Développeur d'application - Android

String Projet 5 =
“**Délivrez une application mobile en production**";

String P5 ="P5_PILI_Xavier";

SOMMAIRE

int L’application = 1 ;

Liste des fonctionnalités:

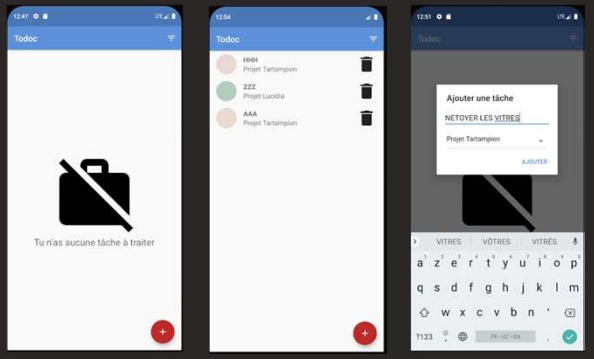
Une Couleur par projet, tri des taches: alphabétique, alphabétique inversé, de la plus ancienne à la plus récente et de la plus ancienne à la plus récente.

Application responsive sur toutes les tailles de téléphones et tablettes Android en modes portrait et paysage.

Application qui supporte Android 5.0 (API 21) et ses versions supérieures.
Code hébergé sur GitHub.

Tests unitaires pour chaque fonctionnalité.
Tests instrumentalisés.

Mission à réaliser,
Gérer la persistance des données de l’application,



int le Code de l'application = 2 ;

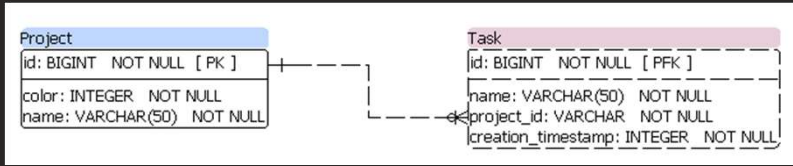
APK Signé: https://github.com/XP06/P5-todoc-master/tree/main/app/release

Dépôt Github : https://github.com/XP06/P5-todoc-master.git

int MDP Le Model Relationel = 3 ;

Relations :

One to One
One to many
Many to many
La classe d'association



1FN : première forme normale

Pour être en première forme normale (1FN), il faut que chaque attribut soit **atomique**. En d'autres termes, aucun attribut ne doit être multivalué (liste de valeurs) ou composé (si on le décompose, on obtient des informations supplémentaires).

2FN : deuxième forme normale

La deuxième forme normale (2FN) ne concerne que les tables avec une **clé primaire composite**.

Pour être en deuxième forme normale (2FN), il faut déjà être en 1FN et en plus respecter la règle suivante : aucun attribut ne faisant pas partie de la clé primaire ne doit dépendre que d'une partie de la clé primaire.

3FN : Troisième forme normale

La troisième forme normale (3FN) ressemble à la deuxième (2FN) mais concerne la dépendance entre attributs non clés.

Pour être en troisième forme normale (3FN), il faut déjà être en 2FN et en plus respecter la règle suivante : aucun attribut ne faisant pas partie de la clé primaire ne doit dépendre d'une partie des autres attributs ne faisant pas non plus partie de la clé primaire.

Agrégation

Quand une classe a un rôle qui correspond à un ensemble ou un regroupement d'objets, il peut être intéressant de mettre en valeur cet aspect.

En effet, cela permet de voir au premier coup d'œil qu'il s'agit d'un ensemble, sans avoir à se pencher sur les multiplicités de l'association (qui sont bien évidemment dans ce cas de type *un à plusieurs* ou *plusieurs à plusieurs*).

Cet aspect d'*ensemble* est modélisé par une association appelée **agrégation** et est matérialisée par un losange du côté de la classe jouant le rôle d'ensemble.

De même que pour l'agrégation, la **composition** sera traduite dans le MPD comme une association classique de type *un à plusieurs*. Il n'y a pas de formalisme particulier.

Composition

Une **composition** est une sorte d'agrégation plus « forte ».

Elle s'emploie lorsque :

- une classe (composite) est "composée" de plusieurs autres classes (composant) ;
- une instance de la classe composant ne peut pas être liée à plusieurs instances de la classe composite (association obligatoirement *un à plusieurs* et non pas *plusieurs à plusieurs*) ;
- si on détruit une instance de la classe composite, ses composants devrait "normalement" être détruits.

En approche orientée objet, il est possible de généraliser des comportements grâce à l'**héritage**.

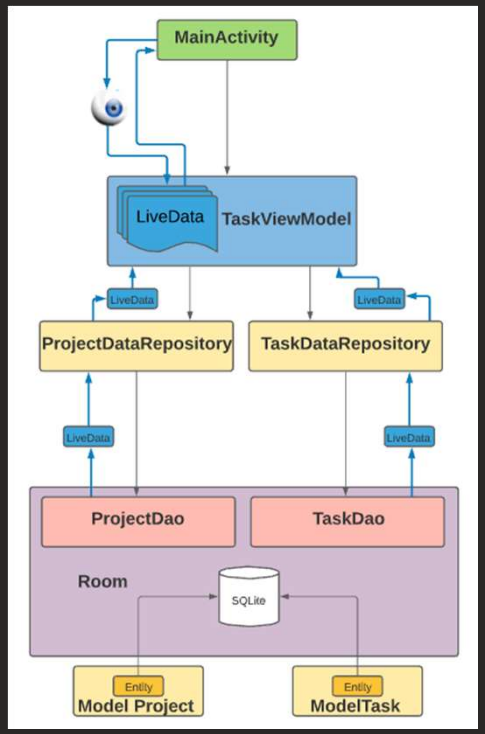
- Le passage de ces sous-classes (`Marteau` et `CleAPipe`) à la super-classe (`Outil`) est appelé **généralisation**.
- Le passage de la super-classe aux sous-classes est appelé **spécialisation**.

Grâce à l'héritage, vous allez pouvoir, entre autres :

- placer des attributs communs (dans la super-classe), qui seront « automatiquement » repris dans les sous-classes ;
- créer des associations communes au niveau de la super-classe, qui, de même, seront « automatiquement » reprises dans les sous-classes.

int Base de Données + Room = 4 ;

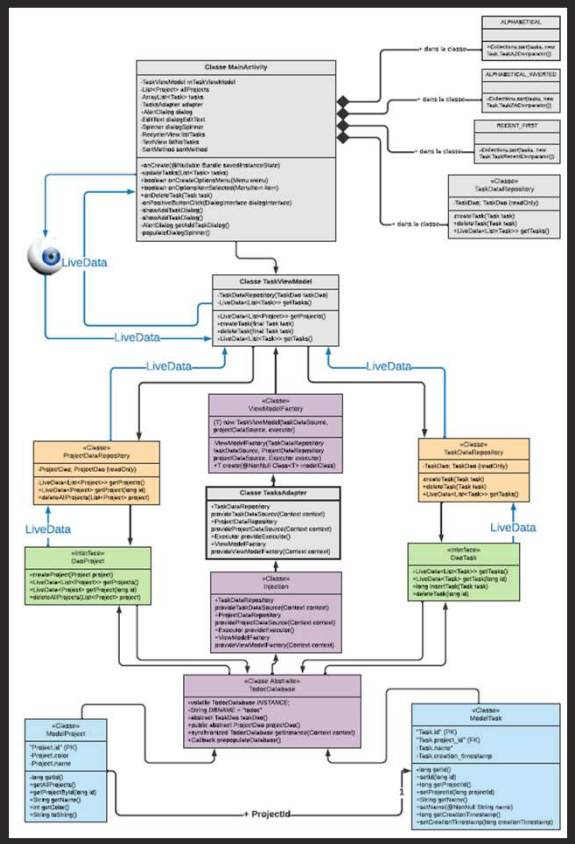
schéma de base de données



Utiliser une base de données peut se résumer à quatre opérations principales, le fameux **CRUD** :

- ***C**reate* : créer des données
- ***R**ead* : lire des données
- ***U**pdate* : modifier des données
- ***D**elete* : supprimer des données

int Diagramme de Classes = 4 ;



Le diagramme de classes est l'un des types de diagrammes UML les plus utiles, car il décrit clairement la structure d'un système particulier en modélisant ses classes, ses attributs, ses opérations et les relations entre ses objets.

```
public class test {

        public static void main(String[] args) {

                String merci = "MERCI";

        System.out.println(merci);
}
}
```