Introduction à Android Studio

Table des matières

[1 Installation Android Studio 3](#_Toc29384878)

[1.1 Récupération de l’installateur 3](#_Toc29384879)

[1.2 Installation 3](#_Toc29384880)

[2 Création de projet Android Studio 3](#_Toc29384881)

[2.1 Type d’activité 3](#_Toc29384882)

[2.2 Généralités, langage et informations de version 3](#_Toc29384883)

[2.2.1 Langage 3](#_Toc29384884)

[2.2.2 Version de l’API Android 3](#_Toc29384885)

[2.2.3 Support des applications instantanées 3](#_Toc29384886)

[2.3 Ajout d’un appareil virtuel 3](#_Toc29384887)

[2.3.1 Choix de téléphone 3](#_Toc29384888)

[2.3.2 Installation de l’OS 3](#_Toc29384889)

[3 Ajout d’une activité 4](#_Toc29384890)

[3.1 Création nouvelle activité 4](#_Toc29384891)

[3.2 Ouvrir une activité depuis une autre 4](#_Toc29384892)

[3.2.1 Ajouter un bouton 4](#_Toc29384893)

[3.2.2 Ajout d’actions lors du click du bouton 4](#_Toc29384894)

[3.2.3 Instructions de lancement d’une autre activité 4](#_Toc29384895)

[4 Ajouter un compteur sur un bouton 5](#_Toc29384896)

[4.1 Création du bouton 5](#_Toc29384897)

[4.2 Variable du compteur 5](#_Toc29384898)

[4.2.1 Création 5](#_Toc29384899)

[4.2.2 Incrémentation 5](#_Toc29384900)

[4.3 Affichage du compteur 5](#_Toc29384901)

[4.3.1 Ajout d’une « TextView » 5](#_Toc29384902)

[4.3.2 Changement du texte 5](#_Toc29384903)

[5 Ajouter un élément à la vue depuis le code 6](#_Toc29384904)

[5.1 Création de l’élément 6](#_Toc29384905)

[5.2 Création du « layout » 6](#_Toc29384906)

[5.3 Affichage du bouton 6](#_Toc29384907)

[6 Persistance des données 6](#_Toc29384908)

[6.1 Base de données SQLite (RoomDatabase) 7](#_Toc29384909)

[6.1.1 Création de la table 7](#_Toc29384910)

[6.1.2 Création du DAO (Data Access Object) 7](#_Toc29384911)

[6.1.3 Création base de données 8](#_Toc29384912)

[6.1.4 Ajouter une note 8](#_Toc29384913)

[6.1.4.1 TaskAsync pour ajouter une note 8](#_Toc29384914)

[6.1.4.2 Dans l’activité 9](#_Toc29384915)

[6.1.5 Afficher les informations de la table 9](#_Toc29384916)

[6.1.5.1 Déléguée 9](#_Toc29384917)

[6.1.5.2 TaskAsync pour récupérer les notes 10](#_Toc29384918)

[6.1.5.3 Dans l’activité 10](#_Toc29384919)

[6.2 Base de données SQLite (avec SQLiteOpenHelper) 11](#_Toc29384920)

[6.2.1 Créer une classe définissant la base de données 11](#_Toc29384921)

[6.2.2 Créer des objets correspondant au tables de la base de données 11](#_Toc29384922)

[6.2.3 Créer une classe de connexion avec la base de données 12](#_Toc29384923)

[6.2.3.1 Constructeur 12](#_Toc29384924)

[6.2.3.2 Ouverture et fermeture de connexion 12](#_Toc29384925)

[6.2.3.3 Insérer des données 13](#_Toc29384926)

[6.2.3.4 Récupérer des données 13](#_Toc29384927)

[6.2.4 Utiliser la connexion avec la base de données 14](#_Toc29384928)

[7 Utilisation d’un senseur (Accéléromètre ici) 14](#_Toc29384929)

[7.1 SensorEventListener 14](#_Toc29384930)

[7.2 Détecter un mouvement 15](#_Toc29384931)

# Installation Android Studio

## Récupération de l’installateur

Pour récupérer l’installateur, on peut soit aller sur [developer.android.com/studio](https://developer.android.com/studio) sous forme d’exécutable ou bien d’archive ZIP.

## Installation

Le package installé par l’installeur comprend tout ce qui est nécessaire pour le fonctionnement de l’IDE.

L’installateur est assez intuitif, rien de spécial. Peu importe quel installateur est choisi, il y a besoin d’être administrateur de l’ordinateur pour l’installation.

# Création de projet Android Studio

## Type d’activité

Lors de la création d’un projet Android Studio, l’application nous demande tout d’abord de choisir un type d’activité. Cela nous permet de choisir quels éléments de base il y aura sur la première interface de l’application.

## Généralités, langage et informations de version

Dans l’étape suivante, on peut choisir le nom, le nom du package, son emplacement de stockage, le langage ([Kotlin](https://fr.wikipedia.org/wiki/Kotlin_(langage)) ou [Java](https://fr.wikipedia.org/wiki/Java_(langage))), et la version minimale de l’API Android que l’application pourra supporter.

### Langage

Pour ce qui est du choix du langage, ce module va rester en Java, car ce langage est plus utilisé.

### Version de l’API Android

La version minimale de l’API que nous allons utiliser sera certainement l’API 23 pour minimum Android 6.0. Plus l’API choisie est récente, plus de fonctionnalités sont disponibles, mais beaucoup moins d’utilisateurs sont touchés.

### Support des applications instantanées

Lors de la création du projet, on nous demande aussi si le projet doit pouvoir supporter les applications instantanées ([Qu’est-ce ?](https://code.tutsplus.com/tutorials/what-are-android-instant-apps--cms-29283)).

## Ajout d’un appareil virtuel

Pour ajouter un émulateur de téléphone virtuel, il faut aller dans l’onglet « Tools » puis AVD Manager (AVD = Android Virtual Device). Cela va ouvrir le gestionnaire d’appareils virtuels, ou l’on peut ajouter un nouvel appareil.

### Choix de téléphone

La première étape de la création d’un appareil virtuel est de choisir un téléphone. Nous avons choisi le Galaxy Nexus pour sa comptabilité avec Android 6.

### Installation de l’OS

Il faut ensuite choisir, éventuellement télécharger l’OS. Une fois l’image de l’OS téléchargée et sélectionnée, la configuration de l’appareil virtuel est terminé.

# Ajout d’une activité

## Création nouvelle activité

Pour ajouter une activité à une application/à un projet, il faut faire un clic droit sur « App », puis sélectionner « New… » « Activity » et « Gallery ».

Cette suite d’action ouvre la galerie d’activités, où l’on peut facilement créer la nouvelle activité.

## Ouvrir une activité depuis une autre

### Ajouter un bouton

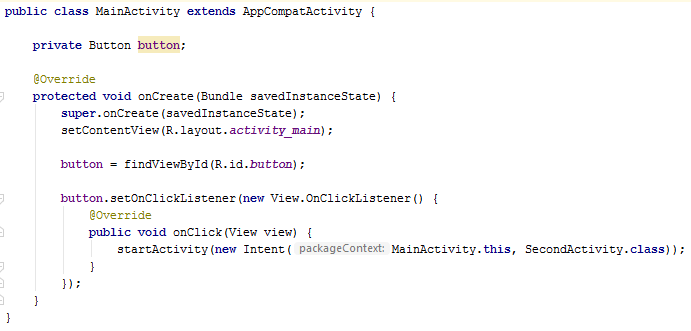
Ajouter un bouton depuis l’éditeur visuel. Pour récupérer le bouton dans le code Java, il faut créer un objet « Button », et exécuter la méthode « findViewById » avec l’identifiant du bouton défini dans la vue. (Champ « id »)

### Ajout d’actions lors du click du bouton

Ensuite il faut ajouter un observateur de l’évènement click du bouton. Pour ce faire, il faut utiliser la méthode « setOnClickListener » et lui passer un objet « View.OnClickListener ». Cet objet abstrait doit être complété par une méthode OnClick, dans laquelle on peut mettre des instructions.

### Instructions de lancement d’une autre activité

Pour ouvrir une activité, il faut créer un objet « Intent » contenant le contexte actuel (l’activité actuelle) et l’activité destination. Ensuite, il faut exécuter « startActivity » avec l’objet « Intent ».



# Ajouter un compteur sur un bouton

## Création du bouton

Comme précédemment, il faut tout d’abord créer un bouton, et le récupérer en Java. Pour cela, il faut lui définir un identifiant dans la vue, et exécuter la méthode « findViewById ». Comme précédemment également, il faut ajouter l’événement grâce à la méthode « setOnClickListener » [(détails)](#_Ajout_d’actions_lors)

## Variable du compteur

### Création

Dans la classe de l’activité, ajouter un propriété compteur.

### Incrémentation

Dans la méthode OnClick, ajouter l’instruction d’incrémentation du compteur.

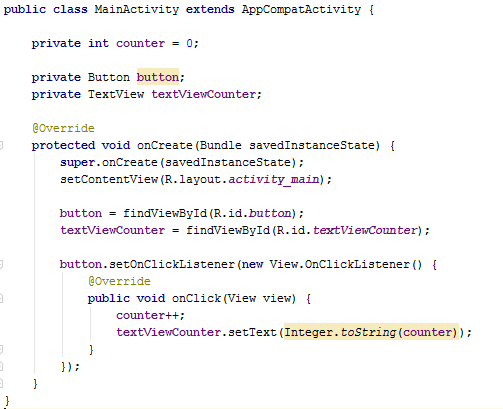
## Affichage du compteur

### Ajout d’une « TextView »

Pour afficher le contenu de la variable du compteur, il faut ajouter un « TextView ». Pour ce faire, il faut, comme pour le bouton, l’ajouter dans la vue, lui donner un id, et récupérer l’objet grâce à la méthode « findViewById ».

### Changement du texte

Pour afficher le nombre de clicks, il suffit d’exécuter la méthode « setText » de la TextView, en lui passant en paramètre une chaine de caractères.



# Ajouter un élément à la vue depuis le code

## Création de l’élément

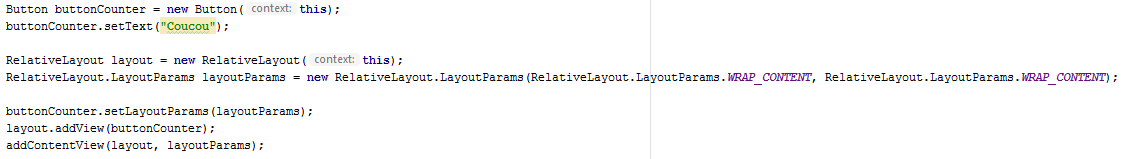
Premièrement, il faut créer l’élément (View) à ajouter. Par exemple, un bouton. Il faut créer un nouveau bouton en lui spécifiant que le contexte est l’activité actuelle, et y ajouter par exemple un texte. La méthode pour ajouter un texte au bouton est « setText ».

## Création du « layout »

Ensuite, il faut créer l’affichage de l’élément. Pour ce faire, il faut créer un « RelativeLayout » en passant en paramètre l’activité actuelle comme contexte. Ensuite, il faut créer les paramètres d’affichage de cet affichage. Cela se fait en utilisant la classe « RelativeLayout.LayoutParams ». Dans cet objet, on peut définir la taille, les marges, et autres paramètres d’affichage de notre bouton. Lors de sa construction, on a besoin de passer en paramètre un type de paramètre de layout, par exemple « WRAP\_CONTENT ».

## Affichage du bouton

Pour afficher le bouton, il faut lui donner les paramètres d’affichage avec « setLayoutParams ». Ensuite, il faut l’ajouter au « layout », avec la méthode « addView » et lancer la méthode « addContentView » en lui donnant en paramètres le « layout » et les paramètres d’affichage.



# Persistance des données

Il existe plusieurs manières d’avoir une persistance des données sur Android :

* Shared preferences. C’est un format de stockage clé/valeur dans un fichier XML
* Stocker dans une base de données structurée en local de type SQLite qui permet une manipulation des données avec des requêtes SQL.
* Stockage en local interne. Le stockage interne est une partie de la mémoire de l’appareil qui n’est, par défaut, ni accessible par l’utilisateur, ni par les autres applications.
* Stockage en local externe. Le stockage externe est la mémoire de l’appareil et/ou une éventuelle carte SD dont les données sont accessibles aux autres applications et à l’utilisateur depuis son téléphone ou en le branchant à un PC. Les données du stockage externe sont en soi publiques mais on peut créer des dossiers privés à l’intérieur. Quand l’utilisateur désinstalle l’application, tout le contenu est supprimé.
* Une base de données en ligne sur un serveur

On va donner deux implémentations différentes de base de données en SQLite en exemple.

## Base de données SQLite (RoomDatabase)

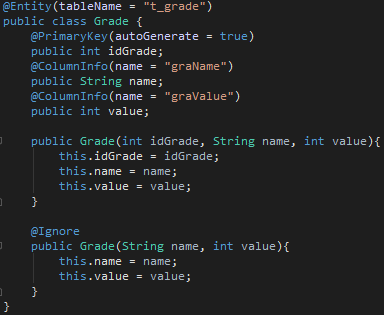
### Création de la table

Tout d’abord, il faut créer la table qui sera directement liée à l’objet. On va lui mettre un tag en dessus indiquant que c’est une entité (@Entity) et on va préciser le nom de la table grâce au paramètre « tableName ».

Ensuite, à l’intérieur de la classe, on va déclarer les différents champs de la table (et donc de l’objet). On va définir un id avec, en dessus, le tag « PrimaryKey » et dans notre cas on va préciser aussi qu’il est autogénéré. Ce qui nous donne « @PrimaryKey(autogenerate = true) ».

Pour les autres attributs, on met le tag « @ColumnInfo(name= "nom du champ ") » en dessus de l’attribut du champ également.

Ensuite on crée le constructeur rassemblant tous ces champs. Si on a besoin d’un constructeur sans id par exemple, on ajoute en dessus de celui-ci le tag « @Ignore ».

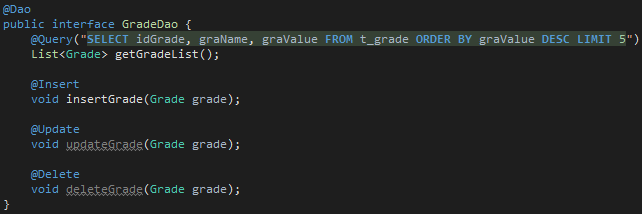


### Création du DAO (Data Access Object)

Cet objet que l’on va créer va servir à exécuter les requêtes dans la table. C’est une interface avec le tag « @Dao ». Il est recommandé de créer un DAO par table.

Les requêtes d’insertion, de mise à jour et de suppression sont directement gérées : il suffit de mettre leur tag respectif.

Pour créer soi-même une requête personnalisée, on utilise le tag « @Query("Votre requête SQL ") » et en dessous on met l’objet qui sera retourné (ou la liste d’objets)



### Création base de données

Puis on va créer la base de données. Cette classe abstraite va étendre la classe « RoomDatabase ». Au-dessus de la déclaration de la classe, on met le tag « @Database(entities = « le nom de la table ».class, exportschema = false, version = 1) ».

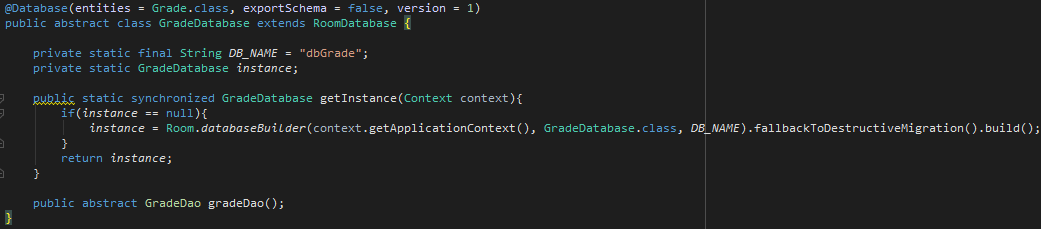
On va déclarer « final string DB\_NAME = "le nom de la base de données" ;» ainsi qu’une instance privée de la classe que l’on vient de créer (afin de créer un singleton).

Au lieu d’un constructeur, on crée une méthode getInstance comme ceci :

Public static synchronized « nom de la classe » getInstance(Context context)

On regarde dedans si l’instance est nulle et si c’est le cas on va l’initialiser comme indiqué sur l’image, c’est à dire à partir de la classe Room en donnant le contexte, la classe actuelle qu’on construit. Et on retourne l’instance.

Et au final on déclare en abstrait le DAO associé (les DAO s’il y en a plusieurs).



### Ajouter une note

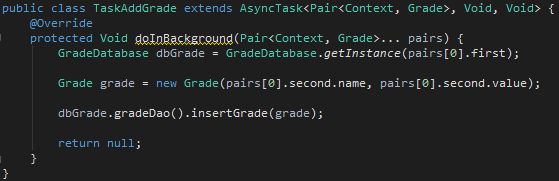
#### TaskAsync pour ajouter une note

On va devoir créer une classe pour effectuer la tâche qui va étendre la classe « TaskAsync » et comme on insère une note on va devoir donner une paire contenant le contexte qui est nécessaire et la note. Les deux autres seront Void car nous n’attendons pas de type de retour pour le dernier et celui du milieu ne nous sert pas car on ne fait rien en parallèle.

Ceci est nécessaire car on ne peut pas ajouter ou récupérer des données de manière synchrone avec les RoomDatabes.

On va devoir créer la méthode overridée « doInBackground » et lui donner la paire en paramètre.

On récupère l’instance de la base de données, on crée la note à partir des informations du paramètre et on l’insère. On doit ensuite retourner null.

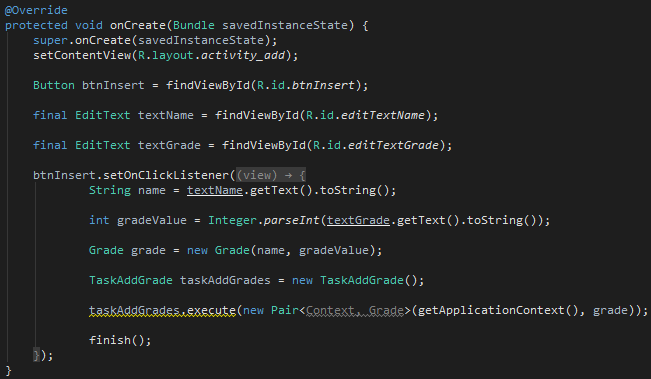


#### Dans l’activité

Pour l’implémentation dans la vue il va falloir créer un formulaire avec les champs nécessaires que l’utilisateur va remplir et récupérer les valeurs entrées dans ceux-ci.

On crée une note grâce aux valeurs. On instancie une « TaskAddGrade » et on l’exécute en lui donnant notre paire contexte, note. Et la note est ajoutée dans la table.

Et pour revenir à l’activité précédente on peut utiliser la méthode « finish()  » qui ferme l’activité.



### Afficher les informations de la table

#### Déléguée

On va créer une interface qui va nous renvoyer le contenu. Elle contient une méthode qui nous retournera les notes.



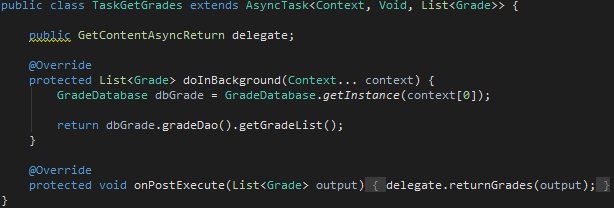
#### TaskAsync pour récupérer les notes

Cette fois-ci, on va donner le contexte pour le premier paramètre et une liste de notes pour le dernier puisque c’est ce qu’on va récupérer.

On va déclarer en public notre déléguée créée juste avant.

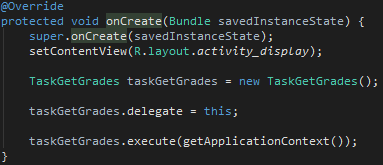
Dans la méthode « doInBackground » on récupère l’instance de la base de données et on retourne la liste de notes obtenue grâce à notre méthode dans le dao.

Et on va déclarer la méthode « onPostExecute », qui demande la liste de grade en paramètre, dans laquelle on fait simplement en sorte que notre déléguée retourne les notes avec comme paramètre l’output obtenu dans le paramètre de « onPostExecute ».



#### Dans l’activité

On va instancier une « TaskGetGrades » et donner comme valeur « this » pour sa déléguée et on l’exécute en donnant le contexte.



Une fois ceci fait, il ne reste plus qu’à overrider la méthode « returnGrades » et gérer l’affichage comme on le souhaite.

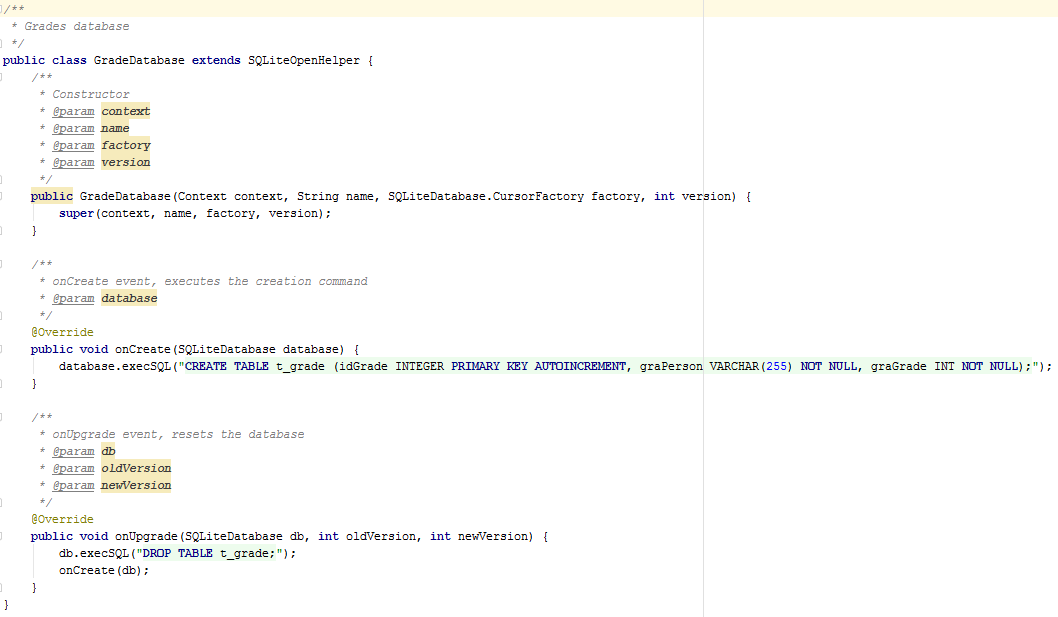


## Base de données SQLite (avec SQLiteOpenHelper)

### Créer une classe définissant la base de données

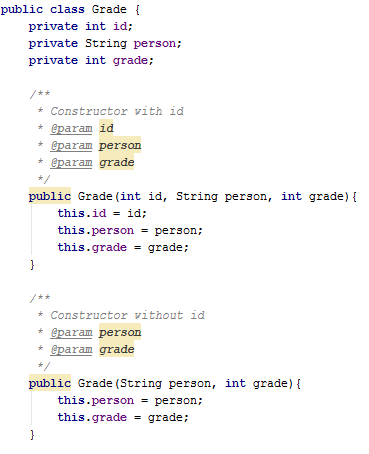
Créer une nouvelle classe étendant la classe abstraite « SQLiteOpenHelper ». Cette extension implique la création d’un constructeur, et de deux méthodes qu’il faut « Override » (faire Alt Enter pour les générer avec Android Studio). La première est la méthode « onCreate », qui est la méthode qui est exécutée lors de la création de la base de données, et la deuxième est « onUpgrade », qui est lancée lorsque la base de données change de version.

Dans la méthode « onCreate », il y a, en paramètre, une « SQLiteDatabase ». Avec ce paramètre, on peut faire « database.execSQL("commande SQL") » en mettant la commande SQL de création de notre base de données.



### Créer des objets correspondant au tables de la base de données

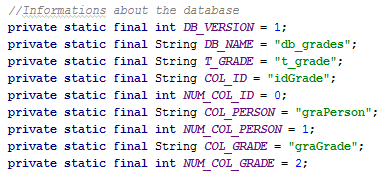
Afin de pouvoir insérer et traiter plus facilement des données, il est conseillé de créer un objet par table. Cet objet contiendra une propriété par champ, et ces propriétés devront avoir des getters-setters (cachés sur l’image pour raison de place).



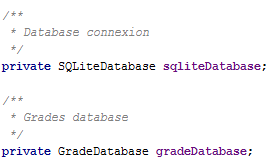
### Créer une classe de connexion avec la base de données

Ensuite, il faut lier la base de données créée avec le programme. Pour cela, il nous faut une classe qui contiendra des méthodes publiques retournant des données de la base de données.

Dans cette classe, il est conseillé de mettre des constantes avec le nom de la base de données, sa version, les noms des tables et champs.



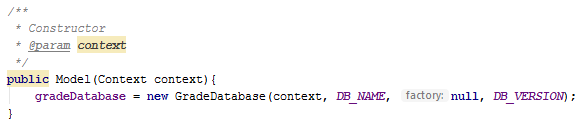
Cette classe doit contenir deux objets. Un objet « SQLiteDatabase » et une instance de la classe que vous venez de créer.



#### Constructeur

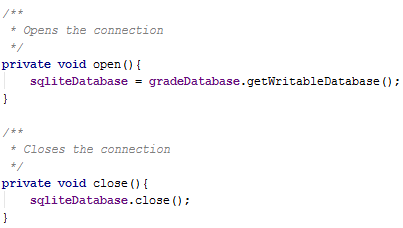
Dans le constructeur, instanciez le deuxième objet, en lui donnant le contexte, le nom de la base de données, et la version. Le contexte doit être un paramètre du constructeur.

Exemple :



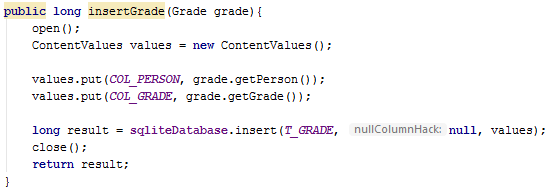
#### Ouverture et fermeture de connexion

Pour ouvrir et fermer la connexion, la solution la meilleure est de créer une méthode « open » et une méthode « close ». La méthode « open » initialise l’objet « SQLiteDatabase » en exécutant la méthode « getWritableDatabase » de l’objet instancié dans le constructeur. La méthode « close » exécute la méthode « close » de l’objet « SQLiteDatabase ».



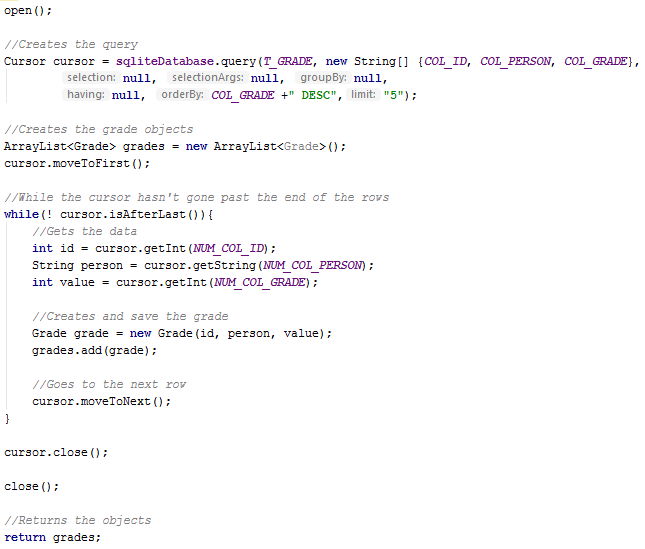
#### Insérer des données

Pour insérer des données, il faut exécuter la méthode « insert » de l’objet « SQLiteDatabase », en lui donnant le nom de la table, et des valeurs, par exemple mises en forme avec un « ContentValues ».



#### Récupérer des données

Pour récupérer des données, il faut exécuter la méthode « query » de l’objet « SQLiteDatabase », en lui donnant la table, les colonnes, et autres paramètres (Where, Order By, etc..). Cette méthode va retourner un « Cursor », que l’on peut parcourir avec une boucle while afin de récupérer les données, et les mettre en forme, par exemple dans un objet.



### Utiliser la connexion avec la base de données

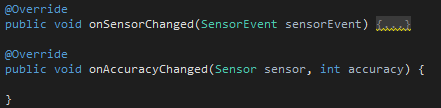
Pour utiliser les méthodes créées dans la classe de connexion avec la base de données, il faut instancier cette classe, par exemple dans le constructeur de d’une activité, et exécuter la méthode au moment où on veut insérer ou récupérer des données, en utilisant, ou pas, les [objets](#_Créer_des_objets).

# Utilisation d’un senseur (Accéléromètre ici)

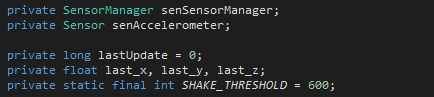
## SensorEventListener

Il faut implémenter le « SensorEventListener » et les méthodes abstraites requises, « onSensorChanged » et « onAccuracyChanged ».

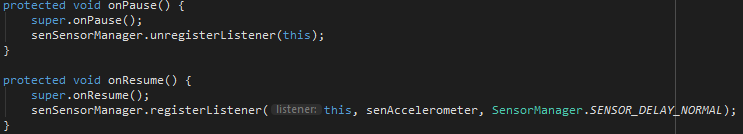




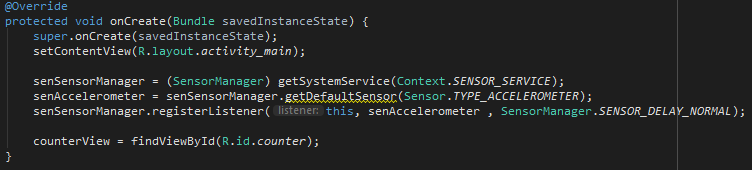
On va déclarer les variables suivantes : un « SensorManager » et un « Sensor ». La dernière fois que l’on a mis à jour dans la variable « lastUpdate », des float pour les dernières positions x, y et z. Et pour finir, une constante pour définir la vitesse à laquelle il faudra avoir bougé l’appareil pour exécuter certaines actions dans la constante « SHAKE\_THRESHOLD ».



Ensuite, deux méthodes peuvent être overridées à savoir « onPause » et « onResume ». Comme leurs noms l’indiquent, elles servent à mettre le listener en pause ou le remettre en route.



Dans la méthode « OnCreate » on va instancier le « SensorManager » et le « Sensor » en tant qu’accéléromètre pour notre exemple. Et on va enregistrer dans le « SensorManager » ce listener en lui donnant « this » comme listener, l’accéléromètre et le délai normal.



## Détecter un mouvement

On récupère le senseur activé. Si c’est l’accéléromètre on va stocker les valeurs x,y et z. On récupère le time actuel pour enregistrer le time de la dernière mise à jour. Grâce à ces valeurs on va calculer la vitesse et pouvoir la comparer à la constante « SHAKE\_THRESHOLD » déclarée précédemment. Et si la vitesse est supérieure on va pouvoir effectuer une action. Dans ce cas, on affiche dans une textview les valeurs de x, y et z.

