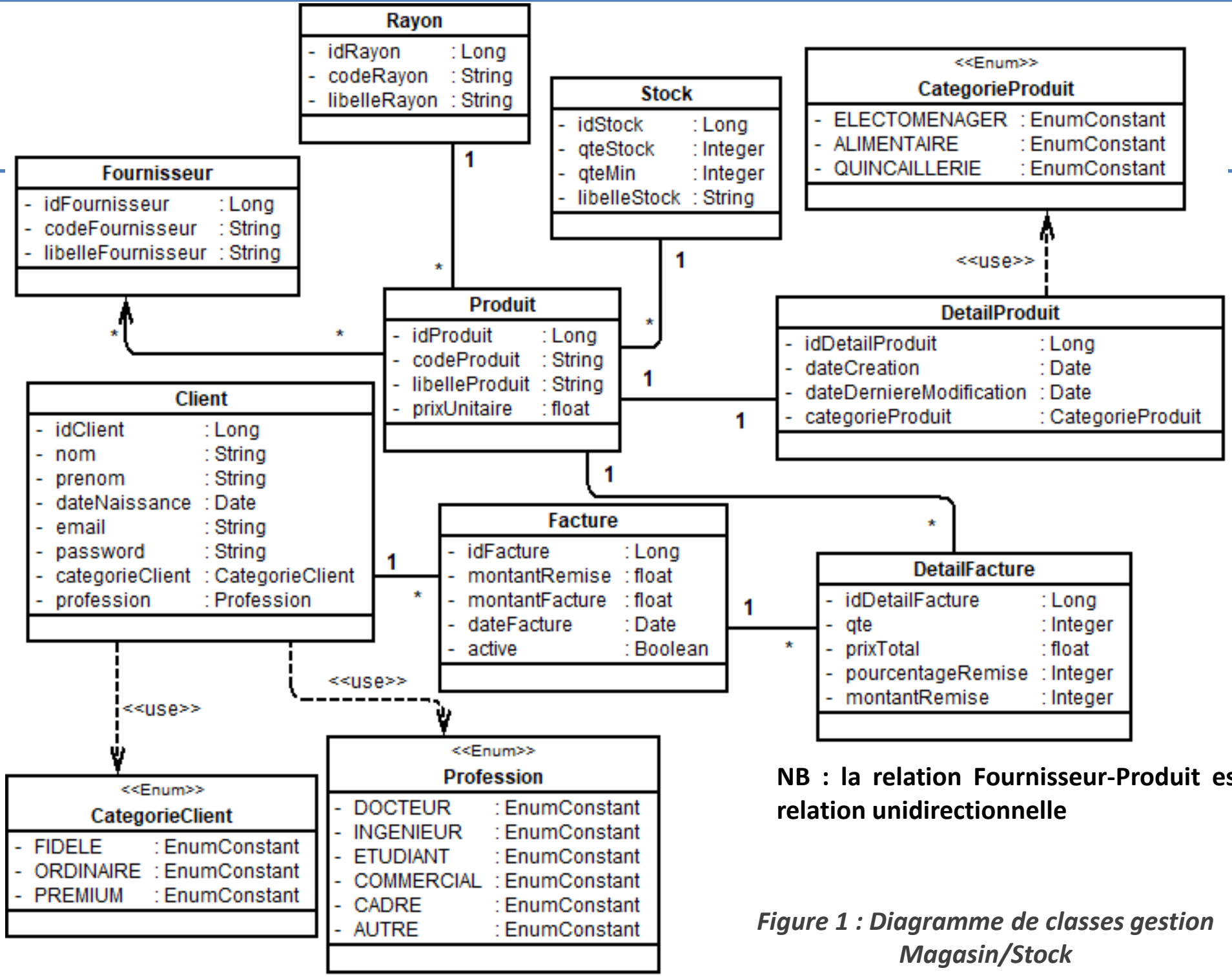
TP étude de cas
Gestion Magasin-Stock

UP ASI
Bureau E204

TP Gestion Magasin-Stock

- On désire créer une application de gestion d'un magasin.
- Le magasin contient des produits répartis sur des rayons.
- Le stock de produit est géré de façon à assurer le suivi des mouvements des produits et détecter rapidement les stocks à alimenter.
- Les clients peuvent commander un ensemble de produits sous forme d'une facture affichant le montant total à payer. Les détails de la facture sont également présentés(prix de chaque produit, quantité commandé, montant remise).
- L'application nous permet également d'afficher des métriques intéressantes tel que le revenu brut généré par un produit ou le chiffre d'affaire généré par une catégorie client bien spécifique (premuim, fidèle, ordinaire)



NB : la relation Fournisseur-Produit est une relation unidirectionnelle

Figure 1 : Diagramme de classes gestion Magasin/Stock

Travail à faire

Partie 1 Spring Data JPA – Première entité

- Créer les entités se trouvant dans le diagramme des classes (sans les associations) et vérifier qu'ils ont été ajoutés avec succès dans la base de données.

Travail à faire

Partie 2 Spring Data JPA – Le mapping des différentes associations

- Supprimer les tables existantes dans la base de données.
- Créer les associations entre les différentes entités.
- Générer la base de donné de nouveau et vérifier que le nombre de tables crées est correcte.

Travail à faire

Partie 3 Spring Data JPA CRUD Repository–Le langage JPQL - JPA Repository

Créer les CRUD des différentes **entités en respectant les signatures suivantes** (à l'exception de l'entité fournisseur et rayon. Pour ces deux entités, vous pouvez insérer les données dans la base de donnée manuellement)

Entité Client

```
List<Client> retrieveAllClients();
Client addClient(Client c);
void deleteClient(Long id);
Client updateClient(Client c);
Client retrieveClient(Long id);
```

Travail à faire

Partie 3 Spring Data JPA CRUD Repository–Le langage JPQL - JPA Repository

Entité Stock

List<Stock> retrieveAllStocks();

Stock addStock(Stock s);

Stock updateStock(Stock u);

Stock retrieveStock(Long id);

Travail à faire

Partie 3 Spring Data JPA CRUD Repository–Le langage JPQL - JPA Repository

Entité Produit

List<Produit> retrieveAllProduits();

Produit addProduit(Produit p, Long idRayon, Long idStock);

Produit retrieveProduit(Long id);

NB: Pour l'ajout de produit, il faut créer en même temps le détail produit (entité produit avec l'entité associé detailProduit)

Travail à faire

Partie 3 Spring Data JPA CRUD Repository–Le langage JPQL - JPA Repository

Entité Facture

List<Facture> retrieveAllFactures();
void cancelFacture(Long id);
Facture retrieveFacture(Long id);

NB : pour l'annulation de la facture, il faut que le champs active de la table facture soit mis à false.

Travail à faire

Partie 4 Spring MVC

Exposer les services implémentés dans la partie 3 avec Postman et/ou Swagger pour les tester.

Travail à faire

Partie 5 : Services avancés

Le chef rayon désire changer le stock d'un produit.
Créer un service permettant l'assignation d'un produit à un stock et exposer le en respectant la signature suivante :

void assignProduitToStock(Long idProduit, Long idStock);

Travail à faire

Partie 5 : Services avancés

Le chef rayon désire affecter un fournisseur au produit.
Créer un service permettant l'assignation d'un fournisseur à un produit et exposer le en respectant la signature suivante :

public void assignFournisseurToProduit(Long fournisseurId, Long produitId) ;

Travail à faire

Partie 5 : Services avancés

On souhaite récupérer les factures d'un client donné.
Créer le service adéquat en respectant la signature suivante :

List<Facture> getFacturesByClient(Long idClient);

Travail à faire

Partie 5 : Services avancés

Un client souhaite acheter deux produits distincts. Une facture doit lui etre fourni avec toutes les informations nécessaires.

Créer un service permettant de créer la facture avec les détails associés assignée au client concerné en respectant la signature suivante :

Facture addFacture(Facture f, Long idClient);

PS : le calcul des montants facture et remise de l'entité **Facture** ainsi que le montant Remise et le prix total de l'entité **detailFacture** doit etre fait convenablement.

Travail à faire

Partie 5 : Services avancés

Nous souhaitons calculer le chiffre d'affaire généré par une catégorie de client spécifique (Fidèle, Ordinaire, Premuim) entre deux dates.
Créer un service permettant de faire le calcul en respectant la signature suivante :

public float getChiffreAffaireParCategorieClient(CategorieClient categorieClient, Date startDate, Date endDate) {

PS : le chiffre d'affaire correspond aux montants dépensés par une catégorie client pour ses diverses factures.

Travail à faire

Partie 5 : Services avancés

Nous souhaitons calculer le revenu brut généré par un produit entre deux dates. Créer un service permettant de faire le calcul en respectant la signature suivante :

float getRevenuBrutProduit(Long idProduit, Date startDate, Date endDate);

PS : le revenu brut généré correspond aux montants générées par la vente de ce produit (prix * quantité pour les différentes factures)

Travail à faire

Partie 6 : Spring Scheduler

Nous souhaitons créer un service schedulé (programmé automatiquement) permettant d'avertir le responsable magasin des produits dont la quantité de stock disponible est inférieure à la quantité min toléré.

Créer un service nous permettant d'afficher les produits concernés tous les jours à 22h en respectant la signature suivante :

String retrieveStatusStock();

NB: Pour des raisons de test, vous pouvez modifier l'horaire selon l'heure affiché sur votre machine. Le message sera affiché simplement sur console.

TP Gestion Magasin-Stock

Si vous avez des questions, n'hésitez pas à nous contacter :

Département Informatique
UP Architectures des Systèmes d'Information

Bureau E204