# Création d'un embryon de gestionnaire de base de données SQLite sous Android

Création de 2 packages dans le projet :

- Un package de classes qui vont modélisent les données « dataobject »
- Un package de classes qui vont gérer les données dans la base SQL « dataaccess »

Cette structure modulaire permet une réutilisation plus facile dans différents projets et une maintenance plus facile.

## Définitions de classe pour stocker les objets JAVA et pour y accéder :

Dans dataobject, c'est là que nous allons définir les classes pour stocker les objets.

- Nous allons d'abord créer une classe abstraite d'objet très générique « Pojo » pour « Plain Old Java Object » (cad « bon vieil objet JAVA »).
    - Pour quoi « abstraite », parce qu'on n'accèdera jamais directement à ces objets.
    - Dedans nous allons définir
        - uniquement un attribut « id » en guise d'identifiant
        - 2 méthodes pour lire et affecter un identifiant
        - Un constructeur minimaliste

```
public abstract class Pojo {

    //Atributs
    private int id;

    //Getters / setters
    public int getId(){return id;};
    public void setId(int id) {this.id = id;};

    //constructeur
    public Pojo(int id){
        this.id = id;
    };


}
```

- Ensuite, nous pouvons créer des classes qui vont étendre cette classe "Pojo" (cad qui vont hériter des attributs et méthodes déjà définies dans la classe Pojo)

    En guise d'exemple nous allons créer une classe « Genre_musical » destinée à lister différents genre musicaux et une classe « Disque » destinée à stocker des informations sur les disques.

- Créer une classe « Genre_musical », avec un champ « Nom », des méthodes de lecture/écrite de ce champ, ainsi que différents constructeurs.

```
package dataobject;

/**
 * Created by HQuinquenel on 22/11/2016.
 */
public class Genre_musical extends Pojo {

    private String nom;

    //Getters / setters
    public String getNom(){return nom;};
    public void setNom(String nom) {this.nom = nom;};

    //constructeurs
    public Genre_musical(int id){
        super(id);
    };
    public Genre_musical(int id, String nom){
        super(id);
        this.nom = nom;

    };
}
```

- Créer une classe « Disque », avec les champs « nom », « artiste » ainsi que « idGenre », qui sera un identifiant assurant la correspondance avec un objet de la classe « Genre_musical ».
- Ajouter les méthodes de lecture/écriture des champs et des constructeurs.
- Enfin, par confort, et pour inspecter facilement ce qu'il y aura dans les objets, ajouter une méthode « toString » qui viendra surcharger (remplacer) la méthode « toString » générique des classes d'objets JAVA.

```
package dataobject;

/**
 * Created by HQuinquenel on 22/11/2016.
 */
public class Disque extends Pojo {

    private String nom;
    private String artiste;
    private int idGenre;

    //Getters / setters
    public String getNom(){return nom;};
    public void setNom(String nom) {this.nom = nom;};
    public String getArtiste(){return artiste;};
    public void setArtiste(String nom) {this.artiste = artiste;};
```

```
    public int getidGenre(){return idGenre;};
    public void setidGenre(int idGenre) {this.idGenre = idGenre;};

    //constructeurs
    public Disque(int id){
        super(id);
    };
    public Disque(int id, String nom, int idGenre){
        super(id);
        this.nom = nom;
        this.idGenre = idGenre;
    };
    public Disque(int id, String nom, String artiste, int idGenre){
        super(id);
        this.nom = nom;
        this.artiste = artiste;
        this.idGenre = idGenre;
    };

    //methods
    @Override
    public String toString(){
        return "ID Disque :" + getId() + " Nom : " + getNom() + " Artiste :
" + getArtiste() +
                " ID Genre : " + getidGenre();
    }

}
```

Une fois ces classes créées, vous avez donc la structure de données et les méthodes d'accès côté JAVA. Reste donc à faire le lien avec le gestionnaire de base de données (et à tester que ça fonctionne sur un jeu de données)

**Définitions de classe pour stocker les objets SQLite et pour y accéder :**

Dans le pakage « dataaccess », c'est là que nous allons définir les classes pour remplir et accéder aux données de la base SQLite.

- D'abord, nous allons créer une classe « DBHelper » qui va nous aider à créer/supprimer la base de données. Cette classe hérite d'une classe « SQLiteOpenHelper » et crée les tables de notre base de données. Par soucis de cohérence, on essaiera de donner au nom des champs de la base SQLite le même que celui des objets JAVA.

  *Attention, une requête SQL mal construite (par exemple une simple virgule en trop) fera planter votre application avant même l'affichage de l'interface !*

```
package dataaccess;

import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

/**
 * Created by prof on 17/03/15.
 */
public class DBHelper extends SQLiteOpenHelper {

    //utilisation du modèle de données



    public static final String DB_NAME = "base_de_donnees_musique.db";
    public static final int DB_VERSION = 1;


    //constructor
    public DBHelper(Context context){
        super(context,DB_NAME,null,DB_VERSION);
    }


    public static String getQueryCreate(){
        return "CREATE TABLE Disque("
                + "id Integer PRIMARY KEY AUTOINCREMENT, "
                + "nom Text NOT NULL,"
                + "artiste Text NOT NULL,"
                + "idGenre Integer NOT NULL"
                + ");" ;
    }

    public static String getQueryDrop(){
        return "DROP TABLE IF EXISTS Disque;";
    }

    @Override
    public void onCreate(SQLiteDatabase db){
        //ceci est automatiquement géré par SQLite
        db.execSQL(getQueryCreate());
```

```
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int
newVersion){
        db.execSQL(getQueryDrop());
        db.execSQL(getQueryCreate());
    }
}
```

*On pourra rajouter la création de la table "Genre_musical"*

- Maintenant, nous allons créer une classe « DataSource» qui va juste effectuer les connexions à la base de données et définir les méthodes pour ouvrir/fermer la BDD. Par confort, elle créera une méthode « newDisqueDataAccessObject» qui instanciera un nouvel objet de classe « DisqueDataAccessObject » (voir la suite du TP)

```
package dataaccess;

import android.content.Context;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;

/**
 * Created by prof on 17/03/15.
 */
public class DataSource {

    //connexion à la base de données


    private SQLiteDatabase db;
    private final DBHelper helper;

    public DataSource(Context context){
        helper = new DBHelper(context);
    }

    public SQLiteDatabase getDB(){

        if (db == null) open ();
        return db;

    }

    public void open() throws SQLException {
        db = helper.getWritableDatabase();
    }

    public void close(){
        helper.close();

    }

    //factory
    public DisqueDataAccessObject newDisqueDataAccessObject(){
```

```
        return new DisqueDataAccessObject(this);
    }
}
```

- Ensuite nous allons juste créer cette classe « DisqueDataAccessObject » qui va nous aider à :
    - Prendre des objets JAVA et les sauvegarder dans des enregistrements de la base SQLite
    - Prendre des enregistrements de la base SQLite et les transférer dans des objets Java,

  le tout via des objets temporaires.

Dans cette classe, nous allons définir des champs correspondant aux colonnes de la table SQLite. Attention, il faut que l'orthographe corresponde exactement aux noms de colonne définis dans le « CREATE TABLE » de la classe « DBHelper » !

Pourquoi « final » ? : pour indiquer que les valeurs d'attributs des objets instanciés ne pourront jamais être changés dans la suite du programme ; ils deviennent donc de fait des constantes.

Pourquoi « static » ? : pour indiquer que la valeur est commune (et donc toujours exactement la même, et pas une copie identique) pour tous les objets dans la classe qui seront instanciés dans le programme. (…bon comme ça on a vraiment blindé le truc !)

```
public static final String COL_ID="id";
public static final String COL_NOM="nom";
public static final String COL_ARTISTE="artiste";
public static final String COL_IDGENRE="idGenre";
public static final String TABLE_NAME="Disque";
```

On rajoutera un champ, et un constructeur pour permettre de manipuler facilement la base de données directement via les objets :

```
private final DataSource datasource;

//constructor
public DisqueDataAccessObject(DataSource datasource){
    this.datasource = datasource;
}
```

- Ecriture d'une méthode qui prend un objet JAVA « Disque » et qui le sauvegarde dans la base de données (INSERT).

*Pourquoi « synchronised » ? : pour résumer afin d'éviter les phénomènes de concurrence d'accès sur les requêtes de la base de données (cf : [http://blog.paumard.org/cours/java-api/chap05-concurrent-synchronisation.html](http://blog.paumard.org/cours/java-api/chap05-concurrent-synchronisation.html))*

```
public synchronized Disque insert(Disque mObjet){

    //on copie les champs de l'objet dans les colonnes de la table.
    ContentValues values=new ContentValues();

    values.put(COL_NOM, mObjet.getNom());
    values.put(COL_ARTISTE, mObjet.getArtiste());
    values.put(COL_IDGENRE, mObjet.getIdGenre());

    //insert query
    int id=(int)datasource.getDB().insert(TABLE_NAME,null,values);

    //mise à jour de l'ID dans l'objet
    mObjet.setId(id);

    return mObjet;


}
```

- Ecriture d'une méthode qui va modifier un enregistrement de la base de données (UPDATE) à partir des informations d'un objet JAVA « Disque ». Cet enregistrement sera identifié grâce à sa colonne « id » et donc nous allons gérer une clause WHERE sur cette colonne.

```
public synchronized Disque update(Disque mObjet){

    //on copie les champs de l'objet dans les colonnes de la table.
    ContentValues values=new ContentValues();
    values.put(COL_ID, mObjet.getId());
    values.put(COL_NOM, mObjet.getNom());
    values.put(COL_ARTISTE, mObjet.getArtiste());
    values.put(COL_IDGENRE, mObjet.getidGenre());

    //gestion de la clause "WHERE"
    String clause = COL_ID + " = ? ";
    String[] clauseArgs = new String[]{
            String.valueOf(mObjet.getId())
    };

    datasource.getDB().update(TABLE_NAME, values, clause, clauseArgs);

    //mise à jour de l'ID dans l'objet
    return mObjet;


}
```

- Ecriture d'une méthode qui va supprimer un enregistrement de la base de données (delete ) à partir des informations d'un objet JAVA « Disque ». Cet enregistrement sera aussi identifié grâce à sa colonne « id » et donc nous allons gérer une clause WHERE sur cette colonne.

```
public synchronized void delete(Disque mObjet){
```

```
    //gestion de la clause "WHERE"
    String clause = COL_ID + " = ? ";
    String[] clauseArgs = new String[]{
            String.valueOf(mObjet.getId())
    };

    datasource.getDB().delete(TABLE_NAME, clause, clauseArgs);

}
```

- Ecriture d'une méthode qui va lire un enregistrement de la base de données et les copier dans les champs d'un objet JAVA « Disque ». Cet enregistrement sera aussi identifié grâce à sa colonne « id » et donc nous allons gérer une clause WHERE sur cette colonne. Le résultat de la requête sera récupéré dans un objet intermédiaire de classe « Cursor » puis transféré dans l'objet de classe « Disque ».

  *On prendra bien soin d'adapter la méthode de récupération de la valeur stockée dans l'objet « cursor » en fonction de la nature du champ concerné (getInt, getString, etc.) et ceci en respectant bien l'ordre des colonnes.*

```
public Disque read(Disque mObjet){


    //columns
    String[] allColumns = new
String[]{COL_ID,COL_NOM,COL_ARTISTE,COL_IDGENRE};


    //clause
    String clause = COL_ID + " = ? ";
    String[] clauseArgs = new String[]{
            String.valueOf(mObjet.getId())
    };


    //select query
    Cursor cursor = datasource.getDB().query(TABLE_NAME,allColumns, "ID =
?", clauseArgs, null, null,null);

    //read cursor. On copie les valeurs de la table dans l'objet
    cursor.moveToFirst();
    mObjet.setId(cursor.getInt(0));
    mObjet.setNom(cursor.getString(1));
    mObjet.setArtiste(cursor.getString(2));
    mObjet.seIdGenre(cursor.getInt(3));
    cursor.close();

    return mObjet;


}
```

- Ecriture d'une méthode qui va lire tous les enregistrements de la base de données et les mettre dans une liste

*On prendra aussi bien soin d'adapter la méthode de récupération de la valeur stockée dans l'objet « cursor » en fonction de la nature du champ concerné (getInt, getString, etc.) et ceci en respectant bien l'ordre des colonnes.*

```
public List<Disque> readAll(){


    //columns
    String[] allColumns = new
String[]{COL_ID,COL_NOM,COL_ARTISTE,COL_IDGENRE};


    //select query
    Cursor cursor = datasource.getDB().query(TABLE_NAME,allColumns, null,
null, null, null,null);

    //Iterate on cursor and retrieve result
    List<Disque> liste_disque = new ArrayList<Disque>();


    cursor.moveToFirst();

    while (!cursor.isAfterLast()){

        liste_disque.add(new Disque(cursor.getInt(0), cursor.getString(1),
cursor.getString(2), cursor.getInt(3)));
        cursor.moveToNext();

    }

    cursor.close();

    return liste_disque;


}
```

VOILA ! Toute l'ossature est maintenant prête pour accueillir vos données et les manipuler.

Donc, pour finir créez une interface sous forme de formulaire permettant de remplir la base de données, de la gérer et de toujours contrôler que le résultat corresponde bien à l'action voulue.

En guise d'exemple, voici le code de la méthode « saveDisque» :

```
public void saveDisque(View v){
    Disque mDisque1=new Disque(-1, this.mEditText1.getText().toString(),
this.mEditText2.getText().toString(),
Integer.parseInt(this.mEditText3.getText().toString())); //L'identifiant "-
1" dit à SQLite de créer un nouvel identifiant en autoincrémentation
    Log.w("CONTROLE", mDisque1.toString());


    //stockage des attributs de l'objet dans la base de données
    mDisqueDataAccessObject.insert(mDisque1);

    //controle de l'écriture dans la base de données
    List<Disque> liste_disque = mDisqueDataAccessObject.readAll();
    for (Disque mDisque : liste_disque){
        Log.w("CONTROLE ", mDisque.toString() + "\n");
    }



}
```

## Pour aller plus loin…

Nous allons ajouter les fonctions pour gérer la table « Genre_musical » à la base SQL et la lier au formulaire pour que la saisie de cette information soit faite à partir d'un menu déroulant automatiquement peuplé.

- Création d'une classe « Genre_musicalDataAccessObject », calquée sur « DisqueDataAccessObject »

```
package dataaccess;

import android.content.ContentValues;
import android.database.Cursor;

import java.util.ArrayList;
import java.util.List;

import dataobject.Genre_musical;

/**
 * Created by HQuinquenel on 23/11/2016.
 */

public class Genre_musicalDataAccessObject {

    public static final String COL_ID="id";
    public static final String COL_NOM="nom";
    public static final String TABLE_NAME="Genre_musical";

    private final DataSource datasource;

    //constructor
    public Genre_musicalDataAccessObject(DataSource datasource){
        this.datasource = datasource;
    }

    public synchronized Genre_musical insert(Genre_musical mObjet){

        //on copie les champs de l'objet dans les colonnes de la table.
        ContentValues values=new ContentValues();

        values.put(COL_NOM, mObjet.getNom());

        //insert query
        int id=(int)datasource.getDB().insert(TABLE_NAME,null,values);

        //mise à jour de l'ID dans l'objet
        mObjet.setId(id);

        return mObjet;


    }

    public synchronized Genre_musical update(Genre_musical mObjet){
```

```
        //on copie les champs de l'objet dans les colonnes de la table.
        ContentValues values=new ContentValues();
        values.put(COL_ID, mObjet.getId());
        values.put(COL_NOM, mObjet.getNom());

        //gestion de la clause "WHERE"
        String clause = COL_ID + " = ? ";
        String[] clauseArgs = new String[]{
                String.valueOf(mObjet.getId())
        };

        datasource.getDB().update(TABLE_NAME, values, clause, clauseArgs);

        //mise à jour de l'ID dans l'objet
        return mObjet;


    }

    public synchronized void delete(Genre_musical mObjet){


        //gestion de la clause "WHERE"
        String clause = COL_ID + " = ? ";
        String[] clauseArgs = new String[]{
                String.valueOf(mObjet.getId())
        };

        datasource.getDB().delete(TABLE_NAME, clause, clauseArgs);

    }

    public Genre_musical read(Genre_musical mObjet){


        //columns
        String[] allColumns = new String[]{COL_ID,COL_NOM};


        //clause
        String clause = COL_ID + " = ? ";
        String[] clauseArgs = new String[]{
                String.valueOf(mObjet.getId())
        };


        //select query
        Cursor cursor = datasource.getDB().query(TABLE_NAME,allColumns, "ID
= ?", clauseArgs, null, null,null);

        //read cursor. On copie les valeurs de la table dans l'objet
        cursor.moveToFirst();
        mObjet.setId(cursor.getInt(0));
        mObjet.setNom(cursor.getString(1));
        cursor.close();

        return mObjet;


    }
```

```
    public List<Genre_musical> readAll(){


        //columns
        String[] allColumns = new String[]{COL_ID,COL_NOM};


        //select query
        Cursor cursor = datasource.getDB().query(TABLE_NAME,allColumns,
null, null, null, null,null);

        //Iterate on cursor and retrieve result
        List<Genre_musical> liste_genre = new ArrayList<Genre_musical>();


        cursor.moveToFirst();

        while (!cursor.isAfterLast()){

            liste_genre.add(new Genre_musical(cursor.getInt(0),
cursor.getString(1)));
            cursor.moveToNext();

        }

        cursor.close();

        return liste_genre;


    }



}
```

- Dans la classe DB_Helper, précisez que la méthode « getQueryCreate » actuelle ne s'applique quà la table « Disque » (dans le nom de la méthode)

```
   public static String getQueryCreate_Disque(){
       return "CREATE TABLE Disque("
               + "id Integer PRIMARY KEY AUTOINCREMENT, "
               + "nom Text NOT NULL,"
               + "artiste Text NOT NULL,"
               + "idGenre Integer NOT NULL"
               + ");"

       ;
   }
```

- De la même manière, définissez une méthode qui crée la table SQL Genre_musical

```
public static String getQueryCreate_Genre_musical(){
    return "CREATE TABLE Genre_musical("
            + "id Integer PRIMARY KEY AUTOINCREMENT, "
            + "nom Text NOT NULL"
            + ");"
            ;
}
```

- Idem avec les méthodes de suppression des tables SQL

```
public static String getQueryDrop_Disque(){
    return "DROP TABLE IF EXISTS Disque;";
}
public static String getQueryDrop_Genre_musical(){
    return "DROP TABLE IF EXISTS Genre_musical";
}
```

- Modifiez en conséquence les méthodes “OnCreate” et “onUpgrade » pour que ces mécanismes se lancent automatiquement.

```
@Override
public void onCreate(SQLiteDatabase db){
    //ceci est automatiquement géré par SQLite
    db.execSQL(getQueryCreate_Disque());
    db.execSQL(getQueryCreate_Genre_musical());
}

@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion){
    db.execSQL(getQueryDrop_Disque());
    db.execSQL(getQueryDrop_Genre_musical());
    db.execSQL(getQueryCreate_Disque());
    db.execSQL(getQueryCreate_Genre_musical());
}
```

- Dans la classe « Datasource », rajouter le code qui instancie un objet de classe « Genre_musicalDataAccessObject »

```
 //factory
public DisqueDataAccessObject newDisqueDataAccessObject(){
    return new DisqueDataAccessObject(this);
}
public Genre_musicalDataAccessObject newGenre_musicalDataAccessObject(){
    return new Genre_musicalDataAccessObject(this);
}
```

- Ensuite, vous allez remplir (1 seule fois) brutalement les informations des genres musicaux (mais vous pouvez aussi faire une interface si vous le désirez) :
    - Définissez une  fonction automatique en guise d'exemple.

```
public void remplitGenre_musical(){
    Genre_musical mGenre_musical1=new Genre_musical(-1,"Rock");

    //stockage des attributs de l'objet dans la base de données
    mGenre_musicalDataAccessObject.insert(mGenre_musical1);

    Genre_musical mGenre_musical2=new Genre_musical(-1,"Classique");


    //stockage des attributs de l'objet dans la base de données
    mGenre_musicalDataAccessObject.insert(mGenre_musical2);

    Genre_musical mGenre_musical3=new Genre_musical(-1,"Rap");

    //stockage des attributs de l'objet dans la base de données
    mGenre_musicalDataAccessObject.insert(mGenre_musical3);



    //controle de l'écriture dans la base de données
    List<Genre_musical> liste_genre_musical =
mGenre_musicalDataAccessObject.readAll();
    for (Genre_musical mGenre_musical : liste_genre_musical){
        Log.w("Genre dans la BDD", mGenre_musical.toString() + "\n");
    }



}
```

    - Dans le « OnCreate » de l'Activity, appelez la fonction et  exécuter une fois le programme pour qu'elle se lance et crée une bonne fois pour toutes les différents genres musicaux.
    - Ensuite, arrêter le programme et commentez la ligne « remplitGenre_musical(); » pour ne pas qu'elle soit lancée une nouvelle fois par mégarde.

- Dans l'interface de mise en page, transformer la case qui renseignait l'identifiant du genre musical en « spinner » (liste déroulante)

```
<Spinner
    android:id="@+id/spinner"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
```

- Adaptez en conséquence le code de l'Activity pour gérer ce nouvel élément

```
En déclaration de variable :
private Spinner mSpinner1=null;

Dans le « OnCreate » :
mSpinner1=(Spinner)findViewById(R.id.spinner);
```

- Déclarer et instancier un objet de la classe "musicalDataAccessObject »

```
En déclaration de variable :
Genre_musicalDataAccessObject mGenre_musicalDataAccessObject=null;

Dans le « OnCreate » :
mGenre_musicalDataAccessObject =
mdatasource.newGenre_musicalDataAccessObject();
```

- Coder la récupération des champs « nom » de la table « Genre_musical » précédemment créée et remplie

```
//récupération des noms du genre musical
List<Genre_musical> liste_genre_musical =
mGenre_musicalDataAccessObject.readAll();
int nb_enr_genre_musical = 0;
for (Genre_musical mGenre_musical : liste_genre_musical){
    nb_enr_genre_musical = nb_enr_genre_musical +1;
}
String[] liste_nom_genre_string = new String[nb_enr_genre_musical];
nb_enr_genre_musical = 0;
for (Genre_musical mGenre_musical : liste_genre_musical){
    liste_nom_genre_string[nb_enr_genre_musical] = mGenre_musical.getNom();
    nb_enr_genre_musical =  nb_enr_genre_musical+1;
}
```

- Et mettez ces valeurs dans le Spinner

```
//et on les mets dans la liste
ArrayAdapter<String> adapter = new ArrayAdapter<String>(MainActivity.this,
        android.R.layout.simple_list_item_1, liste_nom_genre_string);
mSpinner1.setAdapter(adapter);
```

- Logiquement, l'identifiant de l'élément sélectionné dans le spinner correspond à l'identifiant du genre musical stocké dans la table « Genre_musical ». Il suffit donc de le récupérer « **this**.**mSpinner1**.getSelectedItemPosition() » et de le mettre dans les paramètres de sauvegarde de l'objet disque de la méthode « saveDisque ».

```
Disque mDisque1=new Disque(-1, this.mEditText1.getText().toString(),
this.mEditText2.getText().toString(),
this.mSpinner1.getSelectedItemPosition());
```