R4 Real 11
Programmation mobile sous Androïd

Base de données

Mapping objet-relationnel

Librairie Room

https://developer.android.com/training/data-storage/room/

Base de données sous Android

- ✓ Les bases de données sur Android sont gérées par **SQLite**

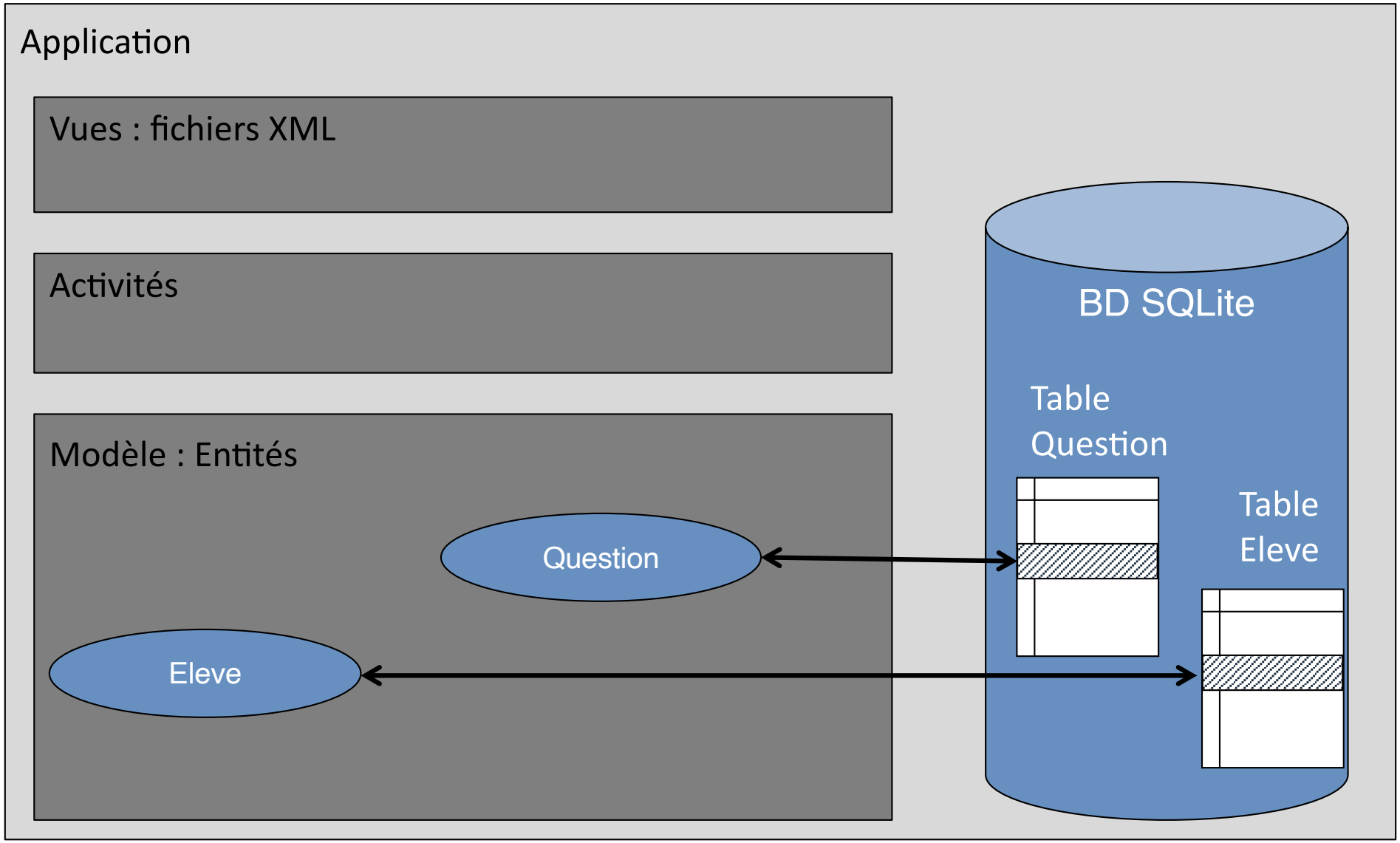
- ✓ **Chaque application peut avoir sa propre base**

- ✓ Pour utiliser SQLite :
  - Soit utilisation classique : SQL, Cursor, etc.
  - Soit mise en place d'un **mapping objet/relationnel**

Mapping objet/relationnel (1/2)

- ✓ Lorsque l'on développe des applications, on est en général amené à manipuler une BD et on est toujours confrontés aux mêmes problèmes récurrents :
  - Trouver des enregistrements (n-uplets)
  - Mettre à jour des enregistrements
  - Ajouter de nouveaux enregistrements
  - Supprimer des enregistrements
  - Traiter une liste d'enregistrements
  - etc

Mapping objet/relationnel (2/2)



Entité

- ✓ C'est une classe qui permet de **représenter** directement **un enregistrement** stocké dans une table du SGBD

- ✓ La classe aura des **attributs** qui correspondront aux **colonnes de la table** qu'elle représente

- ✓ On développera donc une telle classe pour **chaque table de la BD**

Exemple d'entité : la classe Question

```
public class Question {

  // Attributs
   private int id;
   private String question;
   private String bonneReponse;
   private String mauvaiseReponse1;
   private String mauvaiseReponse2;

   // Contructeurs
   …

   // Getters et Setters
   …
```

Comment réaliser le mapping ? (1/2)

- ✓ Créer des méthodes insert, update, remove, select, etc. pour chaque classe.

- ✓ Exemple :

```
public long insert(Question question) {
        // Création d'un ContentValues (fonctionne comme une HashMap)
        ContentValues values = new ContentValues();
        // Ajout clé/valeur : colonne/valeur
        values.put(COL_QUESTION, question.getQuestion());
        values.put(COL_BONNE_REPONSE, question.getBonneReponse());
        values.put(COL_MAUVAISE_REPONSE_1, question.getMauvaiseReponse1());
        values.put(COL_MAUVAISE_REPONSE_2, question.getMauvaiseReponse2());
        // Insertion de l'objet dans la BD via le ContentValues
        return getDB().insert(TABLE_QUESTION_REPONSE, null, values);
```

Comment réaliser le mapping ? (2/2)

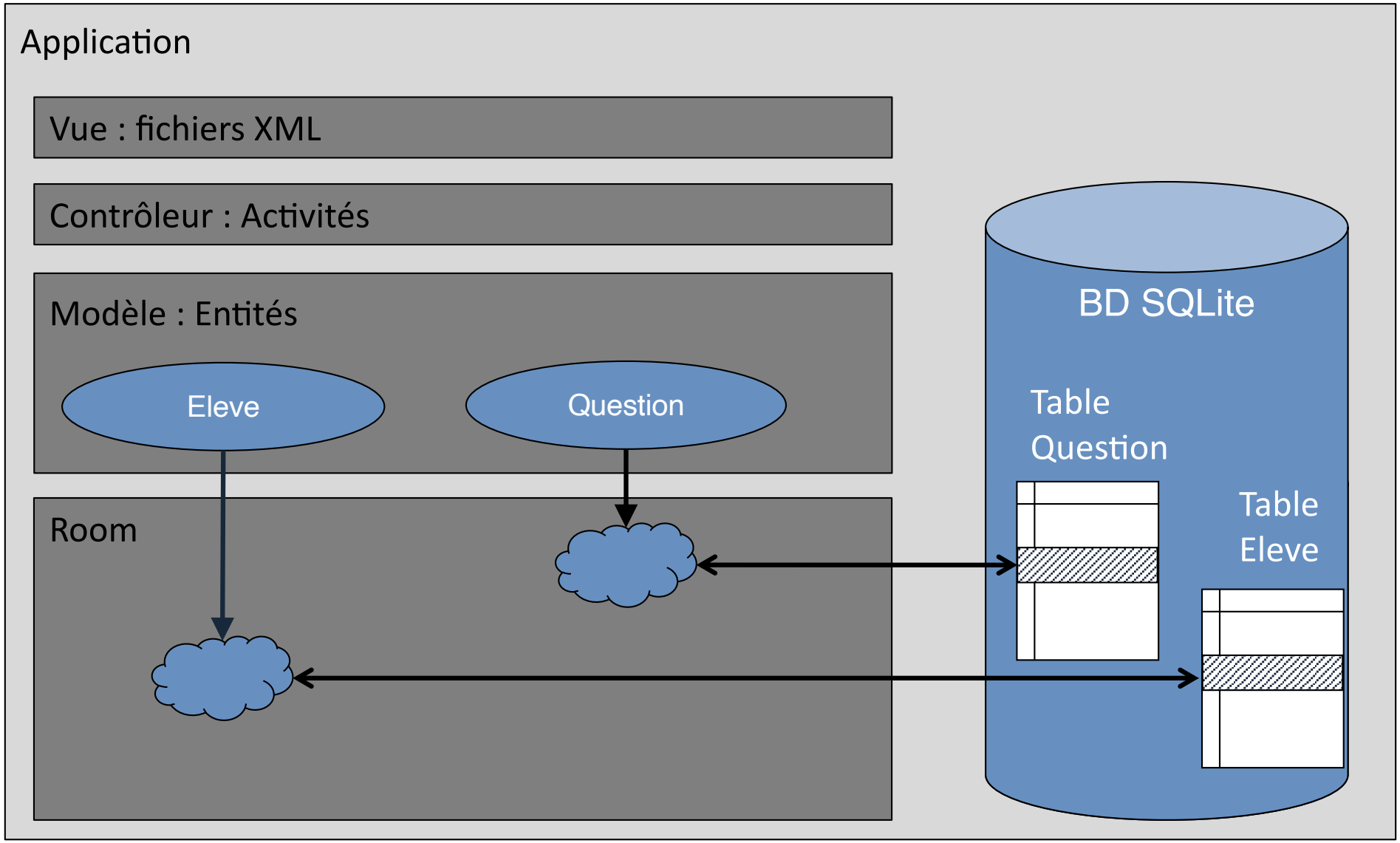
- ✓ Implémenter "à la main" un mapping objet/relationnel (ORM) est une **tâche répétitive et fastidieuse**

- ✓ Il existe des **frameworks/librairies** pour réaliser cette corvée de façon "propre" et automatique :
  - Pour android : **Librairie Room**, SugarORM, greenDAO, OrmLite, Realm, etc.
  - En java « classique » : Hibernate, EclipseLink, etc.
  - En PHP : doctrine, propel, etc.

Un framework ORM

- ✓ **Propose une « programmation BD » plus attractive**
  - Il fournit une API bien définie pour accéder à la BD
  - Il utilise les standards de la programmation objet (exceptions, itérateurs, ...)
- ✓ **S'intègre naturellement au modèle MVC, MVP, MVVP, etc.**
  - Il fournit le socle de la partie « Modèle »
  - En utilisant l'héritage, on peut facilement développer les objets métiers
- ✓ **Participe à la sécurité de l'application**
  - Il utilise les "prepared statements" de SQL pour éviter l'injection de code, il "échappe" les chaînes, etc.

Room (1/2)


Application
Vue : fichiers XML
Contrôleur : Activités
Modèle : Entités
Eleve
Question
Room
BD SQLite
Table
Question
Table
Eleve

Room(2/2)

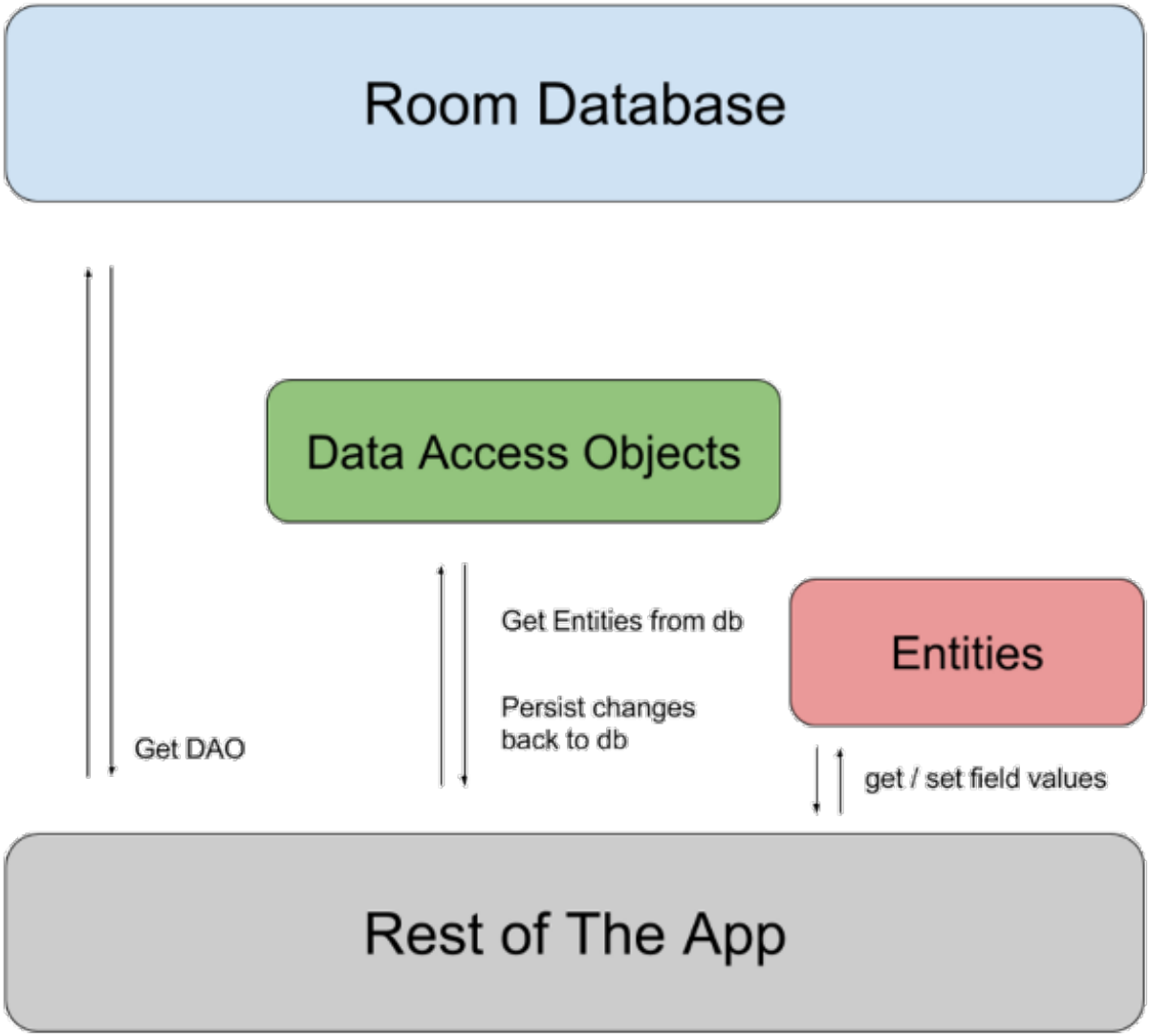
- ✓ Room est une librairie permettant le mapping objet/relationnel

- ✓ Elle permet de
  - Créer des entités
  - Créer des requêtes pour trouver des enregistrements

https://developer.android.com/topic/libraries/architecture/room

https://developer.android.com/training/data-storage/room/

3 Composants pour Room



Database : point d'accès aux données relationnelles persistantes de l'application

DAO : contient les méthodes d'accès à la base

Entity : représentation d'une table

Source : https://developer.android.com/training/data-storage/room/

Créer une entité

✓ A l'aide d'**annotations** !

```
@Entity(tableName = "users",
        indices = {@Index(value = {"first_name", "last_name"},
        unique = true)})
public class User {

    @PrimaryKey(autoGenerate = true)
    private long id;

    @ColumnInfo(name = "first_name")
    private String firstName;

    @ColumnInfo(name = "last_name")
    private String lastName;

    @Ignore
    private List<Book> books;

    // Ajout des getters et setters
    …
```

@Entity description des propriétés de l'entité

@PrimaryKey précise la clé primaire

@ColumnInfo précise le nom de la colonne dans la table

@Ignore attributs non mappé en base

Pour en savoir plus,
https://developer.android.com/training/data-storage/room/defining-data

Créer un DAO - Data Access Object (1/2)

- ✓ Toujours des annotations

```
@Dao
public interface UserDao {

    @Query("SELECT * FROM users")
    List<User> getAll();

    @Query("SELECT COUNT(*) FROM users")
    int count();

    @Query("SELECT * FROM users WHERE id IN (:userIds)")
    List<User> loadAllByIds(long[] userIds);

    @Query("SELECT * FROM users WHERE first_name LIKE :first AND "
            + "last_name LIKE :last LIMIT 1")
    User findByName(String first, String last);

    @Query("SELECT * FROM users WHERE first_name LIKE :first")
    List<User> findByFirstName(String first);
```

Requête à la base mais résultat sous forme d'**objets**

ATTENTION : une classe pour **chaque entité !**

Créer un DAO - Data Access Object (2/2)

- ✓ Egalement des **annotations spécifiques**

```
@Insert
long insert(User user);

@Insert
long[] insertAll(User... users);

@Update
void update(User user);

@Delete
void delete(User user);
```

Pour en savoir plus,
https://developer.android.com/training/data-storage/room/accessing-data

Comment utiliser les entités et les DAO ?

- ✓ Dans les activités, créer des classes **AsyncTask** pour accéder aux données persistantes !

Démo projet TodoList (TP3)

- ✓ Une classe **AsyncTask** permet d'exécuter des tâches en arrière-plan (thread secondaire) tout en mettant à jour l'activité dans le thread principal

- ✓ POURQUOI ? Pour **éviter le gel de l'interface utilisateur UI**

AsyncTask (1/2)

Pour lancer une query, insert, etc. , il faut :

1. créer une classe héritant de la classe abstraite **AsyncTask** et implémenter les méthodes :

- ✓ doInBackground(Params... params)
  - Effectue la demande à la BD par l'intermédiaire d'un DAO (permet d'autres type de tâches lourdes, ex: requêtes réseau, traitements longs, etc.)
  - Exécutée dans un thread en arrière-plan

- ✓ onPostExecute(Result result)
  - Récupère le résultat de la tâche précédente et s'occupe de mettre à jour l'activité (données et interface utilisateur)
  - Exécutée sur le thread principal après la fin de doInBackground(…)

AsyncTask (2/2)

Pour lancer une query, insert, etc. , il faut :

2. créer un objet de la classe asynchrone et exécuter la demande

```
GetTasks gt = new GetTasks();
gt.execute();
```

Attention l'utilisation d'Asyntask est **dépréciée**. Mais elle permet de voir clairement l'usage asynchrone. Les plus aventureux peuvent se tourner vers RxJava pour la remplacer.

TP3 et début du Mini-projet

- ✓ TP3
  - Analyser le projet TodoList fourni
  - Mettre en place une gestion de compte dans un nouveau projet (AndroidProjet disponible sur Chamilo)
  - Le TP3 sert de lancement au mini-projet

- ✓ Mini-projet
  - Réaliser une application mobile destinée aux écoles primaires. Cette application permettra aux élèves de s'auto-évaluer sur des petits exercices mathématiques et des exercices à base de connaissances.

Installation de de Room

- ✓ Ajouter les dépendances nécessaires
  - dans le fichier build.gradle (Module :app)

```
dependencies {
    …
    annotationProcessor libs.room.compiler
    implementation libs.room.runtime
```

  - le fichier libs.versions.toml

```
[versions]
…
room = "2.6.1"

[libraries]
…
room-compiler = { group = "androidx.room", name = "room-compiler", version.ref = "room" }
room-runtime = { group = "androidx.room", name = "room-runtime", version.ref = "room" }
```

Pour en savoir plus sur Room

- ✓ Guide de la librairie Room

https://developer.android.com/topic/libraries/architecture/room

- ✓ Les remplaçants de AsyncTask

https://developer.android.com/training/data-storage/room/async-queries

- ✓ Migration d'une base de données

https://developer.android.com/training/data-storage/room/migrating-db-versions

- ✓ Test

https://developer.android.com/training/data-storage/room/testing-db

- ✓ Converters : permet de transformer des types de données non pris en charge nativement par Room (ex: Date, Enum, etc.) en types compatibles avec SQLite (et inversement)

https://developer.android.com/training/data-storage/room/referencing-data