UTILISATION DE LA LIBRAIRIE ROOM AVEC ANDROID

I) On cree un projet Android

2) Dans le fichier build.gradle(project.sqlite) on fait la modification suivante, pour indiquer a gradle qui est l'outil permet de recuperer les dependences d'un projet.

```
// Top-level build file where you can add configuration options common to all sub-projects/modules.
buildscript {

    repositories {
        google()
        jcenter()

    }
    dependencies {
        classpath 'com.android.tools.build:gradle:3.6.1'

        // NOTE: Do not place your application dependencies here; they belong
        // in the individual module build.gradle files
    }
}
allprojects {
    repositories {
        google()
        jcenter()
        //se positionner ici,pour donner un autre depot internet au projet
        //on va avoir besoin d'utiliser une librairies (room)
        maven { url 'https://maven.google.com' }

    }
}
task clean(type: Delete) {
    delete rootProject.buildDir
}
```

3) Dans le fichier build.gradle(module:app) on ajoute les dependences qui doivent être recupéré pour travailler avec une base de donnee sqlite avec la librairie **ROOM**

```
apply plugin: 'com.android.application'
android {
    compileSdkVersion 28
    buildToolsVersion "29.0.3"
    defaultConfig {
        applicationId "com.example.sqlite"
        minSdkVersion 16
        targetSdkVersion 28
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
```

```
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'),
'proguard-rules.pro'
        }
    }
}
//ici on met les dependances internet dont on a besoin
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'androidx.appcompat:appcompat:1.1.0'
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'androidx.test.ext:junit:1.1.1'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'
    //on va rajouter nos dependances pour utiliser la librairie room
    //le principe de room va consister a utiliser une librairie qui va
    //fonctionner avec des annotations (@insert, @update,....) , ces annotations
    //vont permettre d'envelopper les operations sur la base de donnee sqlite
    //dans des codes java.Ici on a installé les librairies dont on va avoir
besoin.
    implementation "android.arch.lifecycle:extensions:1.1.1"
    implementation "android.arch.persistence.room:runtime:1.1.1"
    annotationProcessor "android.arch.persistence.room:compiler:1.1.1"
    testImplementation "android.arch.persistence.room:testing:1.1.1"
}
```

4) on va creer le package model

5) on va creez la classe Personne.java qui va être défini comme une Entity c'est a dire une classe qui va fournir des objets qui permettront d'acceder directement à la base de donnee sqlite.

```
package com.example.sqlite.model;
import androidx.annotation.NonNull;
import androidx.room.Entity;
import androidx.room.PrimaryKey;
//Cette annotation va definir les objets de la classe Personne comme etant des
objets
//qui seront mappes à la table Personne correspondante de la base de donnee que
l'on va
//créer sous sqlite. (hibernate (java), doctrine (php)).
//permet de definir un objet Personne comme ayant un acces direct à la base de
donnee sqlite grace
//a la librairie Room
@Entity
public class Personne {
    //on définit l'id comme etant non null
    //c'est une clé primaire
    //elle est en autoincrement
    @NonNull
    @PrimaryKey(autoGenerate = true)
    private int id;
    @NonNull
```

```java
    private String nom;
    private String prenom;
    private int age;
    private float salaire;
    public Personne(String nom, String prenom, int age, float salaire) {
        this.nom = nom;
        this.prenom = prenom;
        this.age = age;
        this.salaire = salaire;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getNom() {
        return nom;
    }
    public void setNom(String nom) {
        this.nom = nom;
    }
    public String getPrenom() {
        return prenom;
    }
    public void setPrenom(String prenom) {
        this.prenom = prenom;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public float getSalaire() {
        return salaire;
    }
    public void setSalaire(float salaire) {
        this.salaire = salaire;
    }

@Override
public String toString() {
    return "Personne{" +
            "id=" + id +
            ", nom='" + nom + '\'' +
            ", prenom='" + prenom + '\'' +
            ", age=" + age +
            ", salaire=" + salaire +
            '}';
}


}
```

6) On cree le package dao

7) on cree l'interface DaoGestion.java qui va permettre de définir toutes les operations que l'on souhaite faire sur la base de donnee sqlite par l'intermediaire de la librairie room .On dit que room va encapsuler les operations sur la base de donnee grace aux annotations.

```
package com.example.sqlite.dao;
import androidx.room.Delete;
import androidx.room.Insert;
import androidx.room.Query;
import androidx.room.Update;
import com.example.sqlite.model.Personne;
import java.util.List;
@Dao
public interface DaoGestion {
    //inserer une personne, l'annotation va permettre a la librairie room de
savoir
    //qu'il doit mapper l'objet personne avec la table sqlite pour inserer une
personne
    @Insert
    public void insertPersonne(Personne personne);
    //modifier une personne
    @Update
    public void updatePersonne(Personne personne);
    //supprimer une personne
    @Delete
    public void deletePersonne(Personne personne);
    //retourner la liste des personnes
    @Query("SELECT * FROM personne")
    public List<Personne> loadAllPersonnes();
    //retourne une personne en fonction de son nom
    @Query("SELECT * FROM personne WHERE nom=:namePersonne")
    public Personne loadPersonneByName(String namePersonne);
}
```

8) On définit une classe PersonneDatabase.java qui va permettre de definir la base de donnee gére par ROOM

```
package com.example.sqlite.dao;
import androidx.room.Database;
import androidx.room.RoomDatabase;
import com.example.sqlite.model.Personne;
//classe abstraite non instanciable, la classe etends RoomDatabase qui permet
d'avoir acces
//a une base de donnee sqlite gere par la librairie Room
//on va definir les entity (classe avec l'annotation @Entity) que l'on va utiliser
avec
//notre base de donnee.
@Database(entities = {Personne.class},version=1)
public abstract class PersonneDatabase extends RoomDatabase {
    //On cree une methode abstraite qui va retourner un DAO (Direct Access object)
a partir d'une base de donnee ROOM
    //Personne est un objet DAO car il va acceder directement à la base de donnee
grace a la librairie ROOM.
    public abstract DaoGestion daoGestion();
```

```
}
```

explications:

```
@Database(entities = {Personne.class},version=1)
```

cette annotation permet de définir toutes les entity qui seront des classes qui pourront acceder directement a la base de donnee sqlite via room. Si par exemple dans le dossier Model on définit:

Personne.java, Vehicule.java, Enfants.java….

```
@Database(entities = {Personne.class,Vehicule.java,Enfants.java},version=1)
```

version=1: represente le numero de version de la base de donnee si on choisit de modifier la base de donnee on pourra modifier le numero de version.(il faudrait dans ce cas la soit decider de supprimer completement la base de donnee et on la reconstruit sinon on cree un script pour la modifier)

9)Nous creons ensuite PersonneDatabaseAccessor.java qui est une classe qui va permettre d'obtenir une instance singleton vers la base de donnee sqlite afin de la manipuler.

```
package com.example.sqlite.dao;
import android.content.Context;
import androidx.room.Room;
//cette classe doit nous fournir un singleton pour pouvoir acceder a la base de donnee
//c'est a dire une instance unique  pour acceder a la base de donnee.
public class PersonneDatabaseAccessor {
    //On définit un objet de type PersonneDatabase pour l'acces à la base de donnee via room
    private static PersonneDatabase personneDatabaseInstance;
    //on va définir une constante qui va contenir le nom de la base de donnee (sqlite) qui sera stocke sur
    //sur le telephone.
    private static final String PERSONNE_DB_NAME="personne_db";
    //on va definir un constructeur private , car comme on veut que cette classe nous renvoie un singleton
    //on ne doit pas avoir la possibilite de l'instancier plusieurs fois .
    //si j'enleve la ligne ci-dessous alors on est capable d'ecrire
    //PersonneDatabaseAccessor d=new PersonneDatabaseAccessor();
     private  PersonneDatabaseAccessor(){}
    //On va creer une methode statique (c'est une methode qui est appele directement sur la classe) qui
    //va nous permettre d'obtenir le singleton
    public static PersonneDatabase getInstance(Context context){
         //si l'instance pour se connecter à la base de donnee est null on la cree
         if (personneDatabaseInstance==null){
             //permet de creer une base de donnee sqlite et de la retourner comme instance de base
             //de donnee
             personneDatabaseInstance=
Room.databaseBuilder(context,PersonneDatabase.class,PERSONNE_DB_NAME).build();
         }
```

```
        return personneDatabaseInstance;
    }
}
```

9) On crée une ListView dans activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
    <ListView
        android:id="@+id/liste"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

10) On modifie le fichier MainACtivity.java on cree un thread pour les operations sur la base de donnee et on modifie le Thread UI pour l'affichage a l'ecran

```
package com.example.sqlite;
import androidx.annotation.Nullable;
import androidx.appcompat.app.AppCompatActivity;
import androidx.lifecycle.LiveData;
import androidx.lifecycle.Observer;
import android.app.Person;
import android.os.Bundle;
import android.util.Log;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.TextView;
import com.example.sqlite.dao.PersonneDatabase;
import com.example.sqlite.dao.PersonneDatabaseAccessor;
import com.example.sqlite.model.Personne;
import java.util.ArrayList;
import java.util.List;
public class MainActivity extends AppCompatActivity {
    //Un objet ListView auquel on va associer la ListView défini dans le fichier
xml
    ListView liste;
    //adapter qui va permettre de peupler la listview avec un ArrayList de
Personne
    ArrayAdapter<Personne> aa;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
        //permet de convertir les elements XML en objet JAVA
        setContentView(R.layout.activity_main);
        //R:Classe genere par android qui contient pour chaque composant
        //du programme (View, Layout,Image,....) un id unique.
        liste=findViewById(R.id.liste);
        //Une régle trés importante c'est qu'il ne faut absolument pas realiser
        //des traitements qui peuvent être long dans lE THREAD UI .(processus qui
        //s'executent en memoire centrale et qui gere l'affichage et les
operations
        //de l'uitlisateur sur l'interface graphique.
        //plusieurs technique pour qu'un programme s'execute dans un autre THREAD
        //utilisation de la classe Thread
        //utilisation des services
        //utilisation de la classe AsyncTask:permet de faire des traitements
asynchrones
        //On va créer un nouveau thread,on souhaite que ce thread communique ses
donnees
        //au ThreadUI, dans ce thread on ne peut pas acceder au View du threadUI .
        Thread td=new Thread(new Runnable() {
            @Override
            public void run() {
                //on cree une instance de personne
                Personne p=new Personne("younsi","mousse",50,2500.50f);
                //on doit recuperer une reference vers la base de donnee
                //getBaseContext():represente le contexte de la MainActivity
                PersonneDatabase
dbobjet=PersonneDatabaseAccessor.getInstance(getBaseContext());
                //nous allons inserer l'objet p dans la base sqlite
                dbobjet.daoGestion().insertPersonne(p);
                //on souhaite s'assurer que la personne a ete insere,donc on va
lire le contenu
                //de la base de donnee et l'afiicher dans le logcat
                ArrayList<Personne> liste= (ArrayList<Personne>)
dbobjet.daoGestion().loadAllPersonnes();
                for (Personne pers:liste){
                    Log.d("personne",pers.toString());
                }
                //on va initialiser l'adapter
                //parametres:contexte,layout,liste de personnes
                aa=new
ArrayAdapter<Personne>(getBaseContext(),android.R.layout.simple_list_item_1,liste)
;
                //on va envoyer l'adapter au ThreadUI
                afficherUI(aa);
            }
        });
        //on demarre le Thread
        td.start();
    }
    //cette methode se deroule dans le Thread UI
    public void afficherUI(final ArrayAdapter<Personne> adp){
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                liste.setAdapter(adp);
```

```
            }
        });
    }
}
```

ARCHITECTURE DU PROJET