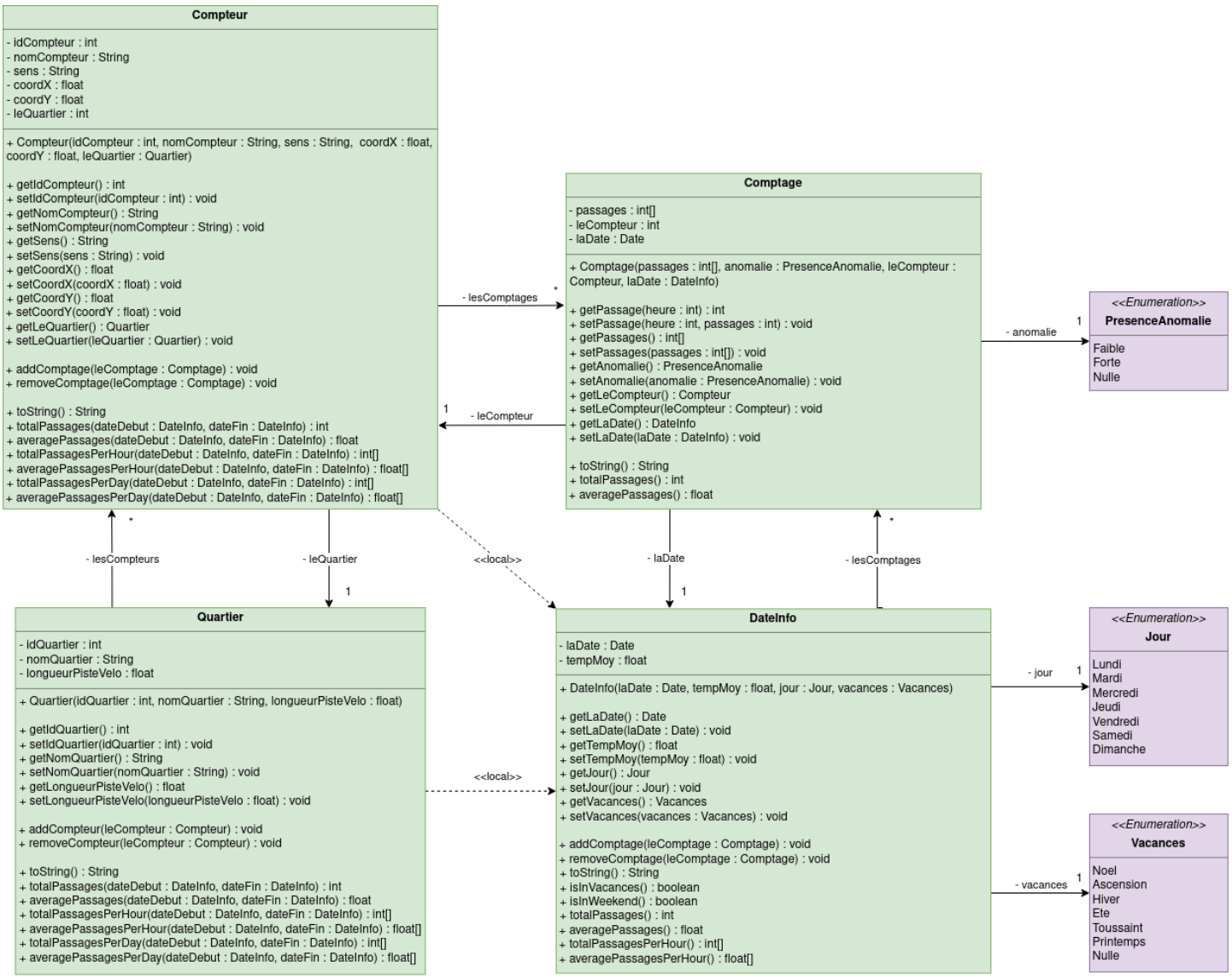
MELLION Jean-Loup
PERRUDIN Mathis
BOUILLIS Awen
NICOLAS Jules

Groupe 4B2

SAE Les vélos de Nantes
Partie Programmation Orientée Objet (R2.01)

Table des matières

Diagramme de classe sans connexion à MySQL



Le diagramme est également disponible dans le dossier /diagrammes.

Quartier :

- idQuartier : L'id du quartier. On utilise un int, car c'est un INT dans la base de données.
- *nomQuartier* : Le nom du quartier. On utilise un String, car c'est un VARCHAR dans la base de données.
- *longueurPisteVelo* : La longueur de la piste cyclable. On utilise un float, car c'est un FLOAT dans la base de données.
- *lesCompteurs* : Les compteurs associés. On utilise une liste de Compteurs, en plus de la référence vers le quartier, pour simplifier l'accès aux données.
- Les différentes méthodes ajoutées (hors toString) sont utilisées dans l'application JavaFX pour récupérer les informations des passages avec différents filtres. Ici pour le quartier, nous avons par exemple averagePassages, qui prend en paramètre les deux dates limites des données que nous voulons utiliser (comme toutes les autres méthodes). Cette méthode utilise ensuite la méthode averagePassages de ses compteurs associés.

Compteur :

- *idCompteur* : L'id du compteur. On utilise un int comme dans la base de données.
- *nomCompteur* : Le nom du compteur. On utilise un String, car c'est un VARCHAR dans la base de données.
- *sens* : Le sens du compteur. On utilise un String, car c'est un VARCHAR dans la base de données. On n'utilise pas d'enum, car il y a plus de sens que les simples 4 directions.
- *coordX* : La coordonnée X (latitude) du compteur. On utilise un float, car c'est un FLOAT dans la base de données.
- *coordY* : La coordonnée Y (longitude) du compteur. On utilise un float, car c'est un FLOAT dans la base de données.
- *leQuartier* : Le quartier du compteur. On utilise une référence vers un objet Quartier, car c'est une FOREIGN KEY dans la base de données.
- *lesComptages* : Les comptages associés. On utilise une liste de Comptages, en plus de la référence vers le compteur, pour simplifier l'accès aux données.
- Les différentes méthodes ajoutées sont exactement celles du Quartier, sauf que ce sont les méthodes du Compteur qui font le tri des comptages à partir des dates, tandis que les méthodes du Quartier appellent simplement les méthodes du Compteur.

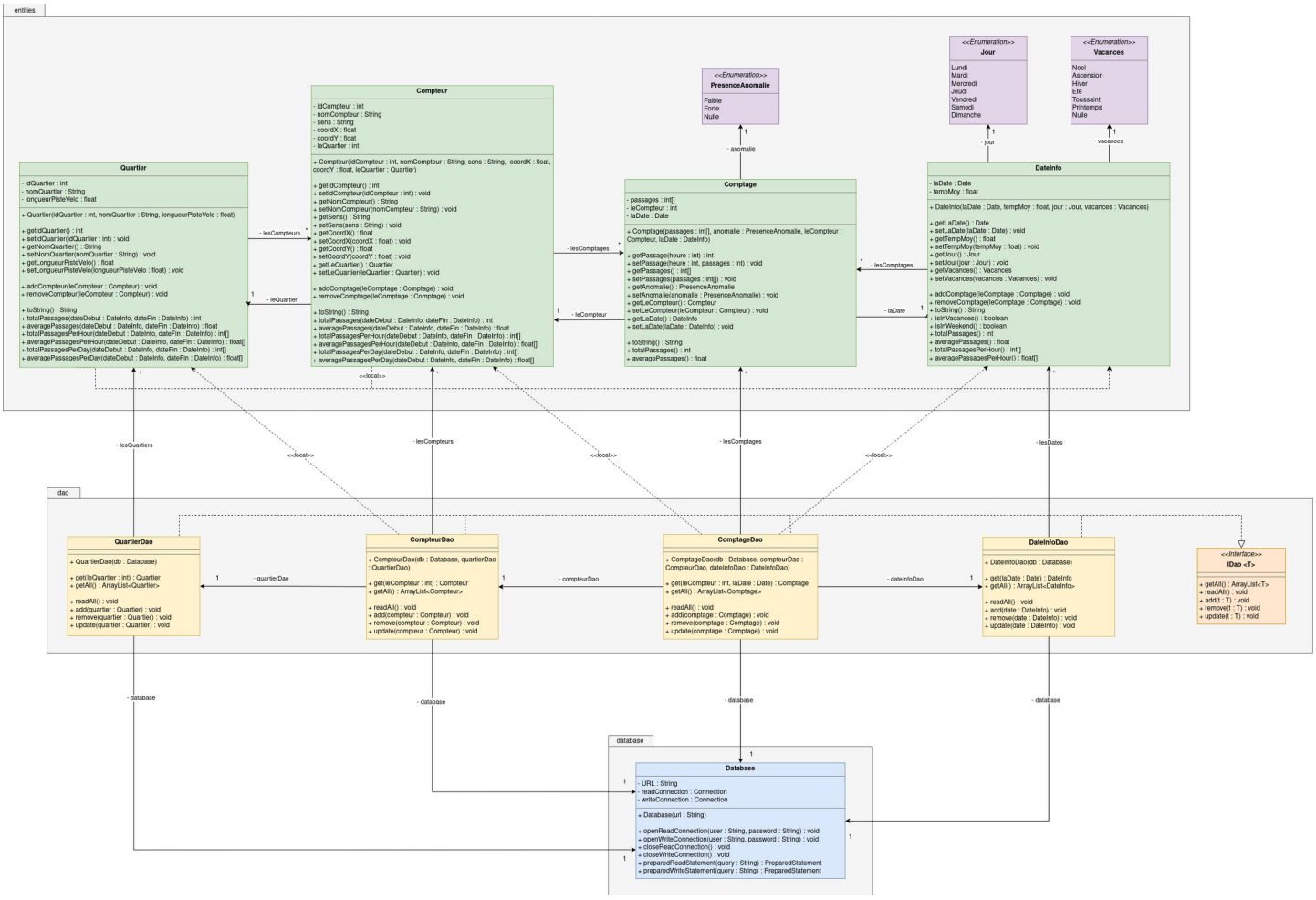
DateInfo :

- *date* : La date. On utilise la date de java.sql car elle est compatible avec JDBC.
- *tempMoy* : La température moyenne. On utilise un float, car c'est un FLOAT dans la base de données.
- *jour* : Le jour associé. On utilise un enum, car on a seulement 7 valeurs possibles.
- *vacances* : Les vacances associées. On utilise un enum, car il n'y a que 6 vacances différentes + 1 pour la valeur "Nulle".
- *lesComptages* : Les comptages associés. On utilise une liste de Comptages, en plus de la référence vers la date, pour simplifier l'accès aux données.
- Les méthodes de DateInfo permettent de savoir simplement si la date est en weekend (samedi, dimanche), ou en vacances. Il y a aussi des méthodes qui ressemblent à celles de Quartier et Compteur, permettant de calculer soit le total soit la moyenne des passages, par heure ou non, mais seulement pour la date de DateInfo et non pas une plage de dates.

Comptage :

- *passages* : Le nombre de passages par heures. Comme on aura toujours 24 heures, on peut utiliser un tableau de taille 24 au lieu d'une ArrayList.
- *anomalie* : La présence d'anomalie dans les données. Vu que l'on a que 3 valeurs possibles (Forte, Faible, Nulle), on peut utiliser un enum.
- *leCompteur* : Le compteur associé. On utilise une référence vers un objet Compteur, car c'est une FOREIGN KEY dans la base de données.
- *laDate* : La date associée. On utilise une référence vers la date associée, reproduisant ainsi la FOREIGN KEY de la table Comptage.
- Les méthodes de Comptage sont :
  - toString(), dans toutes les autres classes, renvoie une représentation de la classe en String.
  - totalPassages, qui calcule le total des passages des 24h, utilisée par les autres classes.
  - averagePassages, qui appelle simplement totalPassages avec une division par 24 pour avoir la moyenne des passages par heure. Cette méthode est également utilisée par les autres classes.

Diagramme de classe avec connexion à MySQL

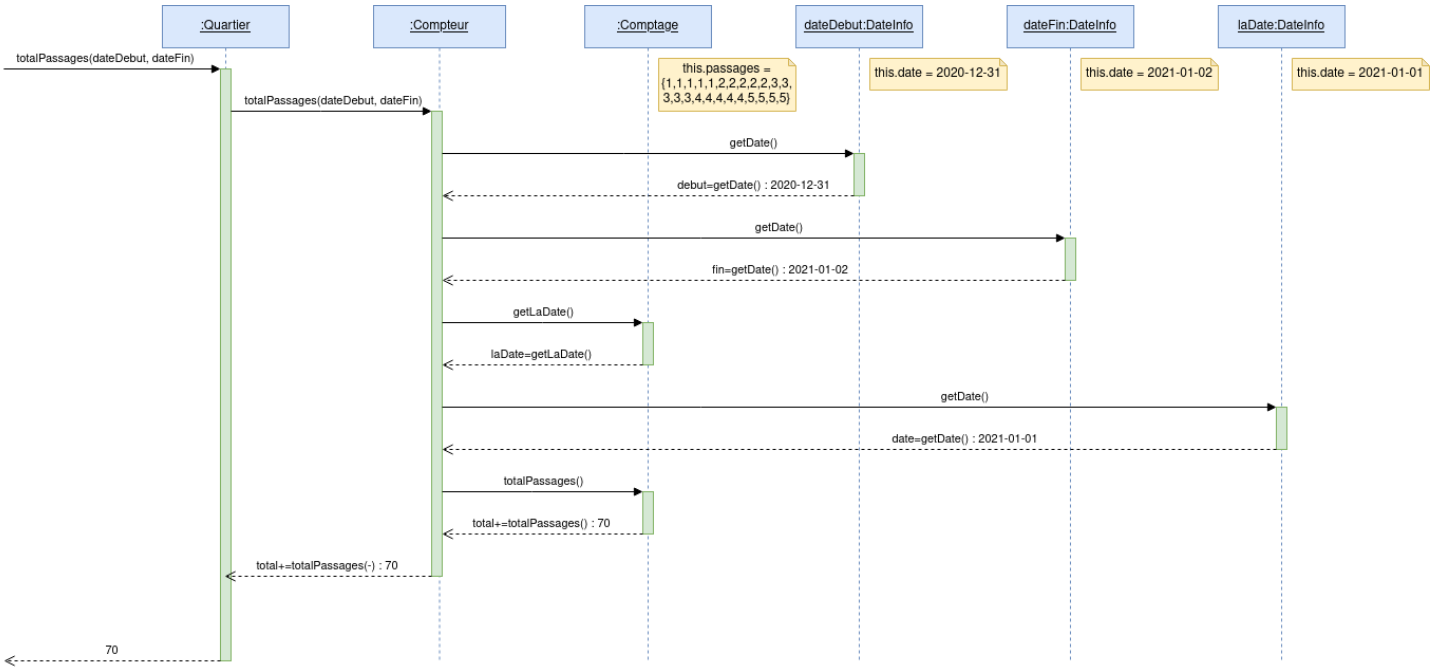


Le diagramme est également disponible dans le dossier /diagrammes.

Nous avons divisé le modèle en 3 sous-packages du package "modele".

- “entities” qui correspond au diagramme de conception de la partie BDD.
- “dao” qui contient les classes DAO associées aux différentes tables dans “entities”. Nous avons utilisé le design DAO, car cela permet de séparer les différentes tables de tout ce qui concerne la connexion avec MySQL.
- Enfin, nous avons le package “database” qui contient la connexion à MySQL (ou MariaDB) et qui permet aux classes DAO de fonctionner.

Diagramme de séquence de la méthode Quartier.totalPassages



Le diagramme est également disponible dans le dossier /diagrammes.

Nous avons choisi cette méthode car elle représente bien les liens entre toutes les classes du modèle. Ici il n'y a que un compteur dans le quartier, et un comptage dans le compteur.

Code complet

Le code complet est disponible dans le dossier */src/modele*. La javadoc est en anglais.

Fichier de test de la classe Comptage

Nous avons choisi la classe Comptage pour le fichier de test. Il utilise JUnit. Il est disponible dans */src/TestComptage.java*.

Fichier de scenario

Le fichier de scénario est disponible dans */src/ScenarioMySQL.java ou /src/ScenarioMariaDB*. Le fichier de scénario initialise toutes les tables à partir de la base de données MySQL, et imprime dans le fichier output.txt tous les toString() de toutes les instances. Il utilise ensuite les méthodes qui sont utilisées dans JavaFX pour filtrer les données de passages en fonction des dates, des quartiers et des compteurs. Il apporte ensuite des modifications sur les données et les sauvegarde ensuite sur la base de données MySQL.

/!\ Comme elle apporte des modifications à la base de données, le scénario marche moins bien sur la partie modification / insertion avec la base de donnée si il y a une deuxième execution du scénario. Pour initialiser la base de donnée, les scripts sont disponnible dans */sql. L*a database est "*bd_velo_4b2*", et les users sont "read_4b2" et "write_4b2" pour éviter les conflits.