetre ‘dependencies’

- Dans l’explorateur :
  - Click droit -> Open Module Settings -> + -> File dependencies



AndroidStudio : Ajouter des Jar
3) Ajouter les jars

- Selectionner les jars
- OK