UCR: Design Pattern GRASP Analyse 3

2020-2021

PLAN

MCD
Modèle conceptuel des données
Diagramme de classes
(rappels) Documentation

MCT
Modèle conceptuel des traitements
Diagramme de Use Cases (UC)
Documentation

Conceptuel

UC Specification
Documentation de UC
Interface utilisateur
Diagramme d'activité (rappel)

PTFE
Plan de tests fonctionnels élémentaires
Documentation

Fonctionnel

MTD-MTT

Design Pattern

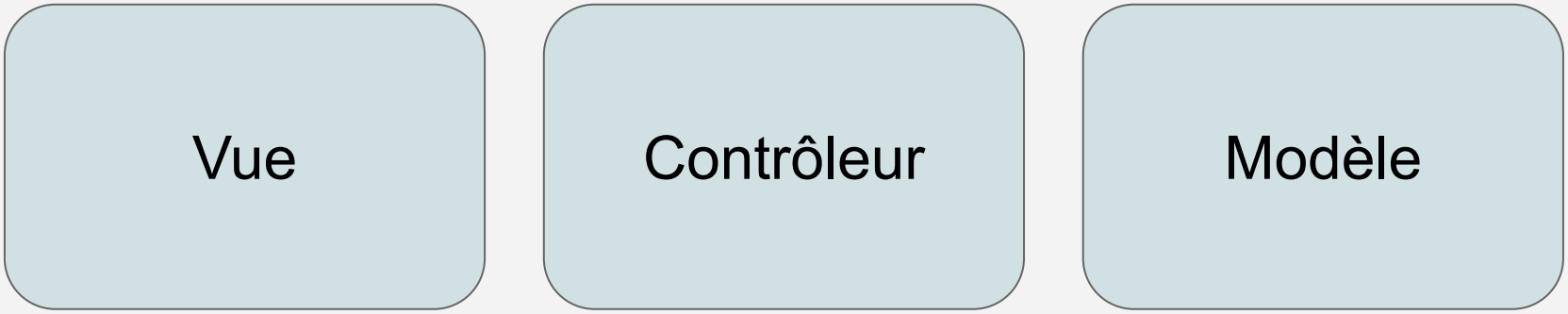
UC Realization
Diagramme de séquence
Diagramme de classes techniques

Technique

Méthodes

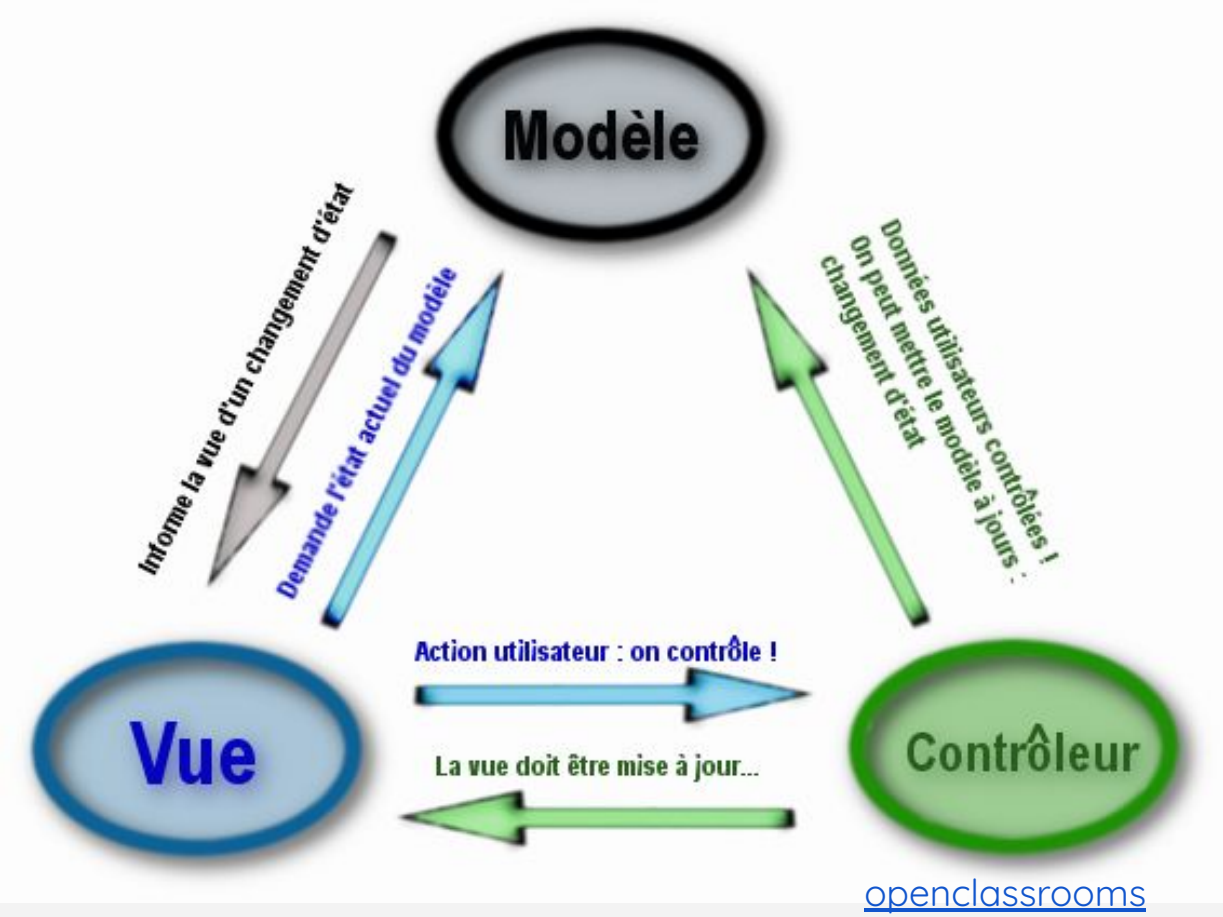
UCR - Design Pattern MVP

Modèle - Vue - Contrôleur



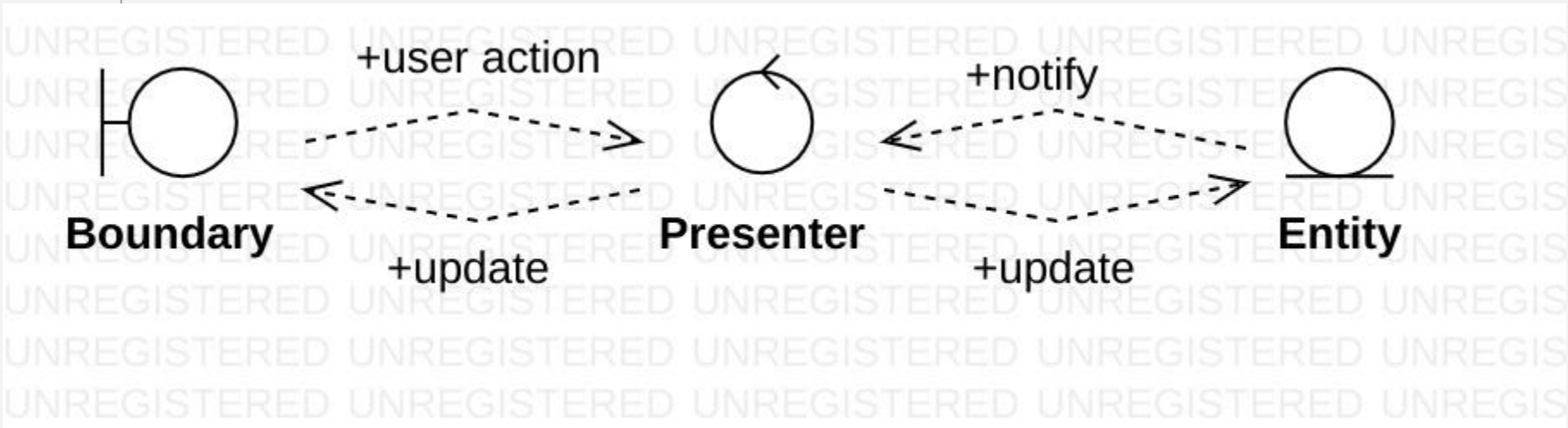
- Vue: IHM - interface graphique.
- Contrôleur: la logique du système - du UC.
- Modèle: les données et leurs logiques.

Modèle - Vue - Contrôleur



openclassrooms

Modèle - Vue - Présentation



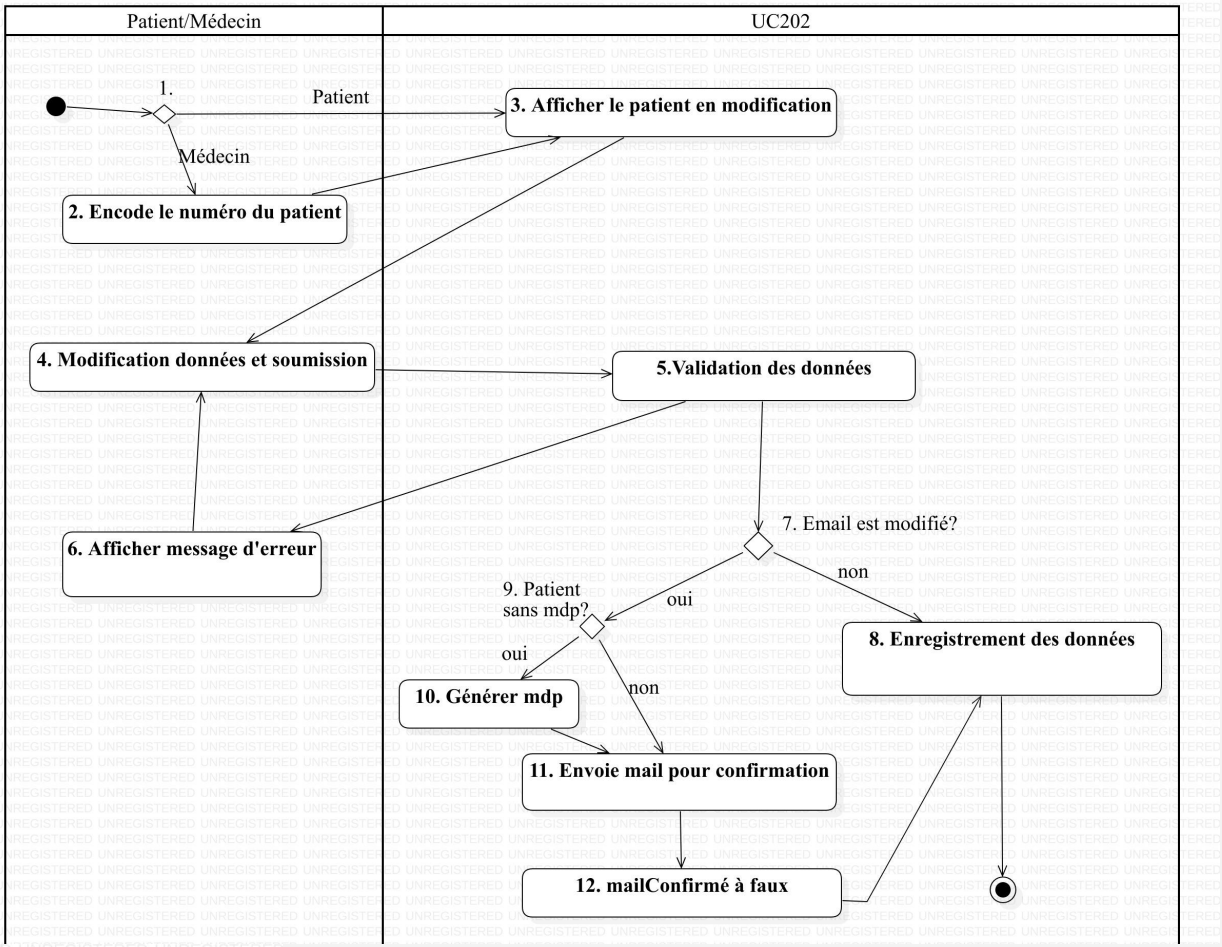
- La vue est totalement indépendante du reste de l'application et est débarrassée de toute logique.
- Le modèle est uniquement centré sur la logique métier. Il est totalement indépendant des autres parties.
- La présentation contient toute la logique de présentation ainsi que l'état courant du dialogue avec l'utilisateur. Elle est indépendante de la bibliothèque graphique utilisée.

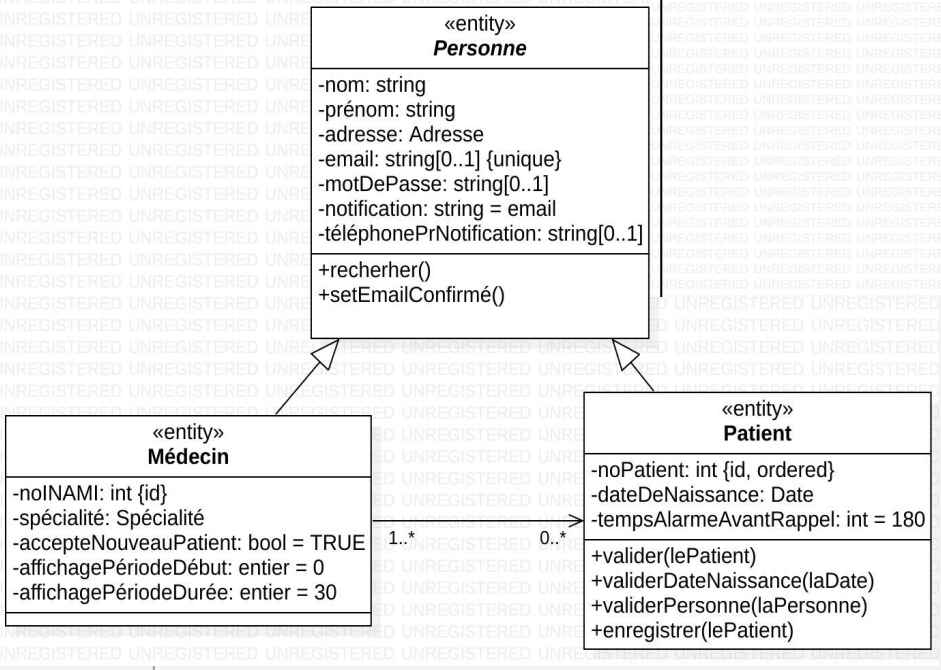
Démo MédiCab: MVP

Exercices

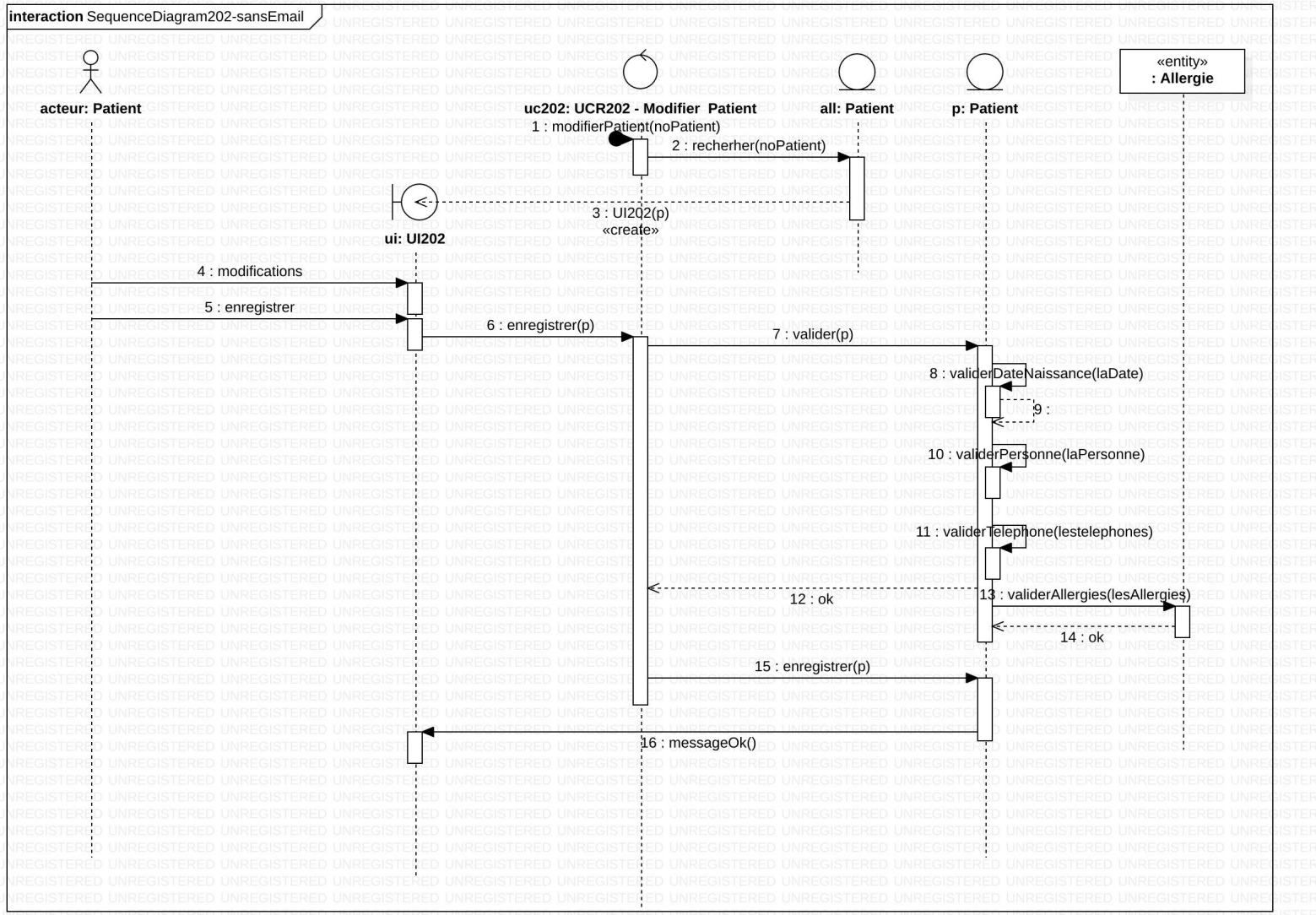
Exercice

Corrigez la cohérence et la syntaxe.





Exercice (suite)



UCR - GRASP

GRASP

- **G**eneral
- **R**esponsability
- **A**ssignment
- **S**oftware
- **P**atterns

C. Larman est l'inventeur des GRASP Patterns ('97)

- **P**atterns

Patron de conception

“Pour capitaliser un savoir précieux né du savoir-faire d’experts” Buschmann '96

Vocabulaire commun entre les analystes et les programmeurs

- **R**esponsability

Modèle d’affectation des responsabilités

Responsabilité = un contrat ou une obligation d’un objet en ce qui concerne son comportement.

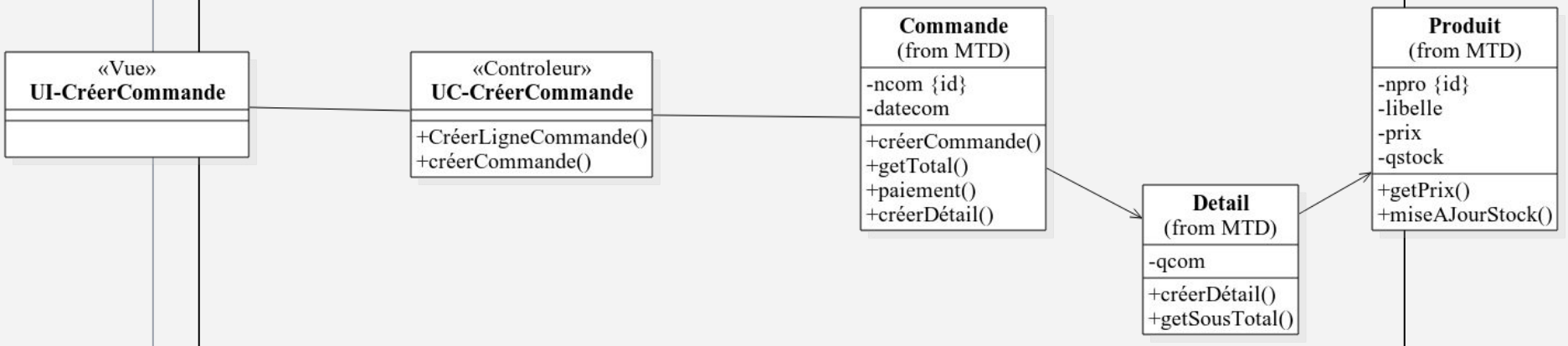
- **R**esponsability

Un objet a la responsabilité de

- Savoir (connaître):
  - les données privées encapsulées
  - les objets connexes
  - les éléments qu'il peut dériver ou calculer
- Faire

- **R**esponsability
  Savoir

exemple la classe Commande est responsable de connaitre sa date, ses lignes de détail



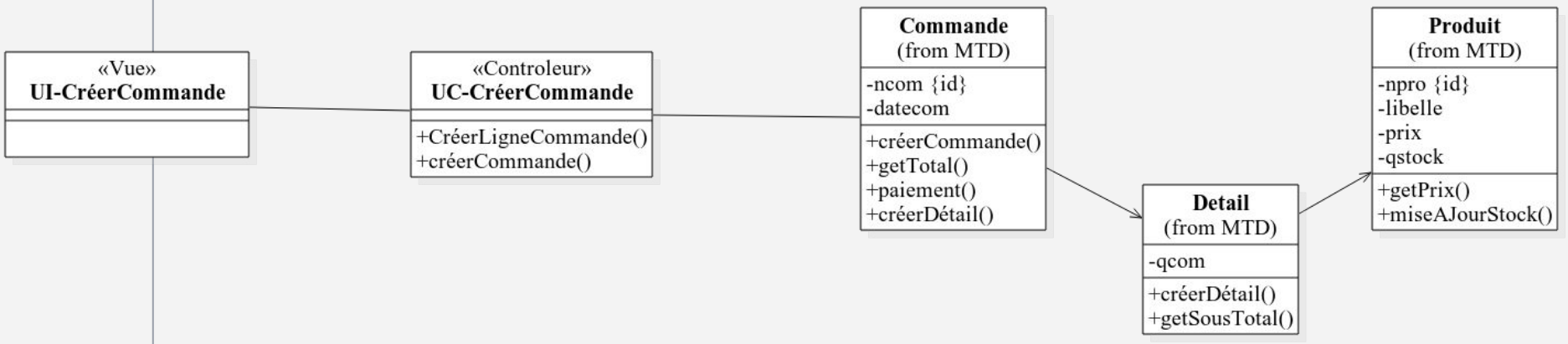
- **R**esponsability

Un objet a la responsabilité de

- Faire :
  - quelque chose lui-même
  - déclencher une action d'un autre objet
  - contrôler et coordonner les activités d'autres objets

- **R**esponsability
  Faire

exemple la classe Commande est responsable de calculer son montant total (en collaboration avec la classe détail)



GRASP

- Règles de bonne pratique en OO
  - Contrôleur
  - Haute cohésion
  - Couplage faible
  - Expert
  - Créateur
  - Polymorphisme
  - Fabrique
  - ...

Contrôleur

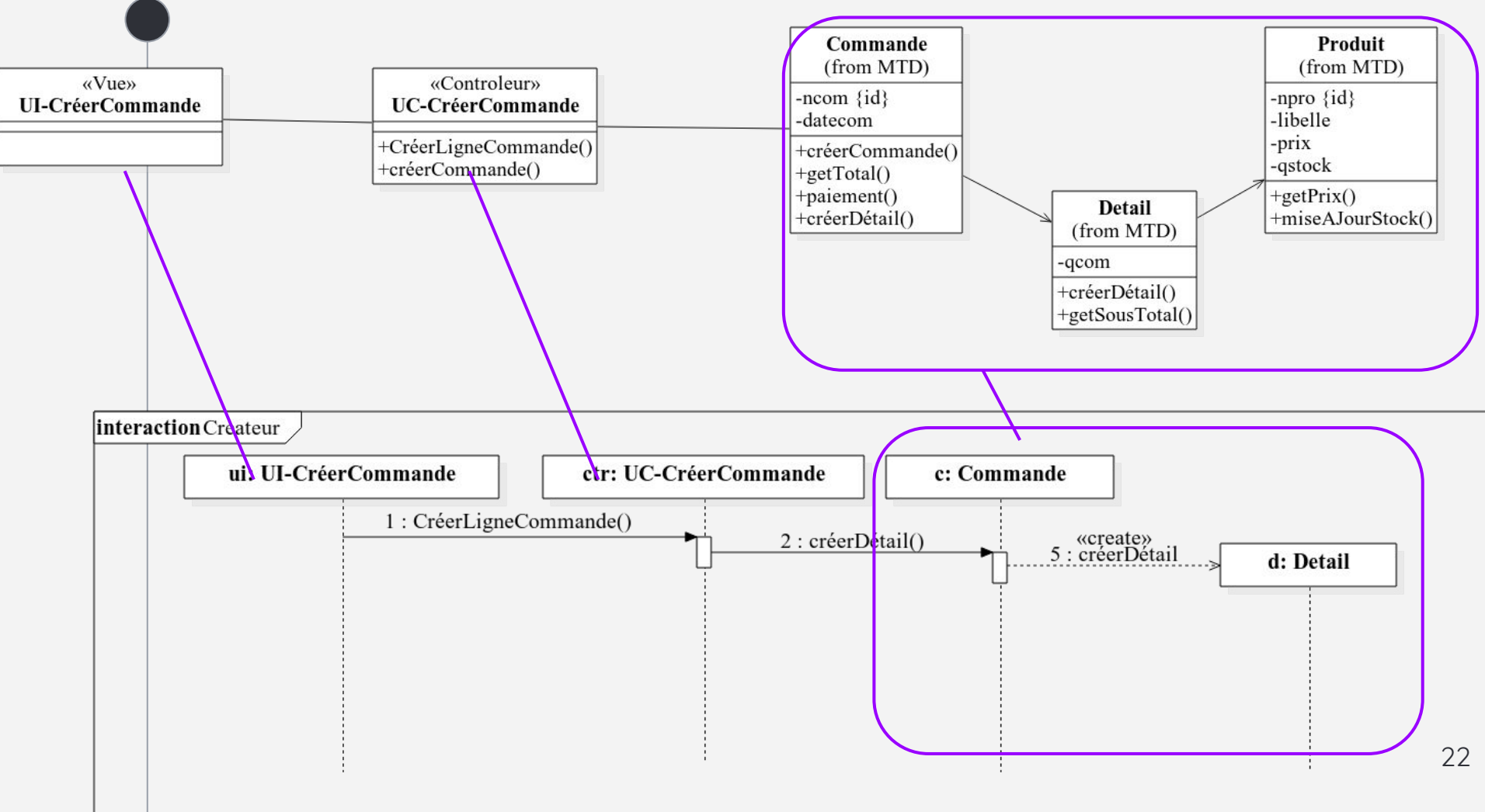
- Question: Quel est le premier objet qui reçoit et coordonne une opération système (après le GUI) ?

  Solution: Soit un objet représentant tout le système, soit un objet représentant un use case

Principes:

- un contrôleur est un objet qui ne fait rien
  - reçoit les événements système
  - délègue aux objets dont la responsabilité est de les traiter
- il se limite aux tâches de contrôle et de coordination
  - vérification de la séquence des événements système
  - appel des méthodes ad hoc des autres objets
- il n'est donc pas modélisé en tant qu'objet du domaine (Fabrication pure)

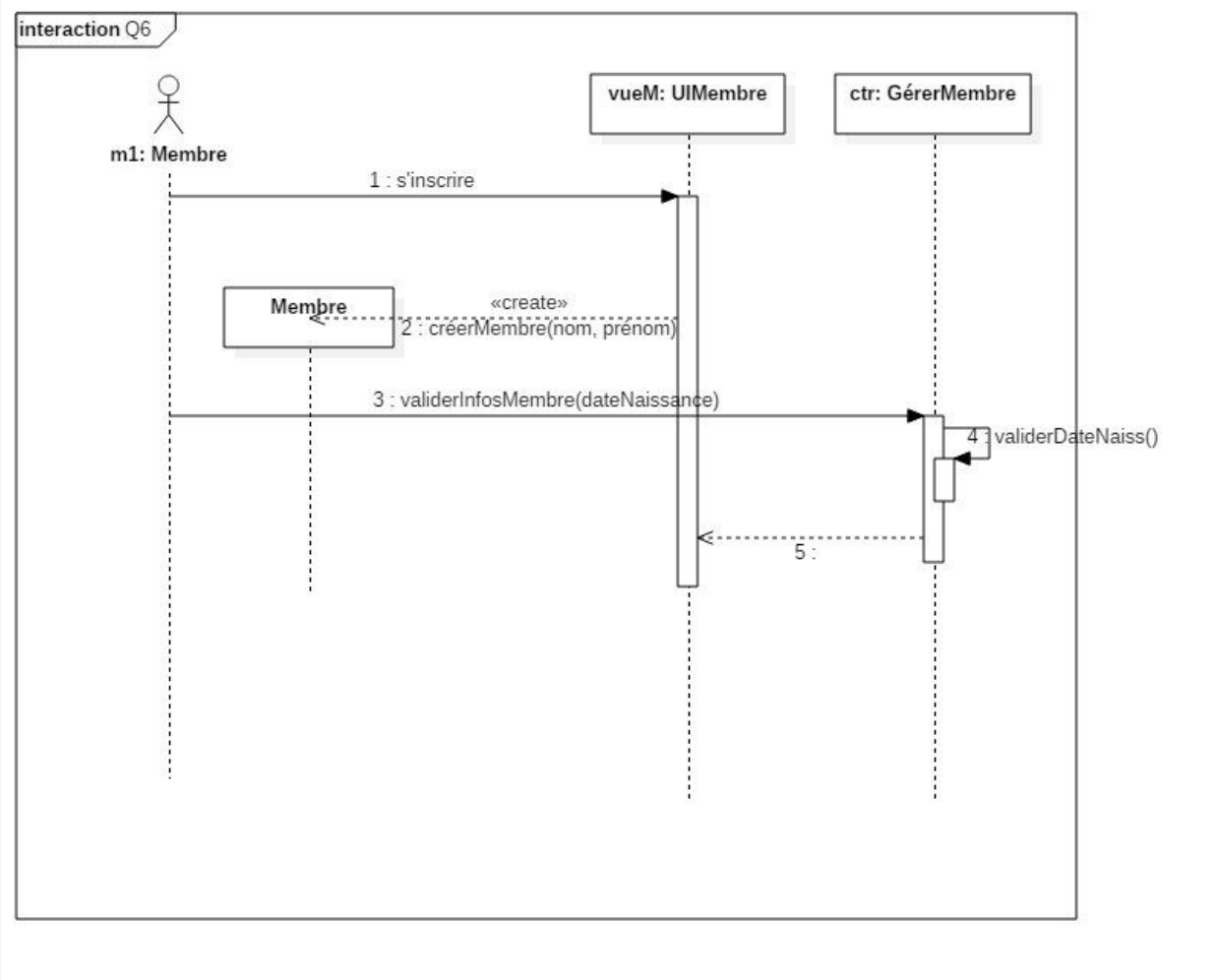
Modèle-Vue-Présentation



Exercices: MVP

Exercice 1

Donnez les 4 éléments qui posent problème. Enoncer à chaque fois la règle transgressée.

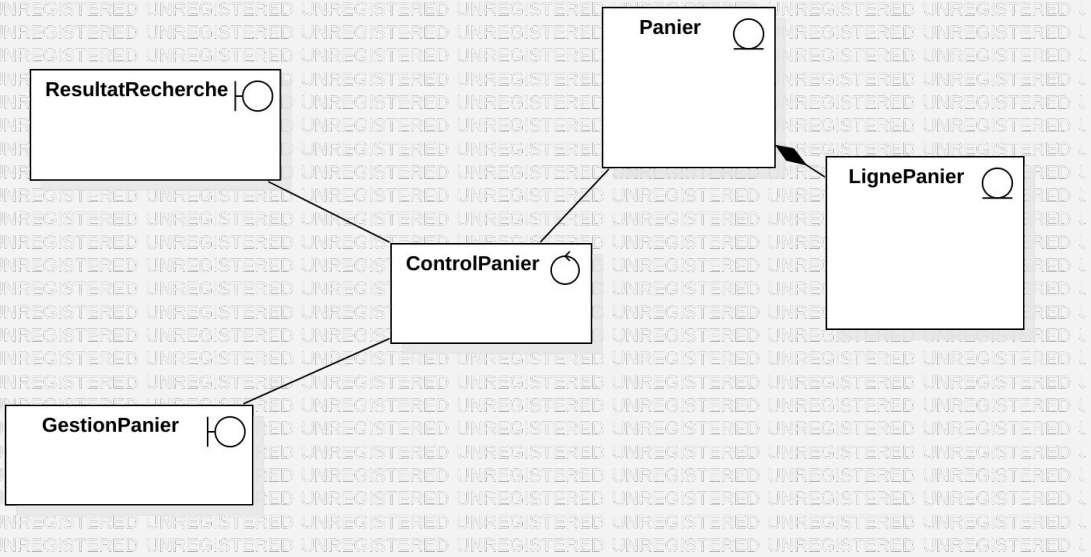


Exercice 2

Elaborer le diagramme de séquence du scénario suivant:

1. L'internaute demande de mettre dans un nouveau panier un article obtenu suite à une recherche
2. Un panier est créé pour cet internaute
3. L'article sélectionné est ajouté au panier
4. Le panier est affiché

Vous avez les classes suivantes:



Haute Cohésion

Question: Comment concevoir des objets clairs, gérables et bien concentrés sur leur tâche ?

Solution: Maximiser la cohésion.

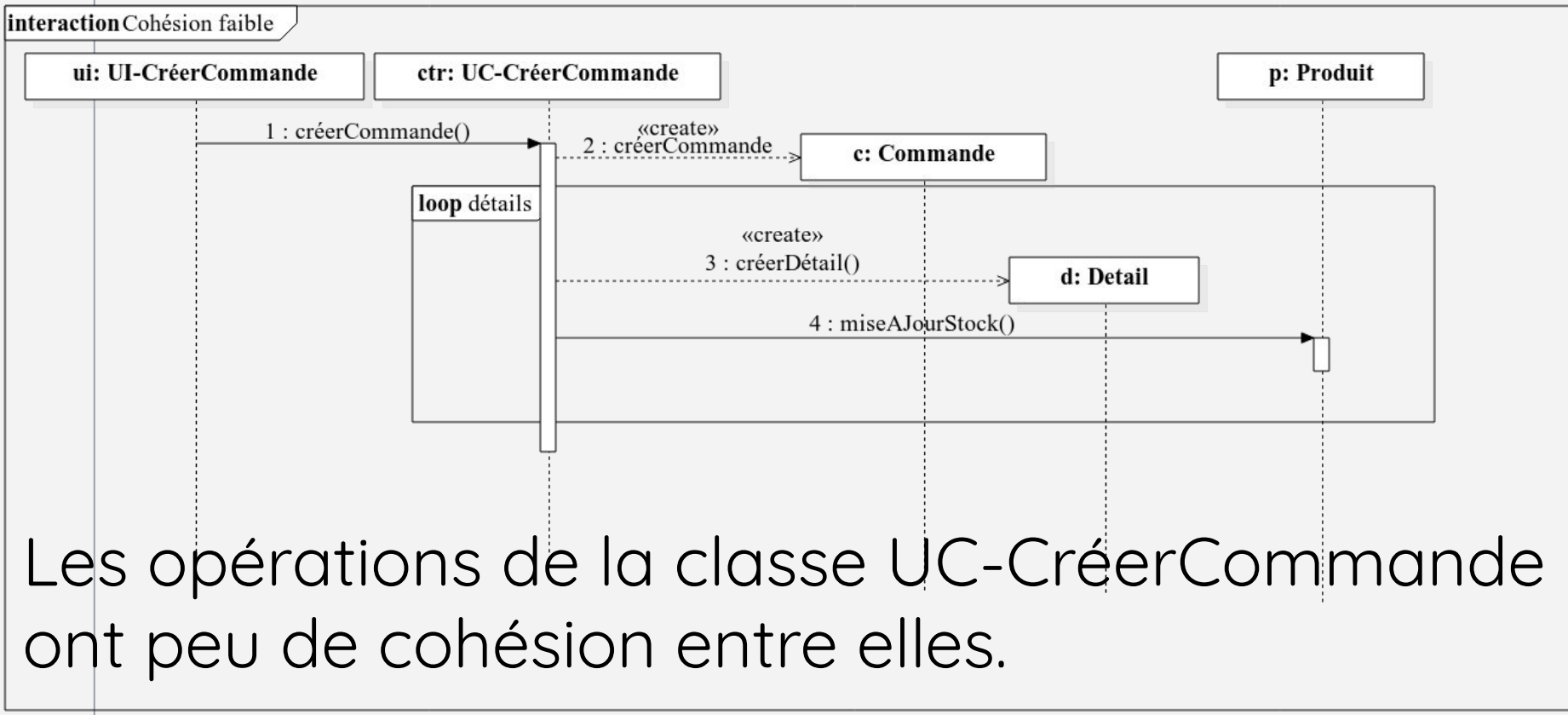
Définition: la *cohésion* exprime combien les opérations de la classe sont liées entre elles

Haute Cohésion

- Une classe qui est fortement cohésive
  - a des responsabilités étroitement liées les unes aux autres
  - n'effectue pas un travail gigantesque

Un test: décrire une classe avec une seule phrase

Exemple avec faible cohésion



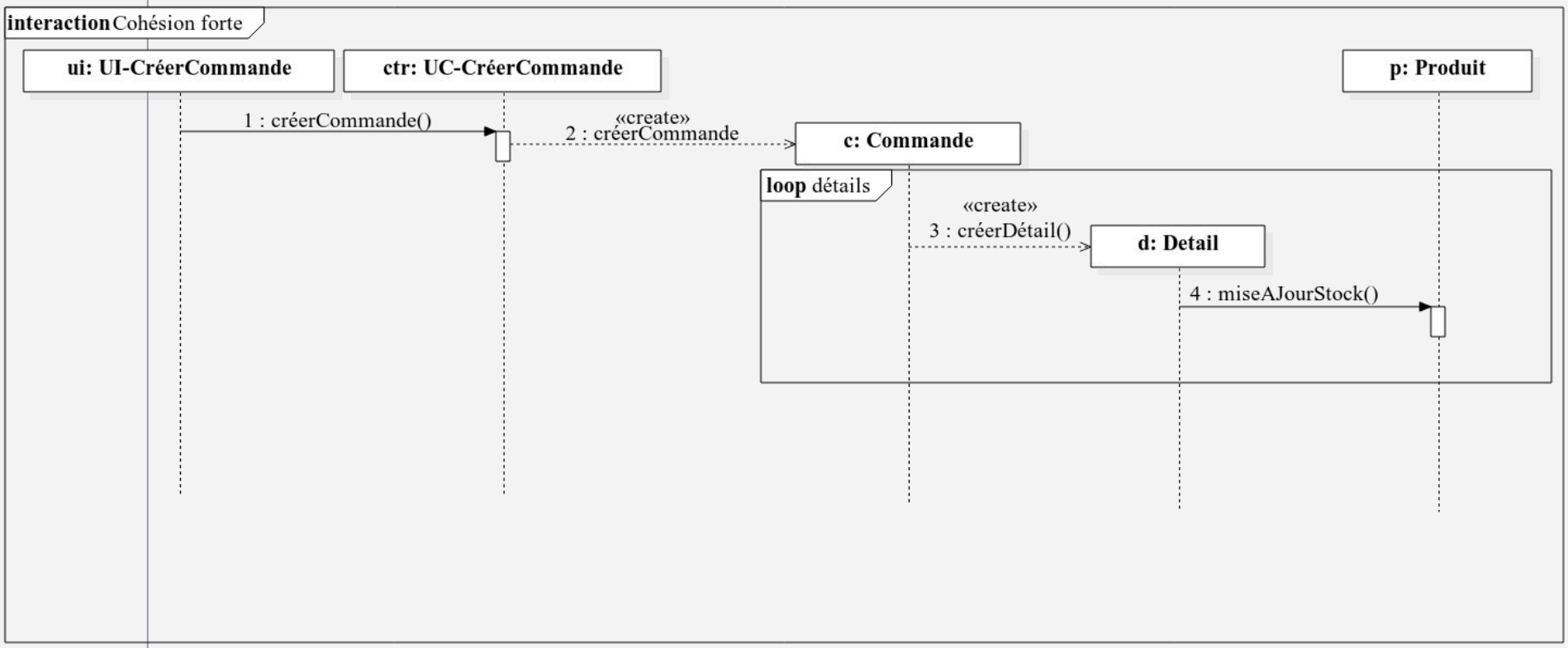
Les opérations de la classe UC-CréerCommande ont peu de cohésion entre elles.

Faible cohésion

Problème:

Si on continue de rendre l'objet ctr responsable de toutes les actions du UC, il va crouler sous les tâches et perdre sa cohésion.

Exemple avec haute cohésion



Couplage Faible

Question: Comment réduire l'impact du changement dans le code ?

Solution: Minimiser le couplage

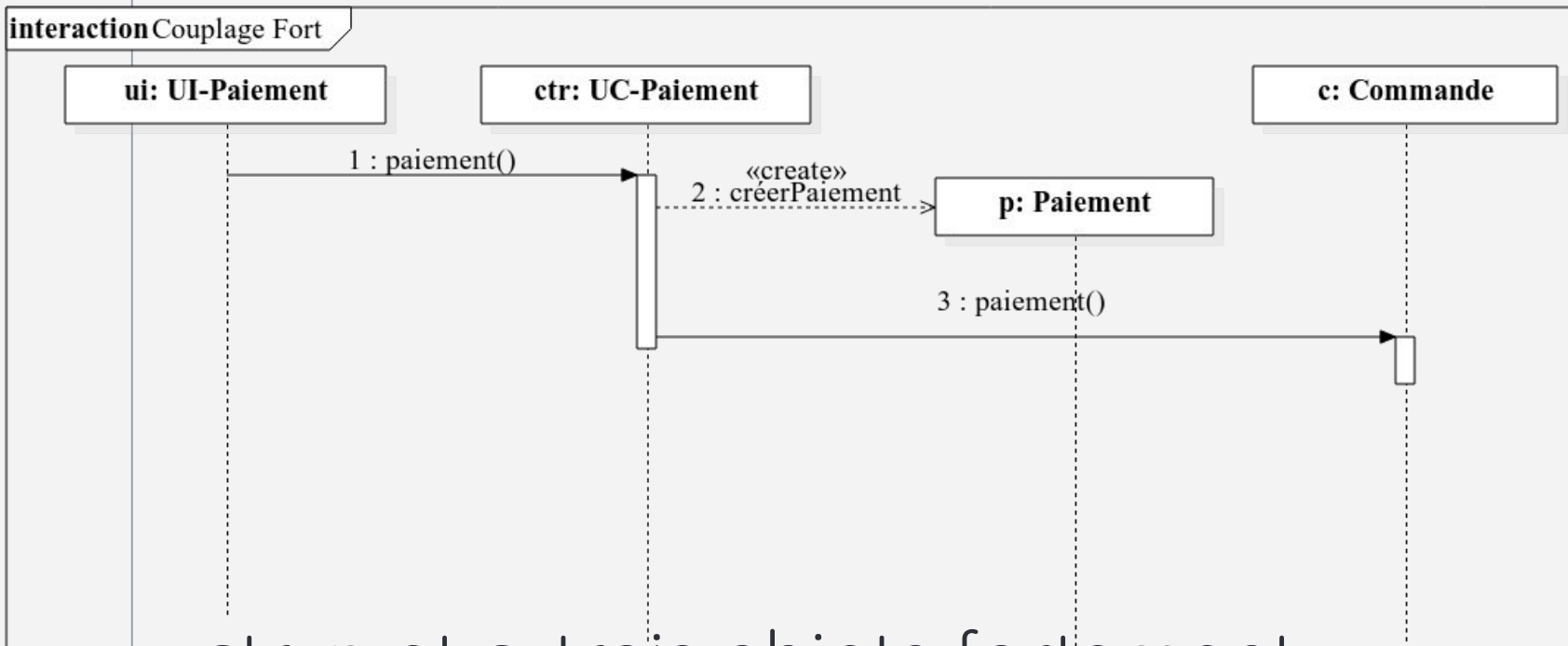
Définition: la *couplage* exprime combien les objets sont dépendants les uns des autres

Couplage Faible

Problèmes du couplage fort:

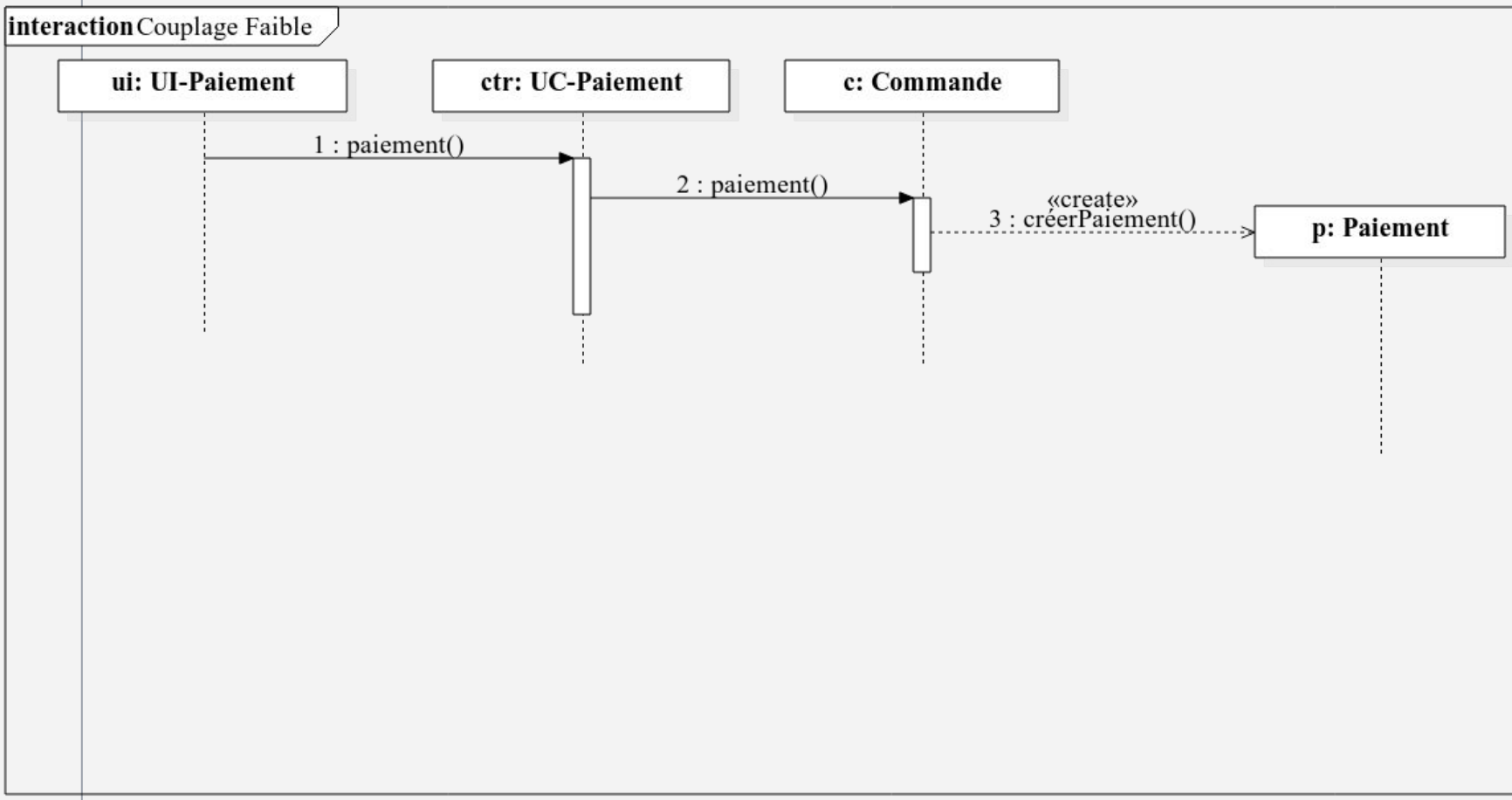
- Un changement dans une classe force à changer toutes ou la plupart des classes liées
- Les classes prises isolément sont difficiles à comprendre
- Réutilisation difficile : l'emploi d'une classe nécessite celui des classes dont elle dépend

Exemple avec haut couplage



ctr, p et c, trois objets fortement dépendants

Exemple avec couplage faible



le Yin et le Yang

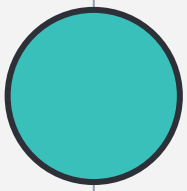
Les deux principes sont interdépendants

Si une classe a beaucoup de responsabilités fort différentes (faible cohésion), elle aura sûrement beaucoup de dépendances (fort couplage).

Haute Cohésion et Couplage Faible

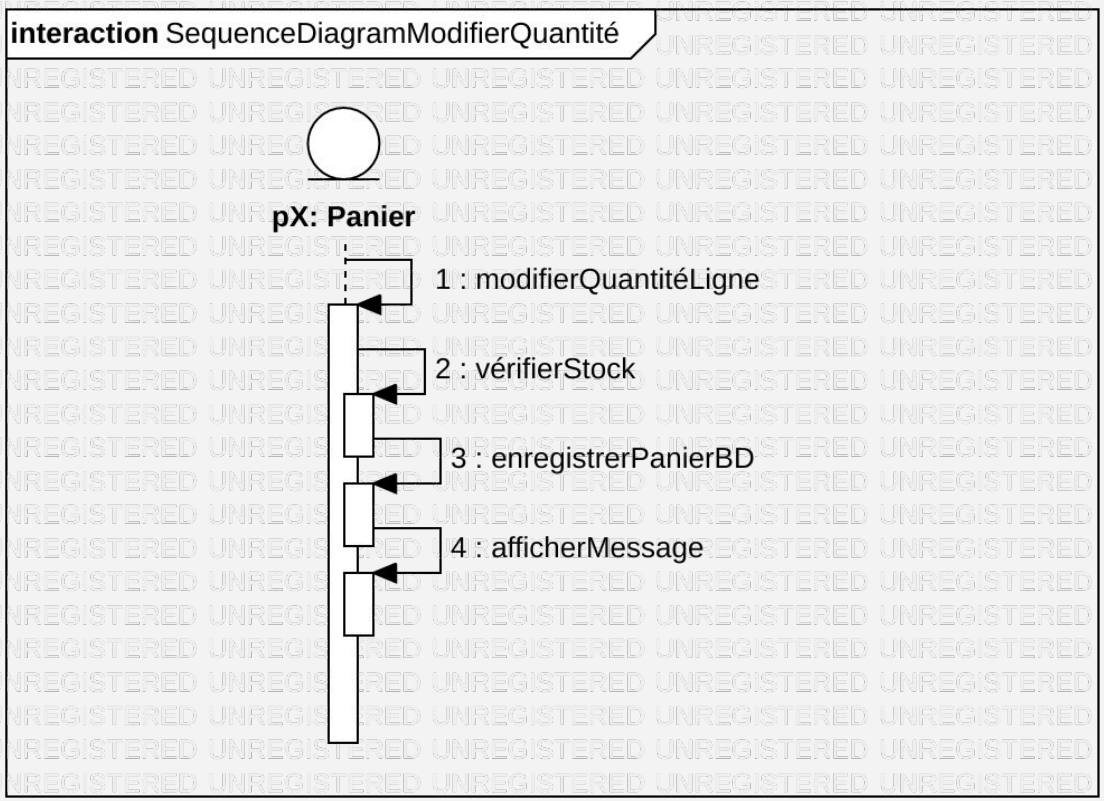
= modularité

La modularité est la propriété d'un système qui a été décomposé en un ensemble de modules cohésifs et faiblement couplés Booch '94



Exercice

Exercice: haute cohésion - Couplage faible



Appliquez les principes de haute cohésion et de couplage faible.

Expert

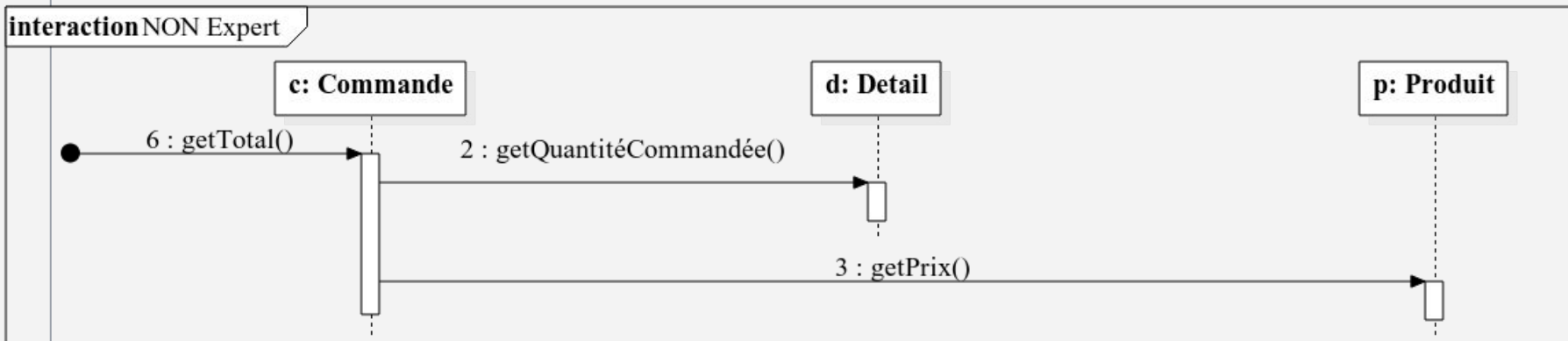
Question: Par quel principe assigner les **responsabilités** aux classes ?

Solution: Choisir la classe qui possède l'information nécessaire pour réaliser la tâche

- Mettre les responsabilités avec les données
- Qui sait, fait
- Faire soi-même

Expert : exemple

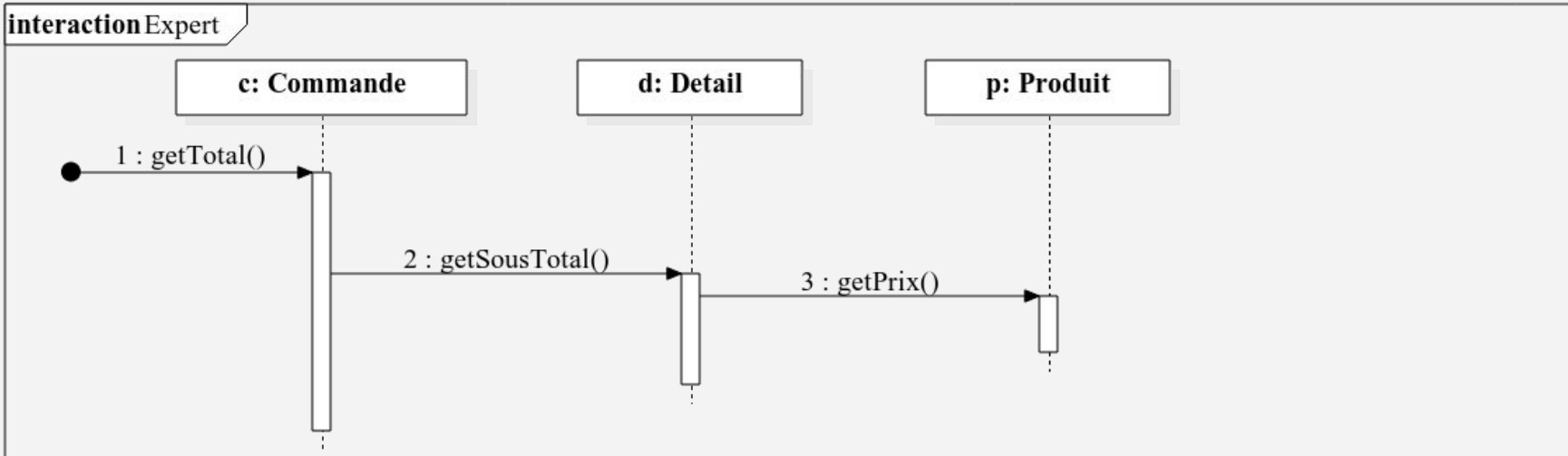
Calculer le total d'une commande



Commande calcule le sous-total et le total ?

Expert : exemple

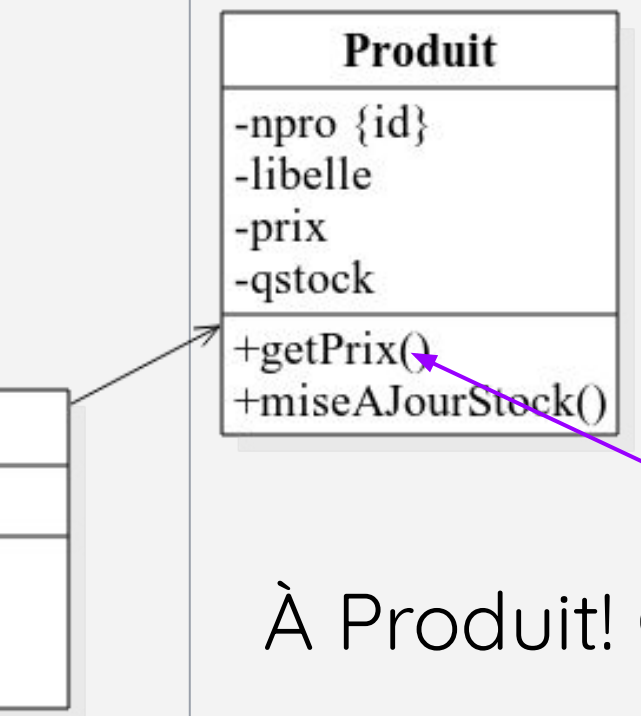
- Calcul du sous-total: à qui la responsabilité ?



À Détail! Car elle seule connaît la quantité commandée et le produit

Expert : exemple

- Fournir le prix d'un Produit: à qui la responsabilité ?

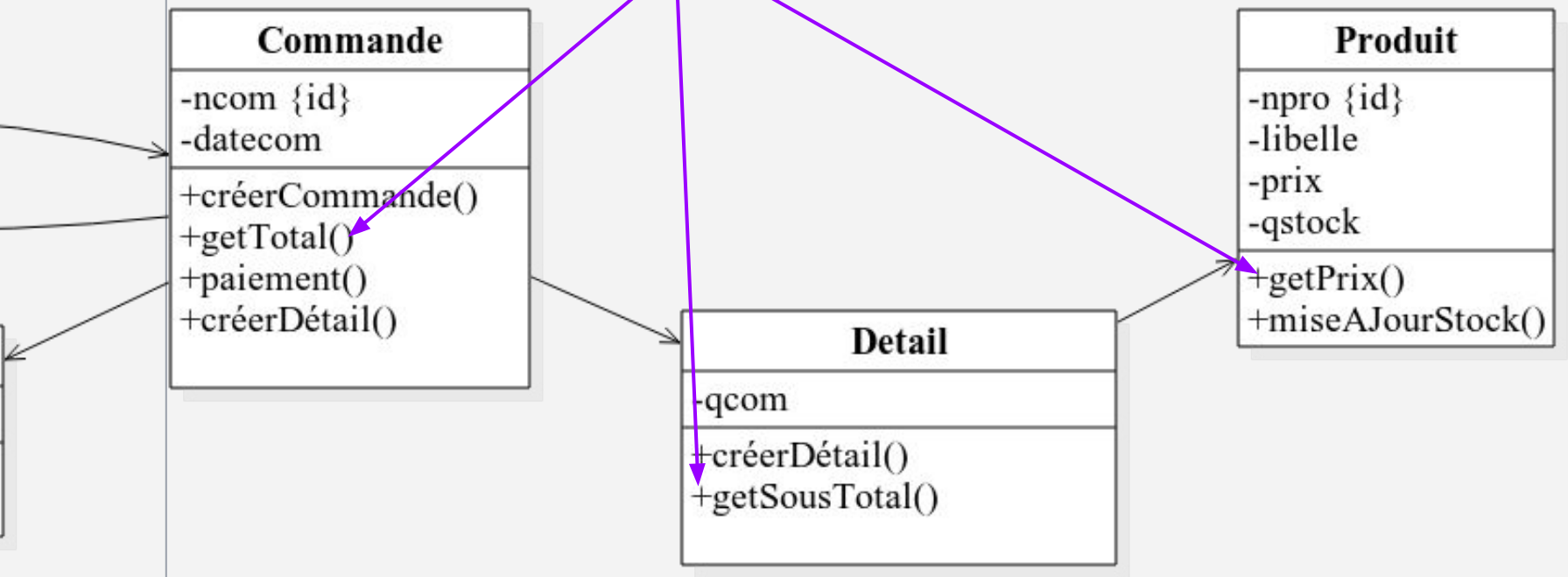


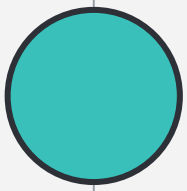
À Produit! Car il contient le prix

Expert : exemple

- Calculer le total d'une commande :

  Solution finale

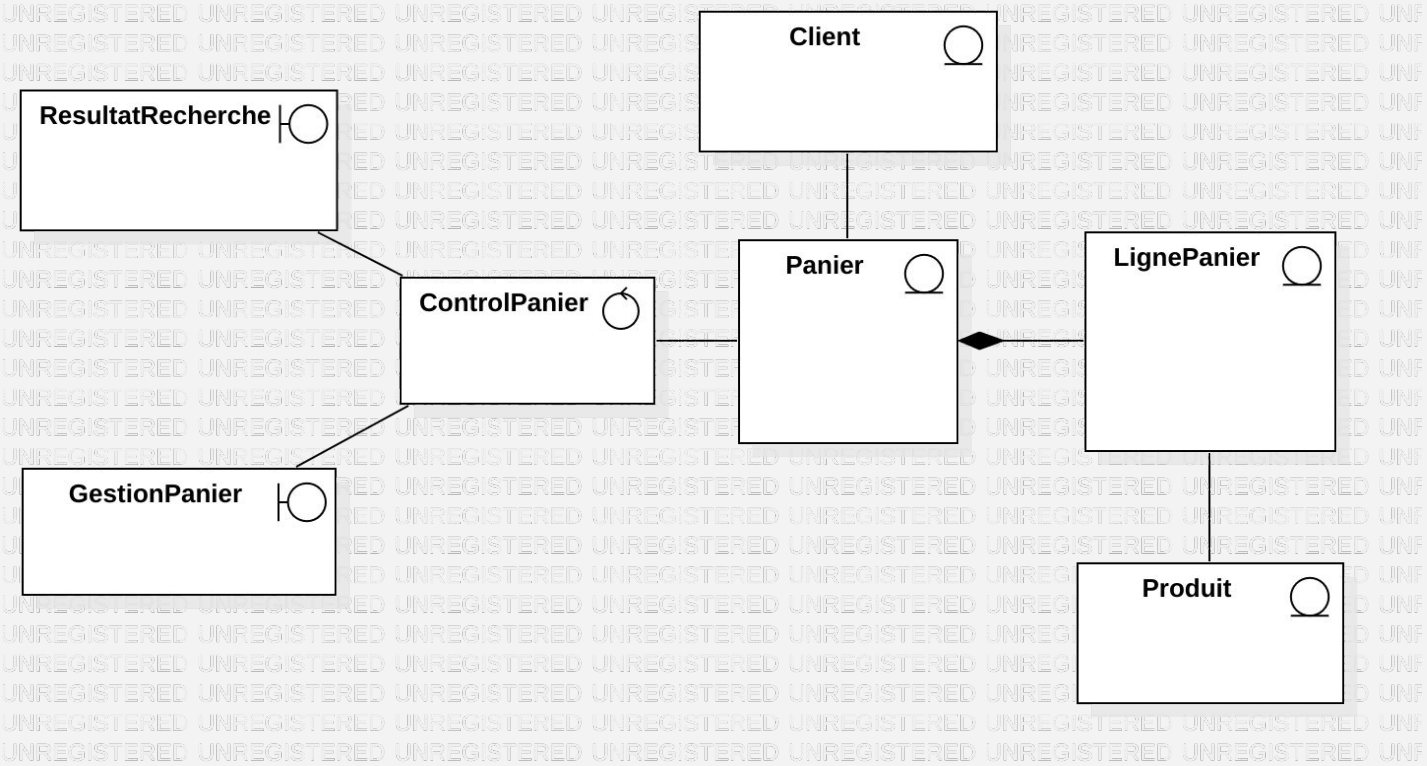




Exercice

Exercice: Expert

- Où placer la méthode nombreProduit() qui donne le nombre de produit dans le panier?



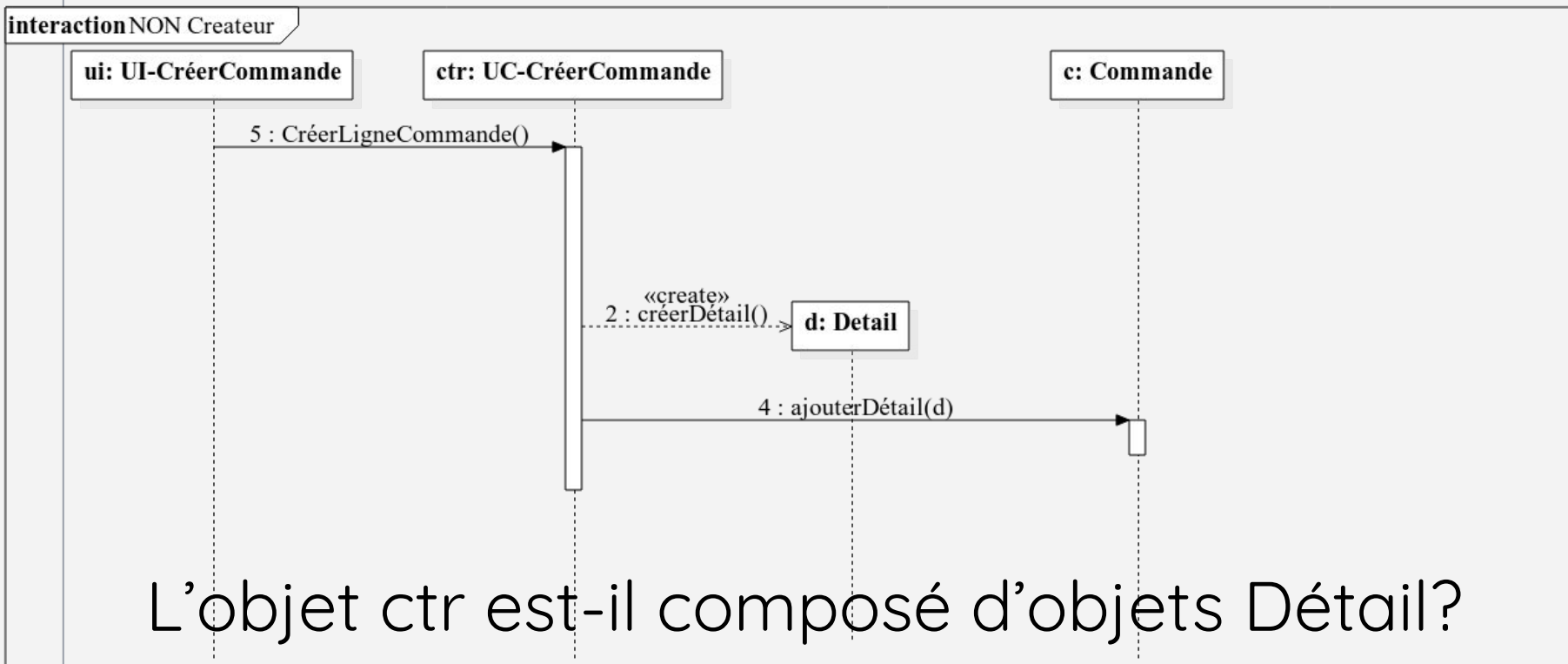
Créateur

Question: Qui est responsable de créer les objets d'une classe A ?

Solution: Idéalement une classe qui est composée de A, contient des A, utilise des A, et/ou possède les infos d'initialisation pour les objets de A

Créateur : exemple

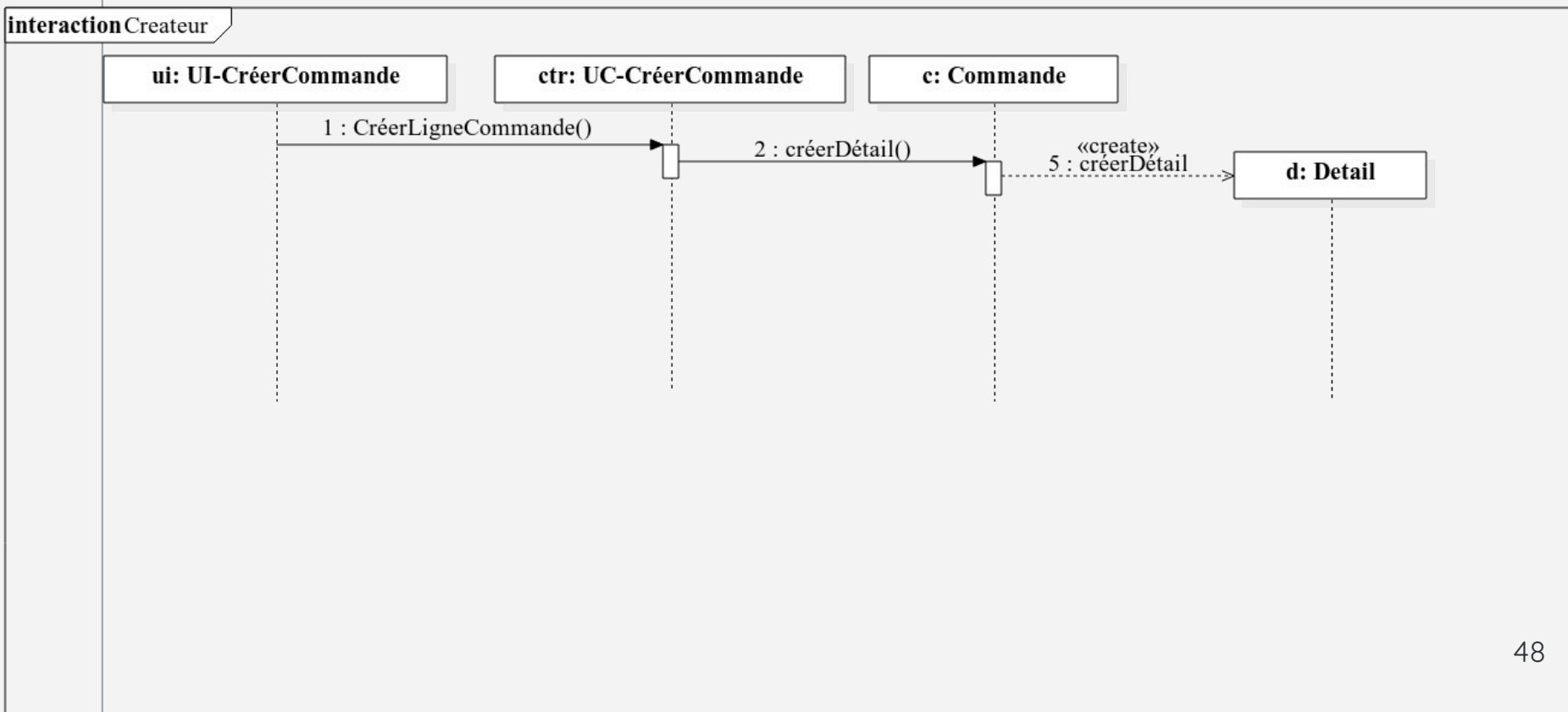
- Qui crée les objets Détail ?

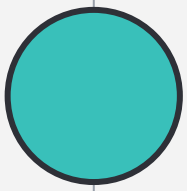


L'objet ctr est-il composé d'objets Détail?

Créateur : exemple

Commande crée les objets Détail

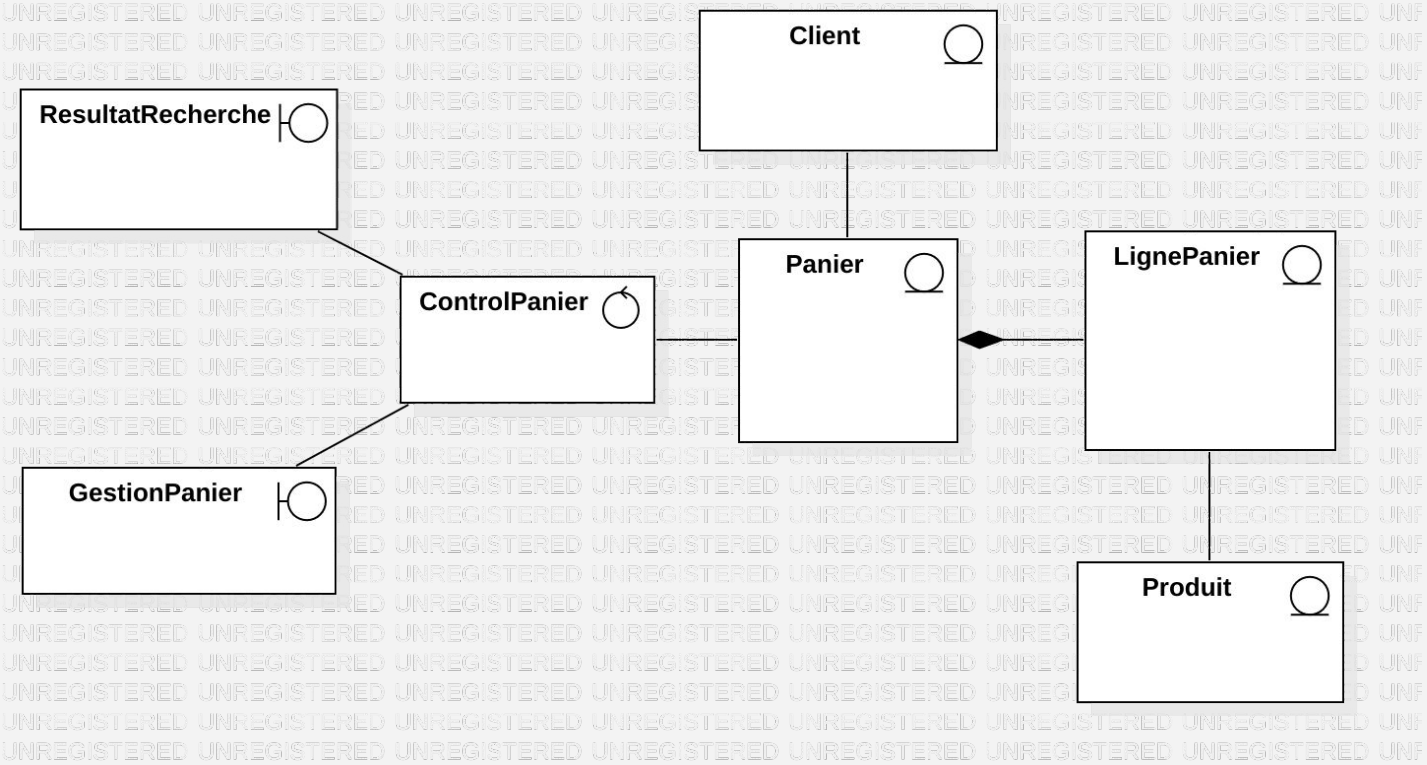




Exercice

Exercice: Créateur

- Quelle classe est responsable de la création d'un objet Panier, Client, LignePanier, Produit, GestionPanier?



Polymorphisme

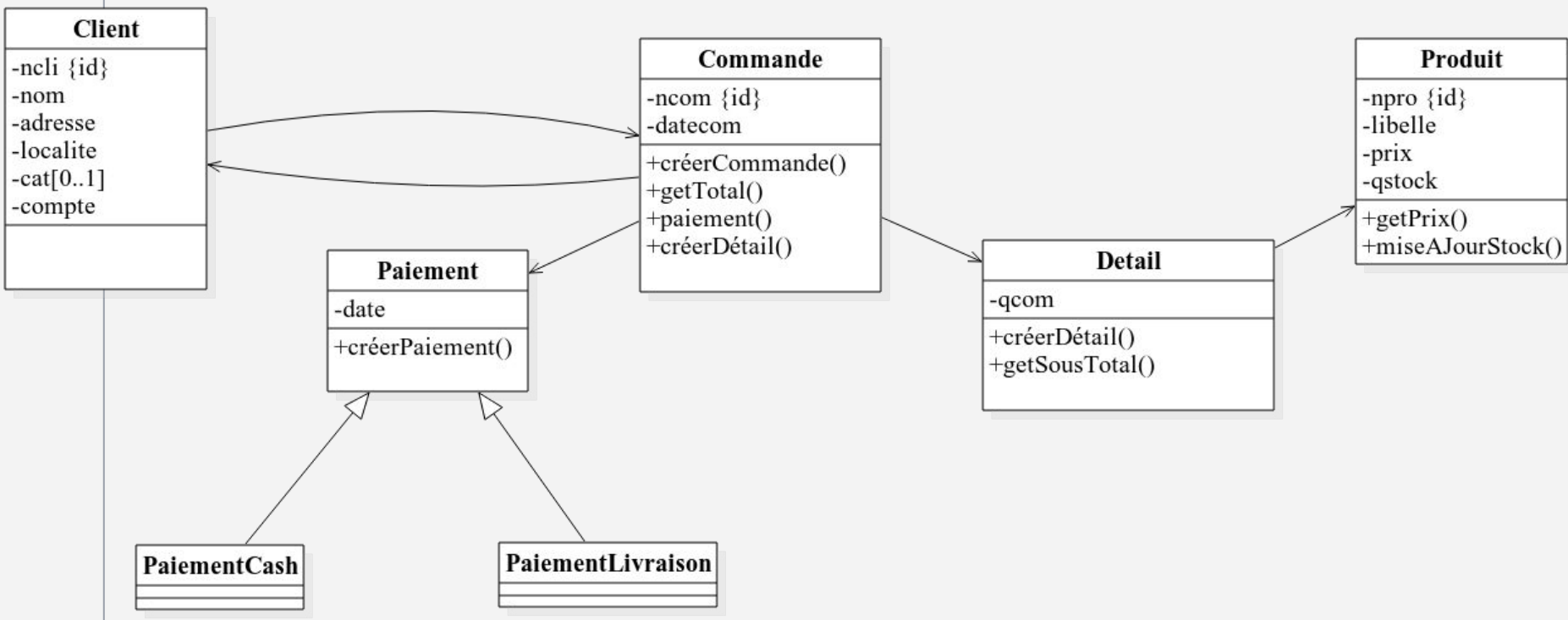
- Question: Comment gérer les traitements alternatifs basés sur le type?

  Solution: Assigner les responsabilités aux classes pour lesquelles le traitement varie

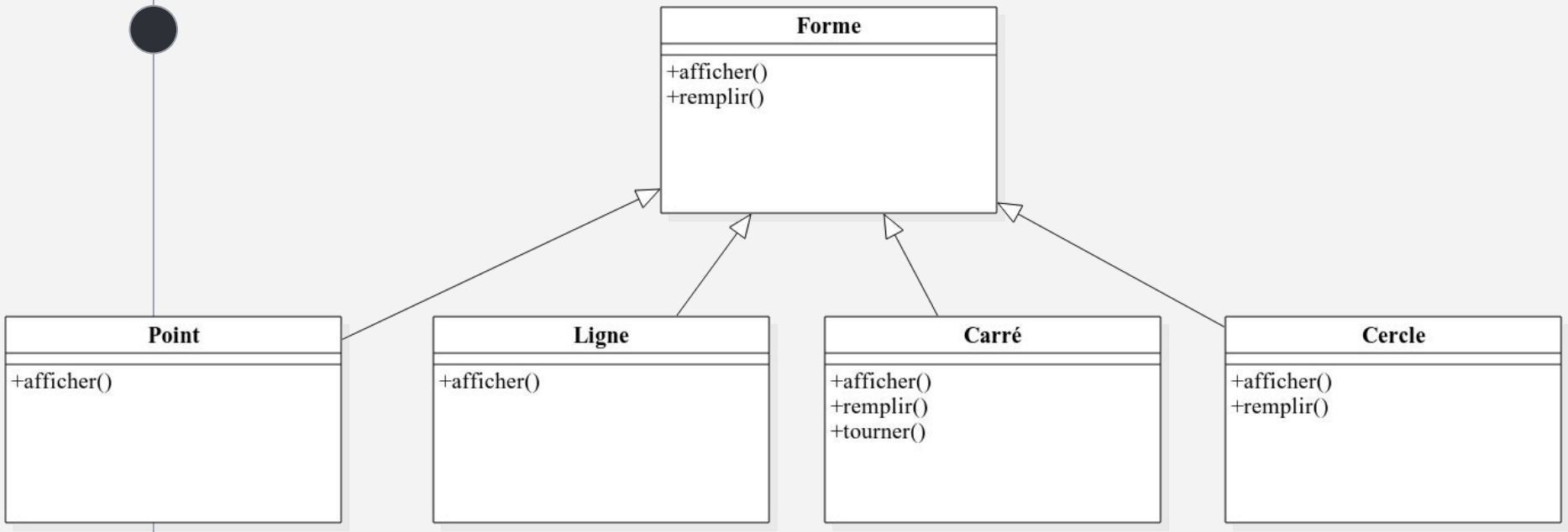
  Corollaire: Ne pas faire de “if” sur le type de l’objet

- Avantages :
  - Évolutivité
    - Points d'extension requis par les nouvelles variantes faciles à ajouter (nouvelle sous-classe)
  - Stabilité du client
    - Introduire de nouvelles implémentations n'affecte pas les clients

Polymorphisme

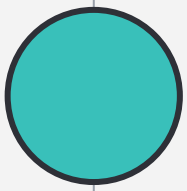


Polymorphisme



Pour aller plus loin

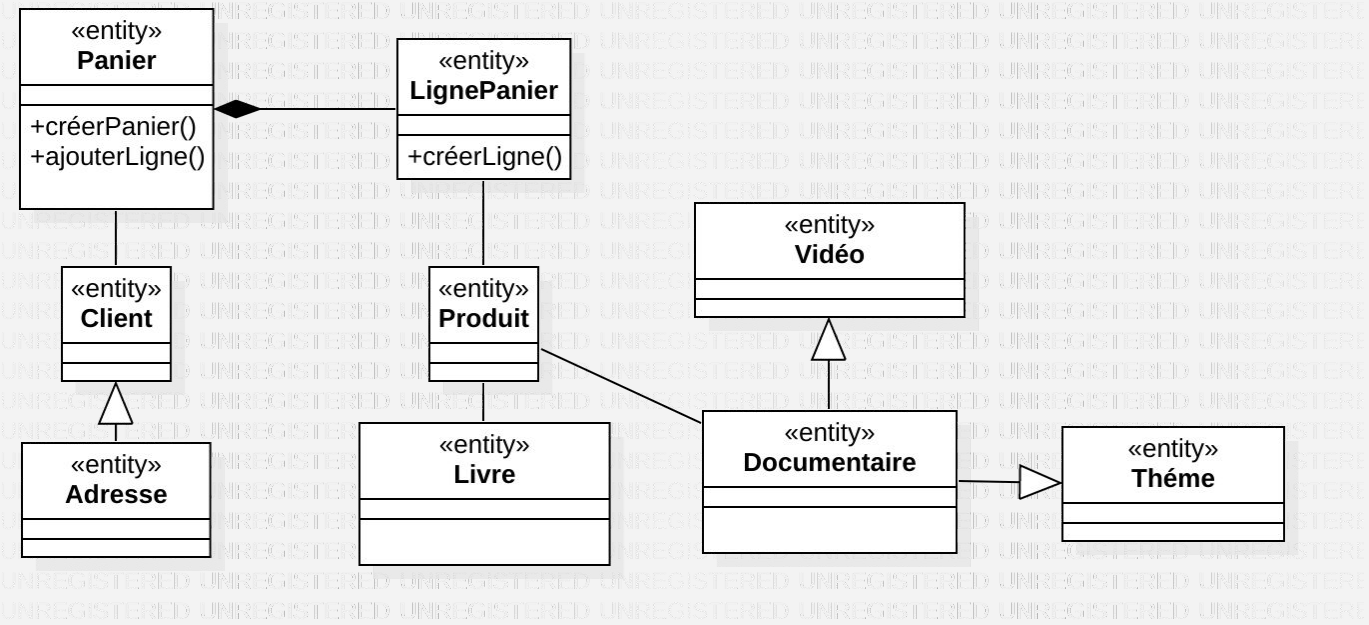
Un match de football polymorphique H Bersini (p 256)



Exercice

Exercice: Polymorphisme

- Voici un diagramme de classes à corriger.



Fabrique

- Question: A qui assigner les responsabilités quand toute solution semble enfreindre la Haute Cohésion ou le Couplage Faible ?

  Solution: Créer une classe artificielle (pas directement liée à un concept-métier), et contenant des responsabilités bien cohésives

Problème: Back-up des commandes

Le pattern Expert donne cette responsabilité à la classe Commande, MAIS

- cette opération n'est pas liée au concept de commande donc entraîne une faible cohésion
- Couplage fort entre la classe Commande et la BD back-up
- Peu de réutilisation pour le back-up d'autres classes

Solution: Back-up des commandes

Une classe BackUp responsable du back-up:

- Cette classe est artificielle, elle ne correspond pas à un concept métier
- La classe Commande garde sa forte cohésion et son faible couplage
- La classe BackUp a une seule responsabilité
- La classe BackUp est réutilisable

ET encore : KISS - DRY - YAGNI

KISS: **Keep It Simple Stupid**

Plus le programme est grand, plus le code a tendance à être complexe. Il sera plus difficile à comprendre à maintenir, à debugger,...

Personne n’aime un code complexe.

Si vous avez un code qui se complexifie, coupez-le en petits morceaux plus simples.

Il est plus facile de faire un code complexe qu’un code simple.

ET encore : KISS - DRY - YAGNI

DRY: **Don’t Repeat Yourself**

Principe important pour écrire un code simple et facile à maintenir. Un morceau de code ne peut jamais être écrit deux fois.

Personne n’aime écrire, lire, modifier plusieurs fois le même code.

Si vous avez un code qui se répète, faites une méthode commune, un héritage, ...

ET encore : KISS - DRY - YAGNI

YAGNY: **You Aren’t Gonna Need It**

Certain aime développer un code qui pourrait servir dans le futur. Développer pour maintenant pas pour le futur.

Personne n’aime écrire un code qui ne servira pas.