Sujet : Meal Delivery Application
Auteur : ABED Mohamed Aziz
Date : April 20, 2025

Conception Préliminaire 2 : Meal Delivery Application

1. Raffinement (ajout de contraintes) du diagramme de classes pour faciliter la programmation

a. Navigabilité des associations

- Client --- passe --- Invoice : Cette association est unidirectionnelle (de Client vers Invoice). Un client doit pouvoir accéder à ses factures, mais une facture n'a pas besoin de pointer directement vers un client (le username dans Invoice suffit). Cela simplifie la gestion des références.
- DeliveryWorker --- gère --- Invoice : Unidirectionnelle (de DeliveryWorker vers Invoice). Un livreur accède aux factures qu'il gère, mais une facture n'a pas besoin de pointer vers le livreur.
- Invoice --- contient --- Order : Unidirectionnelle (de Invoice vers Order). Une facture contient plusieurs commandes, mais une commande n'a pas besoin de pointer directement vers la facture.
- Order --- référence --- Meal : Unidirectionnelle (de Order vers Meal). Une commande référence un repas, mais un repas n'a pas besoin de pointer vers les commandes.

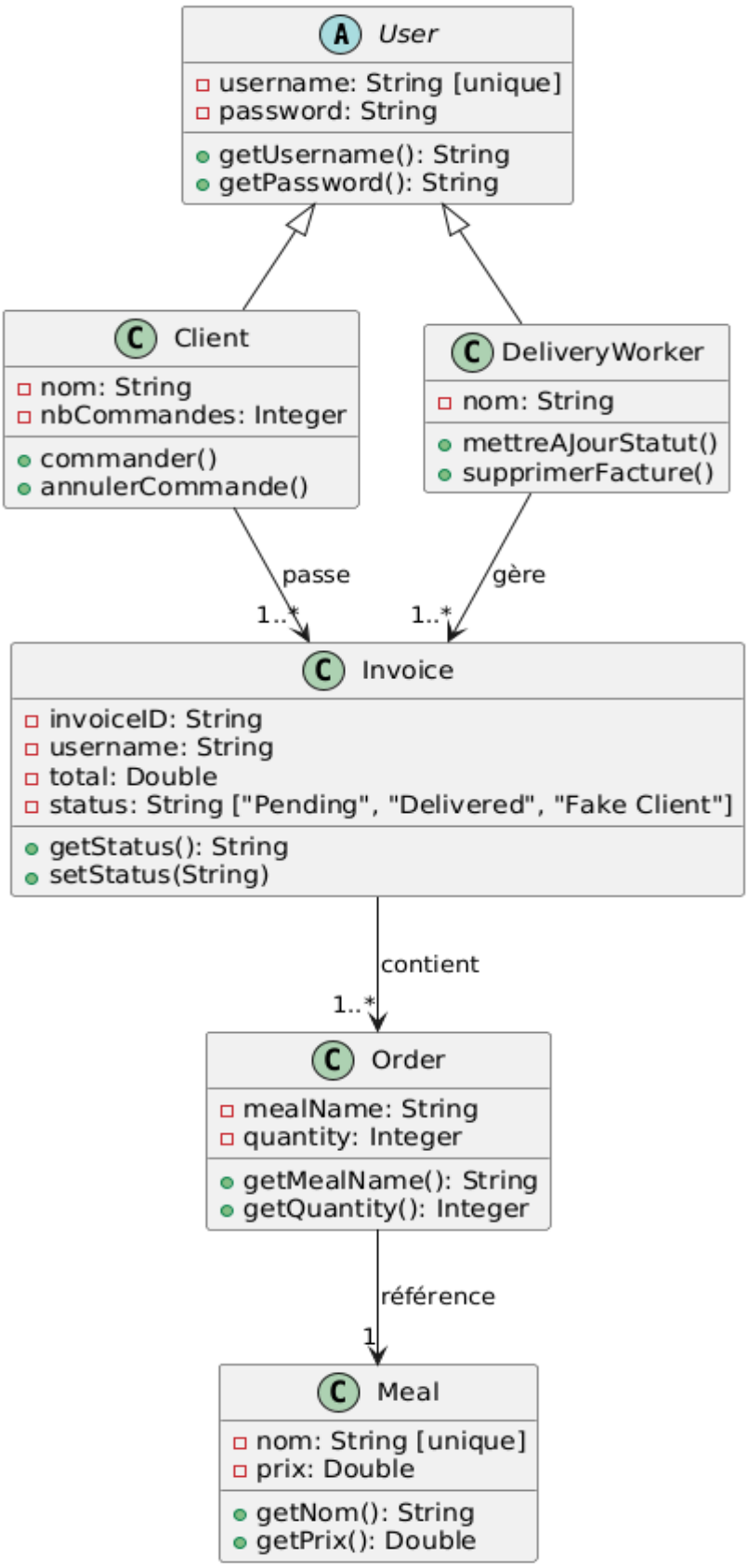
b. Encapsulation et visibilité

- Les attributs doivent être protégés avec une visibilité appropriée :
  - Attributs comme username, password (dans User), nom, prix (dans Meal), etc., doivent être privés (- en UML).
  - Les opérations comme commander(), annulerCommande(), ou supprimerFacture() doivent être publiques (+ en UML).
- Les constructeurs protégés peuvent être utilisés dans les sous-classes (Client, DeliveryWorker) pour redéfinir la construction à partir de User.

c. Contraintes supplémentaires

- Unicité : Les attributs username (dans User) et nom (dans Meal) doivent être uniques, comme spécifié dans les contraintes du projet.
- Statut de la facture : L'attribut status de Invoice est limité aux valeurs "Pending", "Delivered", "Fake Client".
- Annulation : Seules les factures en statut "Pending" peuvent être annulées (contrainte fonctionnelle à intégrer dans le modèle)

d. Diagramme de classes raffiné :



User (abstraite)
- username: String [unique]
- password: String

+ getUsername(): String
+ getPassword(): String

Client hérite de User
- nom: String
- nbCommandes: Integer
+ commander()
+ annulerCommande()

DeliveryWorker hérite de User
- nom: String
+ mettreAJourStatut()
+ supprimerFacture()

Meal
- nom: String [unique]
- prix: Double
+ getNom(): String
+ getPrix(): Double

Invoice
- invoiceID: String
- username: String
- total: Double
- status: String ["Pending", "Delivered", "Fake Client"]
+ getStatus(): String
+ setStatus(String)

Order
- mealName: String
- quantity: Integer
+ getMealName(): String
+ getQuantity(): Integer

Associations :
Client --- passe --- Invoice [1 --- *] (unidirectionnelle)
DeliveryWorker --- gère --- Invoice [1 --- *] (unidirectionnelle)
Invoice --- contient --- Order [1 --- *] (unidirectionnelle)
Order --- référence --- Meal [* --- 1] (unidirectionnelle)

2. Modéliser le cycle de vie des objets d'une classe principale dans un diagramme de machines états

Prenons la classe **Invoice** (facture), qui est une classe principale dans ce contexte, car elle est au cœur des interactions entre clients et livreurs. Le cycle de vie d'une facture est influencé par les actions du client (commander, annuler) et du livreur (mettre à jour le statut, supprimer).

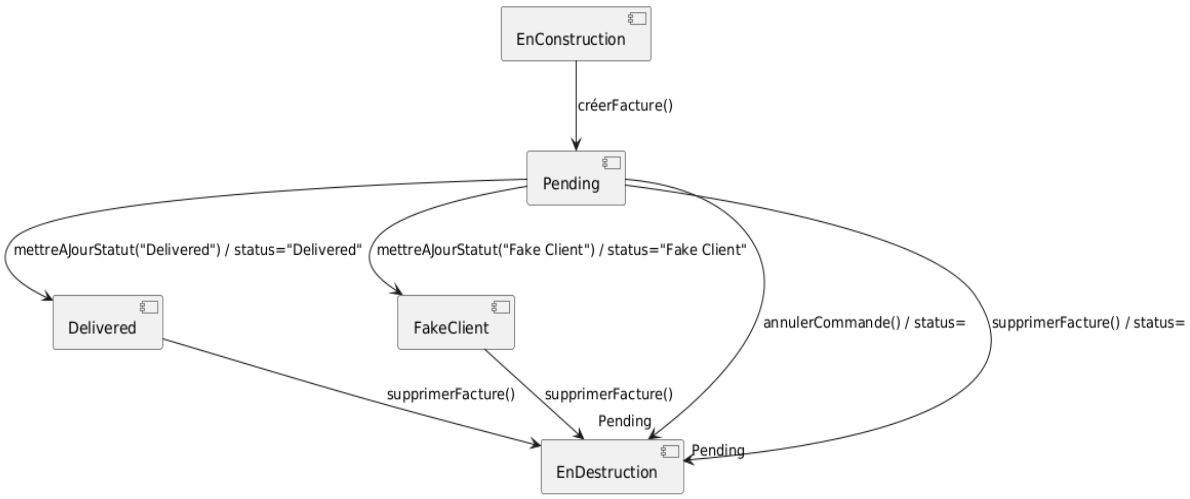
a. Identification des états significatifs

- EnConstruction : État initial lors de la création de la facture (lors d'une commande)
- Pending : La facture est en attente de livraison (status = "Pending").
- Delivered : La facture est livrée (status = "Delivered").
- FakeClient : Le client est marqué comme faux (status = "Fake Client").
- EnDestruction : État final lors de la suppression de la facture (annulation ou suppression).

b. Transitions entre états

- EnConstruction → Pending : Après la création de la facture (lors de la commande).
- Pending → Delivered : Événement mettreAJourStatut("Delivered") par le livreur, condition : status = "Pending", action : status = "Delivered".
- Pending → FakeClient : Événement mettreAJourStatut("Fake Client") par le livreur, condition : status = "Pending", action : status = "Fake Client".
- Pending → EnDestruction : Événement annulerCommande() par le client ou supprimerFacture() par le livreur, condition : status = "Pending".
- {Delivered, FakeClient} → EnDestruction : Événement supprimerFacture() par le livreur.

c. Transitions entre états



[EnConstruction] --(créerFacture())--> [Pending]
[Pending] --(mettreAJourStatut("Delivered")[status="Pending"]/status="Delivered")--> [Delivered]
[Pending] --(mettreAJourStatut("Fake Client")[status="Pending"]/status="Fake Client")--> [FakeClient]
[Pending] --(annulerCommande()[status="Pending"])--> [EnDestruction]
[Pending] --(supprimerFacture()[status="Pending"])--> [EnDestruction]
[Delivered] --(supprimerFacture())--> [EnDestruction]

[FakeClient] --(supprimerFacture())--> [EnDestruction]

3. Effectuer le 2ème raffinement

a. Traduction des associations en attributs

- Client --- passe --- Invoice : Unidirectionnelle, multiplicité 1 --- *. Dans Client, on ajoute un attribut factures: List<Invoice>. Pas d'attribut inverse dans Invoice (le username suffit).
- DeliveryWorker --- gère --- Invoice : Unidirectionnelle, multiplicité 1 --- *. Dans DeliveryWorker, on ajoute un attribut facturesGérées: List<Invoice> (bien que cela puisse être implicite via une requête sur toutes les factures).
- Invoice --- contient --- Order : Unidirectionnelle, multiplicité 1 --- *. Dans Invoice, on ajoute un attribut commandes: List<Order>. Pas d'attribut inverse dans Order.
- Order --- référence --- Meal : Unidirectionnelle, multiplicité * --- 1. Dans Order, l'attribut mealName: String est déjà une référence au repas (par son nom). Pas d'attribut inverse dans Meal.

Classes mises à jour :

- Client : Ajout de factures: List<Invoice>.
- DeliveryWorker : Ajout de facturesGérées: List<Invoice> (optionnel, selon l'implémentation).
- Invoice : Ajout de commandes: List<Order>.

b. Traduction des agrégations/compositions

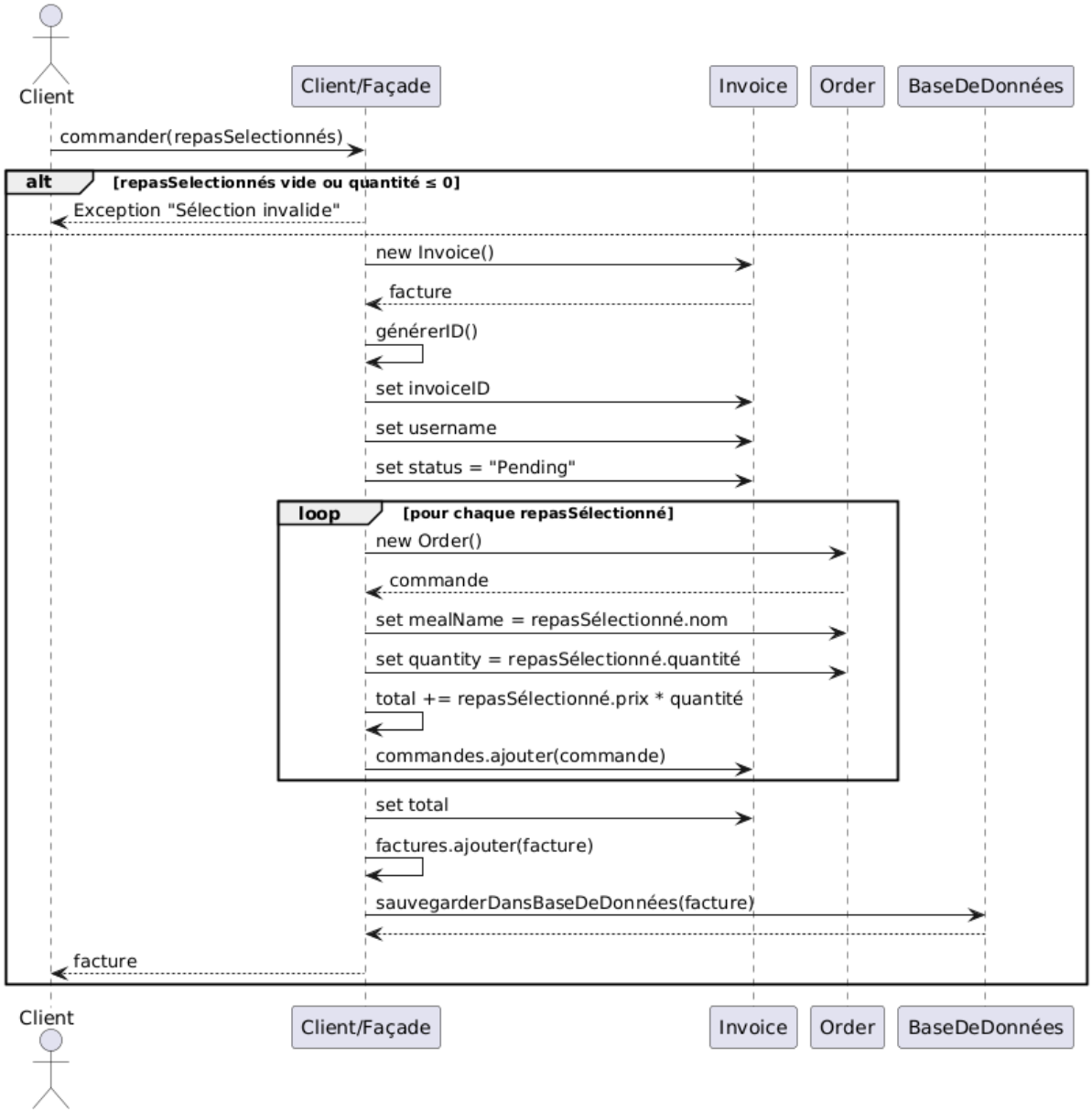
- Invoice --- contient --- Order : C'est une composition, car les commandes (Order) n'ont de sens que dans le contexte d'une facture (Invoice). Si une facture est supprimée, les commandes associées doivent l'être aussi.
    - Traduction : Invoice a un attribut commandes: List<Order>, et le destructeur de Invoice doit supprimer les objets Order associés.

c. Traduction des diagrammes de séquence et machines-états en algorithmes

DSUC1 : Client Commande des Repas :

Description textuelle :

- Le client sélectionne des repas et clique sur "Commander".
- La façade crée une facture avec le total.
- Pour chaque repas sélectionné, une commande est ajoutée avec la quantité.
- La facture est affichée au client.



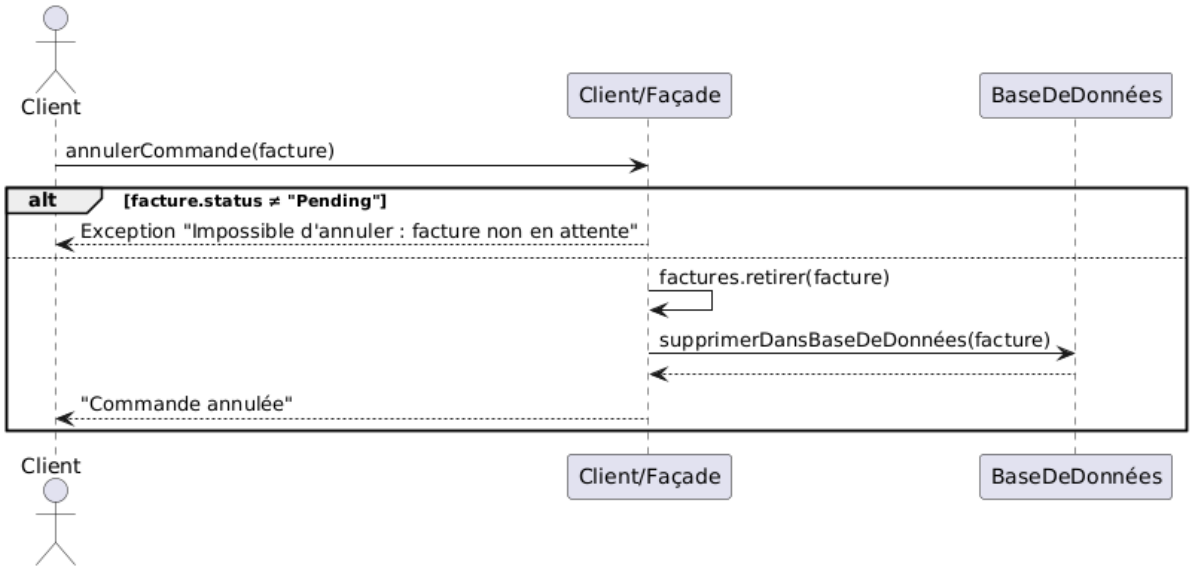

```
// Dans Client ou Façade
commander(List<RepasSelectionnés> repasSelectionnés):
    si repasSelectionnés est vide ou ∃ quantité ≤ 0 alors
        lever une exception "Sélection invalide"
    facture = nouvelle Invoice()
    facture.invoiceID = générerID()
    facture.username = this.username
    facture.status = "Pending"
    total = 0
    pour chaque repasSélectionné dans repasSelectionnés:
        commande = nouvelle Order()
        commande.mealName = repasSélectionné.nom
        commande.quantity = repasSélectionné.quantité
```

```
        total += repasSélectionné.prix * repasSélectionné.quantité
        facture.commandes.ajouter(commande)
    facture.total = total
    this.factures.ajouter(facture)
    sauvegarderDansBaseDeDonnées(facture)
    retourner facture
```

DSUC2 : Client Annule une Commande :

Description textuelle :

- Le client sélectionne une commande et clique sur "Cancel".
- La façade vérifie si l'état est "Pending".
- Si oui, les commandes liées et la facture sont supprimées.
- Un message de confirmation ou d'erreur est affiché.



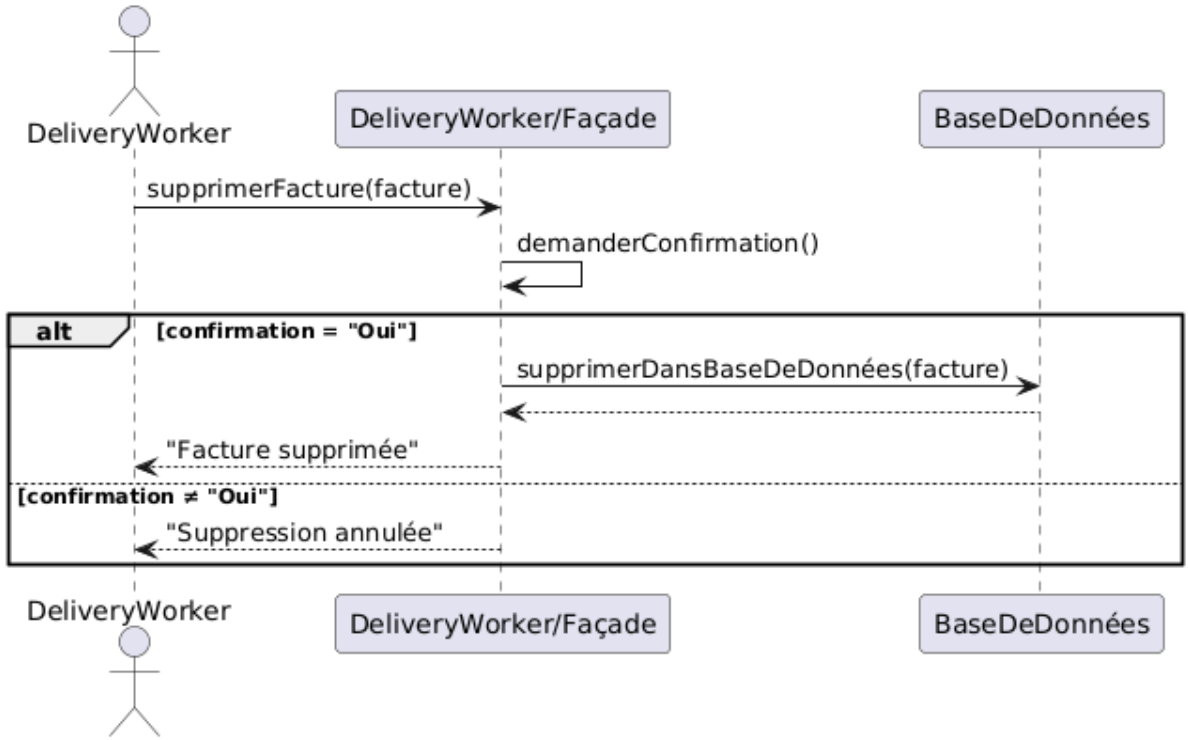

```
// Dans Client ou Façade
annulerCommande(Invoice facture):
    si facture.status ≠ "Pending" alors
        lever une exception "Impossible d'annuler : facture non en attente"
    this.factures.retirer(facture)
    supprimerDansBaseDeDonnées(facture) // Supprime aussi les commandes associées
    retourner "Commande annulée"
```

DSUC3 : Livreur Supprime une Facture :

Description textuelle :

- Le livreur sélectionne une facture et clique sur "Delete".
- Une confirmation est demandée.
- Si "Oui", les commandes liées et la facture sont supprimées.

- Un message de confirmation est affiché.




```
// Dans DeliveryWorker ou Façade
supprimerFacture(Invoice facture):
    confirmation = demanderConfirmation()
    si confirmation = "Oui" alors
        supprimerDansBaseDeDonnées(facture) // Supprime aussi les commandes
associées
        retourner "Facture supprimée"
    sinon
        retourner "Suppression annulée"
```