

LOG210 ANALYSE ET CONCEPTION DE LOGICIELS: SÉANCE 07

SURVOL

- Travail en équipe
- Rappel méthodologie
- Rétroaction mini-test
- Retour Exercice RDCU
- GRASP Faible Couplage F16.12/A17.12
- GRASP Forte Cohésion F16.14/A17.14
- GRASP Polymorphisme F22.1/A25.1
- GRASP Fabrication Pure F22.2/A25.2
- GRASP Indirection F22.3/A25.3
- GRASP Protection des variations F22.4/A25.4
- Patron Transformer ID en objets (ndc 9.4)
- Patron Faire soi-même (ndc 9.8)



TRAVAIL EN ÉQUIPE

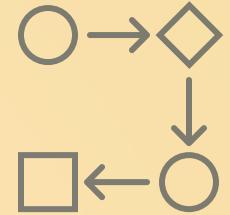
DÉVELOPPEMENT DE LOGICIELS

RENCONTRES EFFICACES

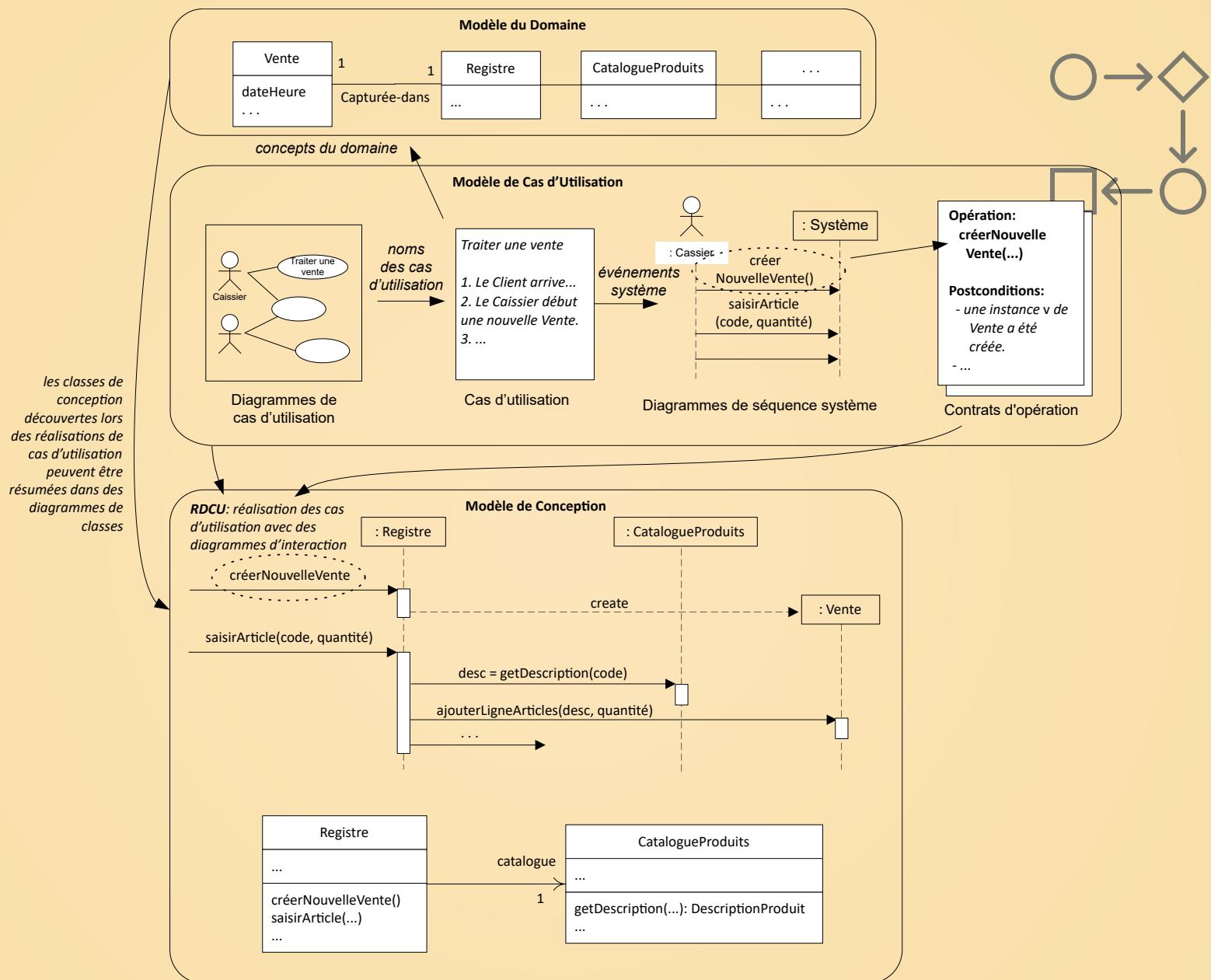


1. N'inviter que celles et ceux qui doivent absolument être présent.e.s
2. Préparer et distribuer l'ordre du jour *bien avant* que ça commence
3. Terminer tôt si les objectifs ont été atteints
4. Respecter l'ordre du jour
5. Planifier la rencontre autour des pauses habituelles (p.ex., à midi, à la fin de la journée)

[TeamGeek]



MÉTHODOLOGIE



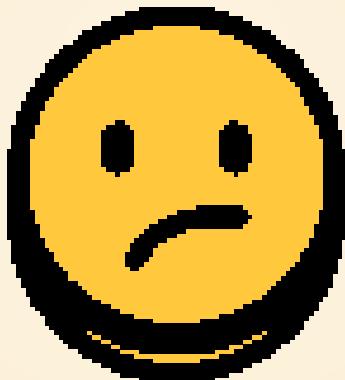


RÉTROACTION

MINI-TEST



QUESTIONS DIFFICILES



Selon les statistiques hier matin.



Socrative → ETSLOG210



Selon le principe MVC, les classes _____ sont plus stables que les classes _____ parce que _____ :

- A. du modèle / de la vue / le code des classes du modèle ne change jamais.
- B. du modèle / de la vue / faire des vues correctement est difficile et leur code change souvent.
- C. de la vue / du modèle / le code des classes du modèle ne change jamais.
- D. de la vue / du modèle / faire des vues correctement est difficile et leur code change souvent.



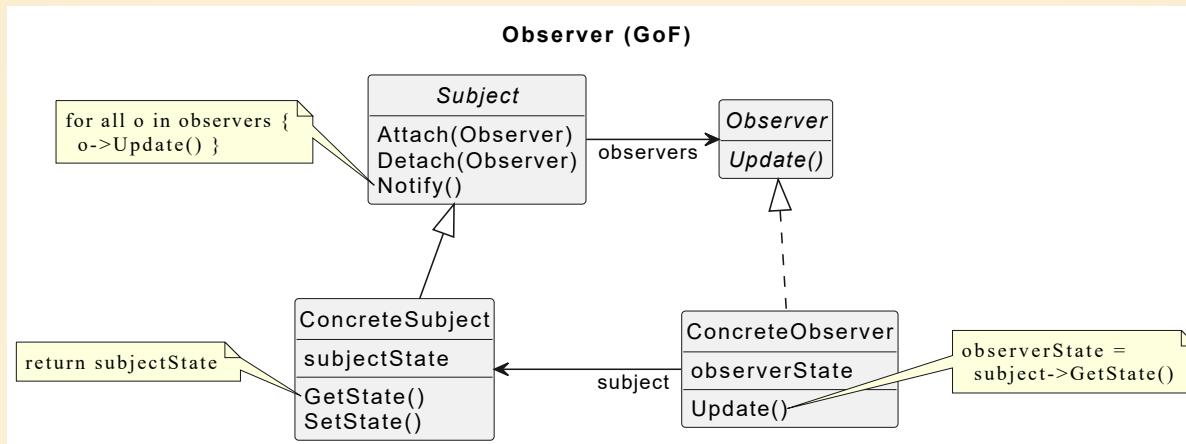
Socrative → ETSLOG210

Quel principe GRASP exploite le concept de masquage
de l'information?

- A. Polymorphisme
- B. Indirection
- C. Fabrication Pure
- D. Protection des variations



Socrative → ETSLOG210



Considérez le pattern Observateur où le principe GRASP **Protection des variations** s'y applique. Les classes _____ sont protégées des variations des classes _____ grâce à l'interface stable _____ :

- A. sujets / observateurs concrets / Subject
- B. observateurs concrets / sujets / Observer
- C. sujets / observateurs concrets / Observer
- D. observateurs concrets / sujets / Subject



Socrative → ETSLOG210



Dans le pattern Observateur, qu'est-ce qui empêche les sujets d'avoir un fort couplage aux observateurs concrets?

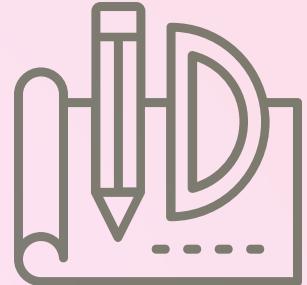
1. Le patron fait en sorte que les sujets ne voient pas du tout les observateurs.
Par exemple, les classes du modèle n'ont pas le droit de connaître les vues.
2. Les sujets ont un accès restreint aux observateurs. Les sujets voient les observateurs à travers l'interface Observer
3. Le patron fait en sorte que les sujets ne voient pas du tout les observateurs.
Par exemple, les classes du modèle n'ont pas le droit de connaître les vues.

Opération:
créerNouvelle
Vente(...)

Postconditions:
- une instance v de
Vente a été
créeée.
- ...

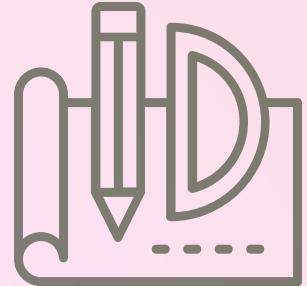
RETOUR EXERCICE RDCU

Google Classrooms



RDCU

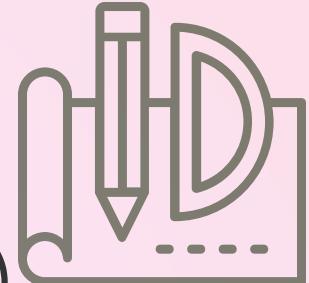
RÉALISATION D'UN CAS D'UTILISATION



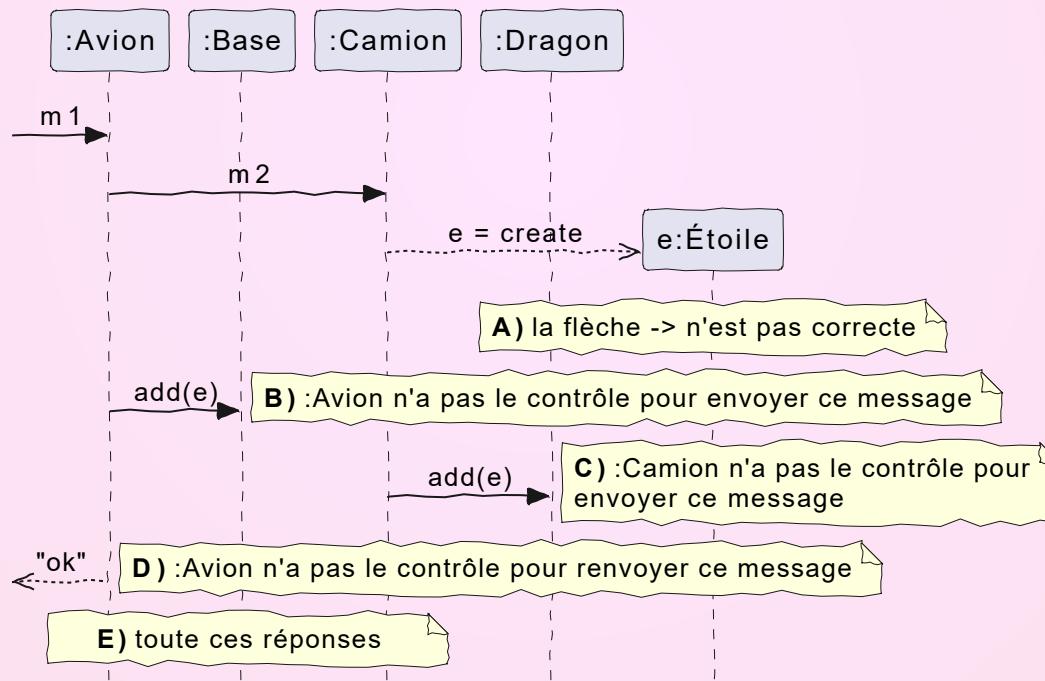
RAPPEL: DIAGRAMMES DE SÉQUENCE



Socrative → ETSLOG210

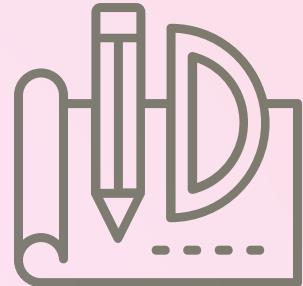


Quel(s) est(sont) le(s) problème(s)
de **flot de contrôle** avec le diagramme suivant?

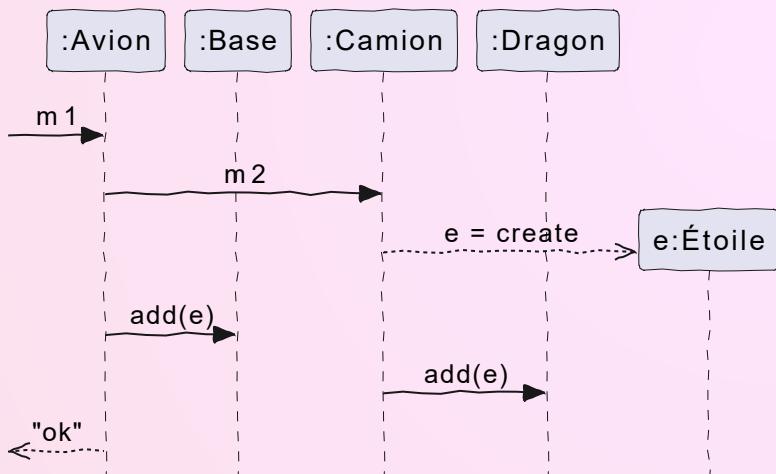




Socrative → ETSLOG210



Selon le diagramme suivant, qui voit *e* ?

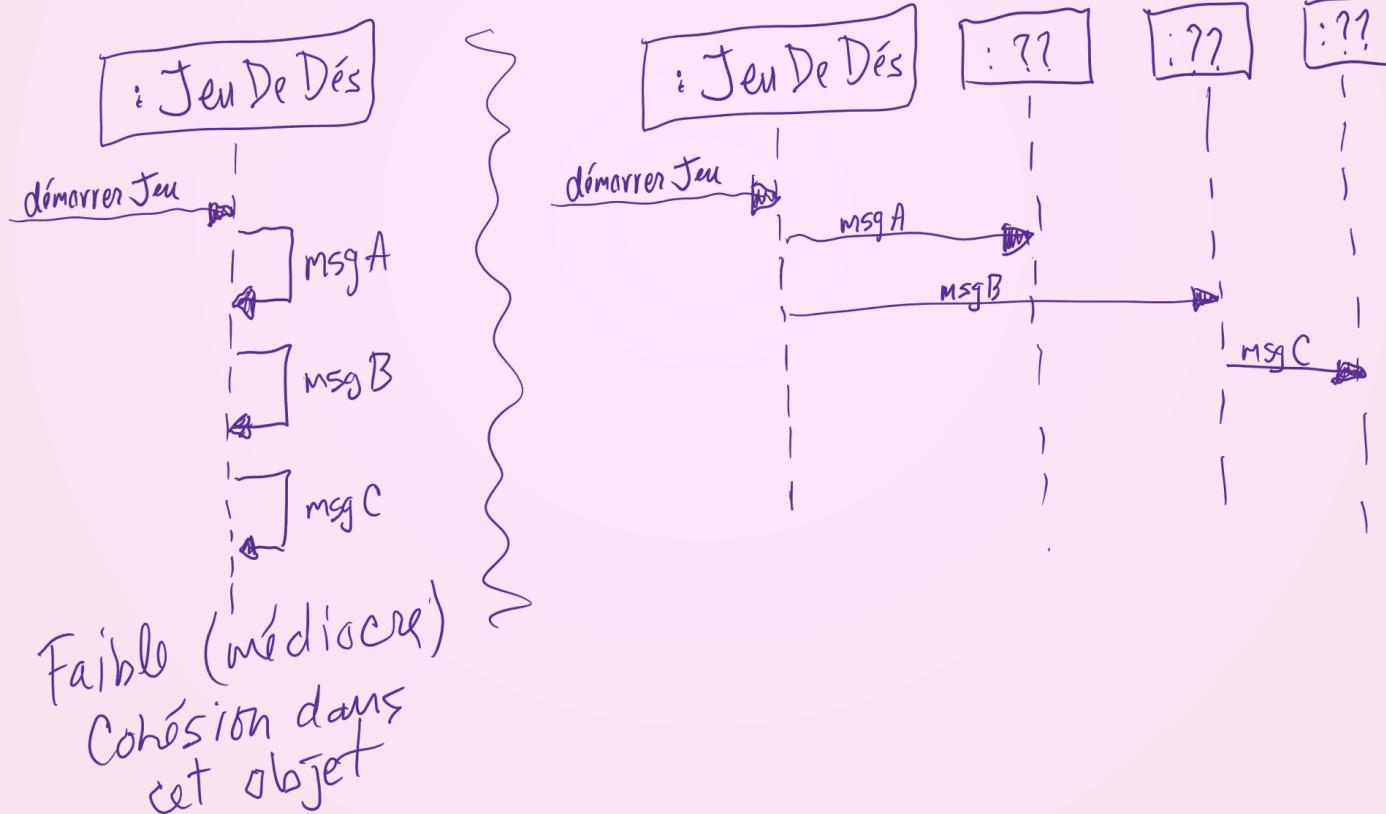


- A. :Avion
- B. :Base
- C. :Camion
- D. :Dragon
- E. (tous)

RDCU



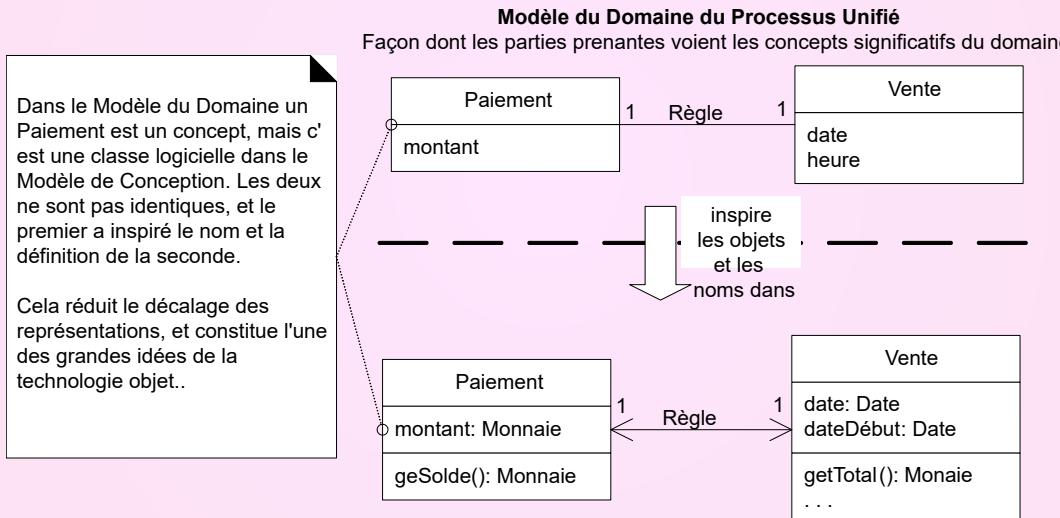
Prendre les bonnes décisions pour une solution facile à comprendre et modulaire...



DÉCALAGE DES REPRÉSENTATIONS

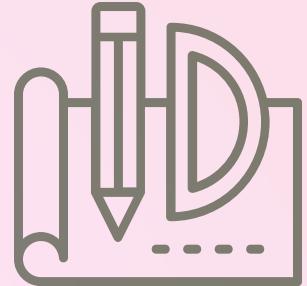


Facile? Les classes logicielles devraient ressembler à des classes conceptuelles (concepts du monde réel).



Qui fait quoi? Qui a quelle responsabilité?

RDCU



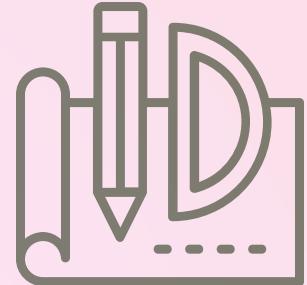
Approche: conception orientée-responsabilités

GRASP

General Responsibility Assignment Software Patterns

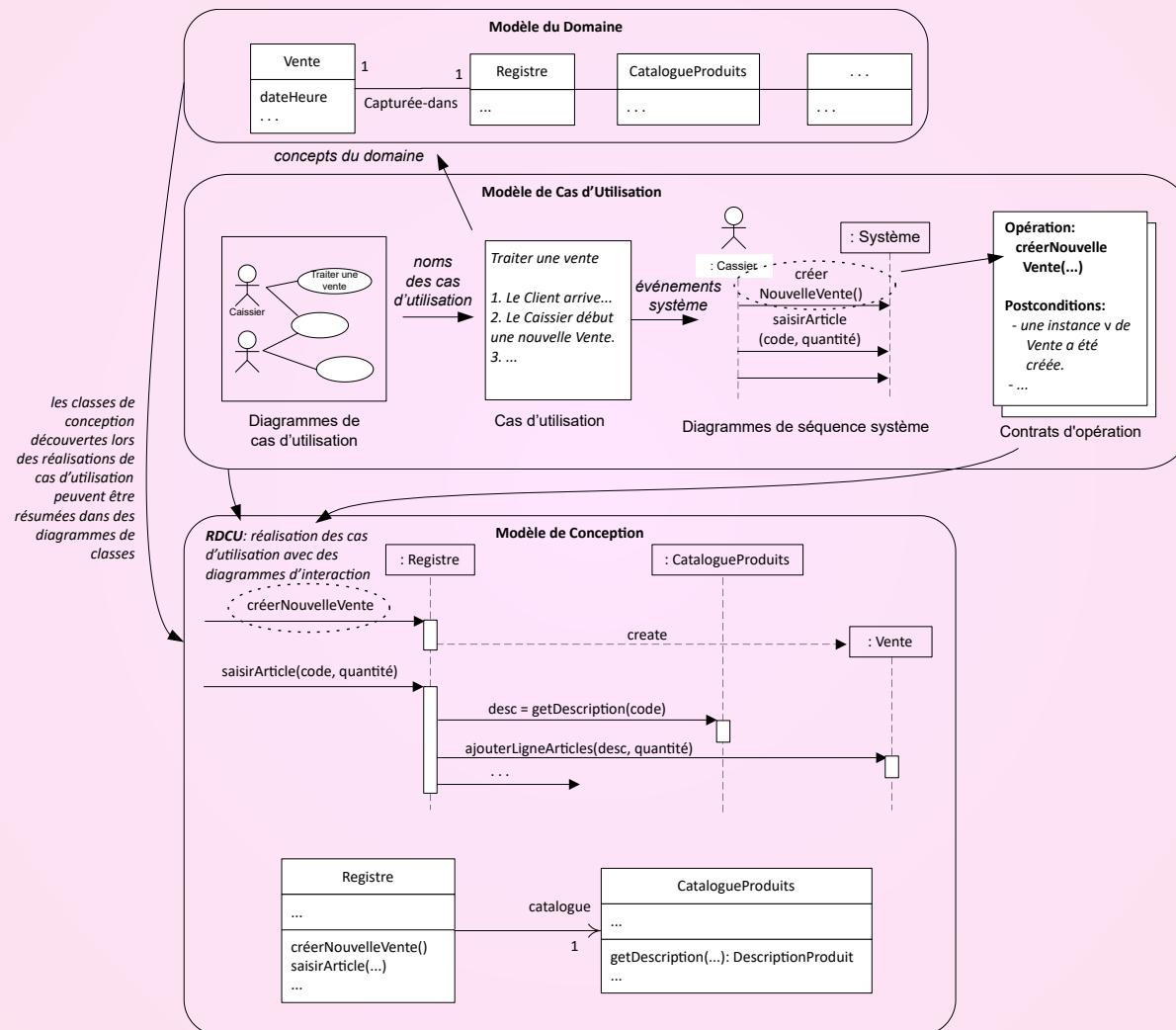
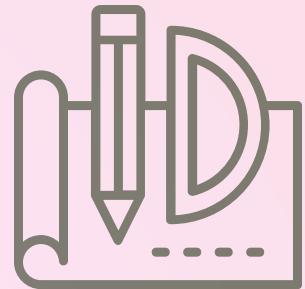
Pour décider où mettre les méthodes...

GRASP



- Contrôleur (séparation des couches)
- Créateur
- Expert en information
- Faible couplage
- Forte cohésion
- Polymorphisme
- Fabrication pure
- Indirection
- Protection de variations

RDCU (SURVOL)



« ÉVALUATION »

- Faible couplage (évaluation)
- Forte cohésion (évaluation)

On applique ces principes après un autre principe, pour évaluer lorsqu'il y a plusieurs choix de conception.

FAIBLE COUPLAGE

FAIBLE COUPLAGE

Problème: Comment réduire l'impact des modifications?

Solution: Affecter les responsabilités de sorte à éviter tout couplage inutile. Appliquer ce principe pour évaluer les solutions possibles.

QUI CRÉE UN PAIEMENT?

Selon GRASP Créeateur:

1. un Registre « enregistre » un Paiement, alors
Registre crée
2. une Vente « utilise étroitement » un Paiement, alors
Vente crée

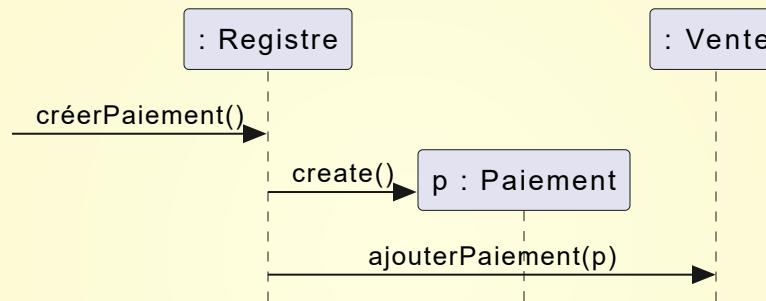
Comment décider? On évalue les deux possibilités sur
le plan du couplage.

ÉVALUER SUR LE PLAN DE LA COUPLAGE



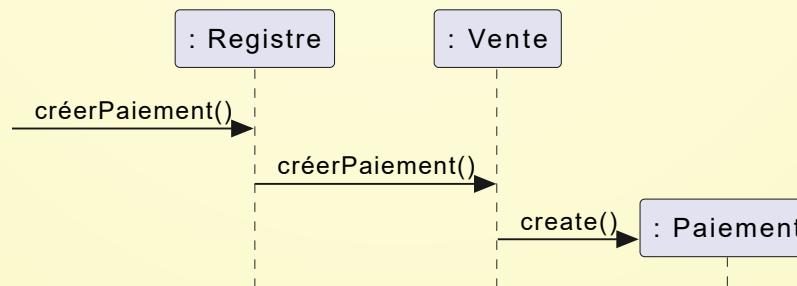
1.

Le Registre crée un Paiement



2.

La Vente crée un Paiement



LE VRAI PROBLÈME DU COUPLAGE

Ce n'est pas le couplage fort en tant que tel. C'est le **couplage fort vers les choses instables**:

- une classe dont son API risque de changer
- les classes de la couche de présentation (vues, le patron Observateur résout ce problème)
- un module externe hors de notre contrôle (dépendances npm sont un bel exemple)

FORTE COHÉSION

FORTE COHÉSION

Problème: Comment s'assurer que les objets restent compréhensibles et faciles à gérer, et, bénéfice second, qu'ils contribuent au Faible Couplage?

Solution: Affecter les responsabilités pour que la cohésion demeure élevée. Appliquer ce principe pour évaluer les solutions possibles.

RAPPEL: COHÉSION



Socrative → ETSLOG210

Vrai ou Faux?



a plus de cohésion que

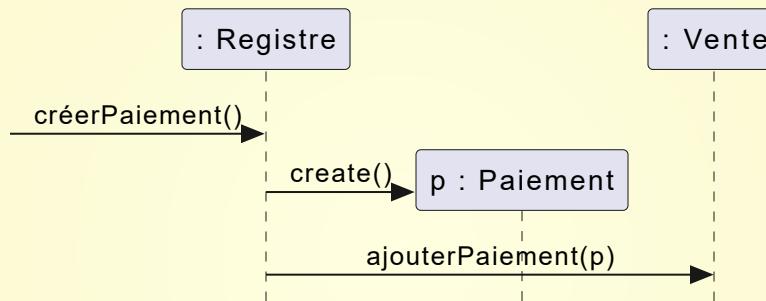


ÉVALUER SUR LE PLAN DE LA COHÉSION



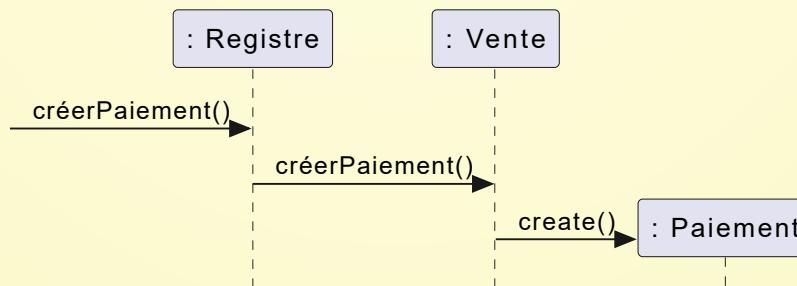
1.

Le Registre crée un Paiement



2.

La Vente crée un Paiement



POLYMORPHISME

POLYMORPHISME

Problème: Comment gérer des alternatives dépendantes des types? Comment créer des composants logiciels “enfichables (pluggables)” ?

Solution: Affectez les responsabilités, en utilisant des opérations polymorphes aux types pour lesquels le comportement varie.

Corollaire: Ne testez pas le type d'un objet et n'utilisez pas de logique conditionnelle pour apporter des alternatives basées sur le type.

POLYMORPHISME - EXEMPLES DU LIVRE

- Calculateurs de taxe externes (`getTaxes()`)
 - Interface implementée par chaque type (classe) différent
- Types de case Monopoly (`atterrirSur()`)
 - Méthode abstraite implementée par chaque classe concrète

POLYMORPHISME

RÉUSINAGE

- “Replace conditional with polymorphism”
- Description de l'exercice
- Exercice en équipe sur Google Docs



FABRICATION PURE

FABRICATION PURE

Problème: Quel est l'objet qui doit être responsable lorsqu'on ne veut pas transgresser les principes de *Faible Couplage* et de *Forte Cohésion* mais que les solutions offertes par Expert (par exemple) ne sont pas appropriées?

Solution: Affecter un ensemble de responsabilités fortement cohésif à une class artificielle ou de commodité qui ne représente pas un concept du domaine; il s'agit d'une entité fabriquée de toutes pièces pour prendre en charge la forte cohésion, le faible couplage et la réutilisation.

FABRICATION PURE

- Exemples du livre
 - StockagePersistant (s'occupe des responsabilités pour stocker des objets)
 - “Cornet” entre Joueur et Dé (réutilisable dans d’autres jeux)

CONTRÔLEUR ET FABRICATION PURE



Solution du Contrôleur: Affectez une responsabilité à la classe qui correspond à l'une de ces définitions:

1. Elle représente le système global, un « objet racine », un équipement ou un sous-système (contrôleur de façade).
2. Elle représente un scénario de cas d'utilisation dans lequel l'opération système se produit (contrôleur de session ou contrôleur de cas d'utilisation). On la nomme GestionnaireX, où X est le nom du cas d'utilisation.

FABRICATION PURE

Exemple: Choix no. 2 dans GRASP Contrôleur
(contrôleur de session ou de cas d'utilisation) est une
Fabrication Pure.

CONTRÔLEUR SURCHARGÉ DE RESPONSABILITÉS



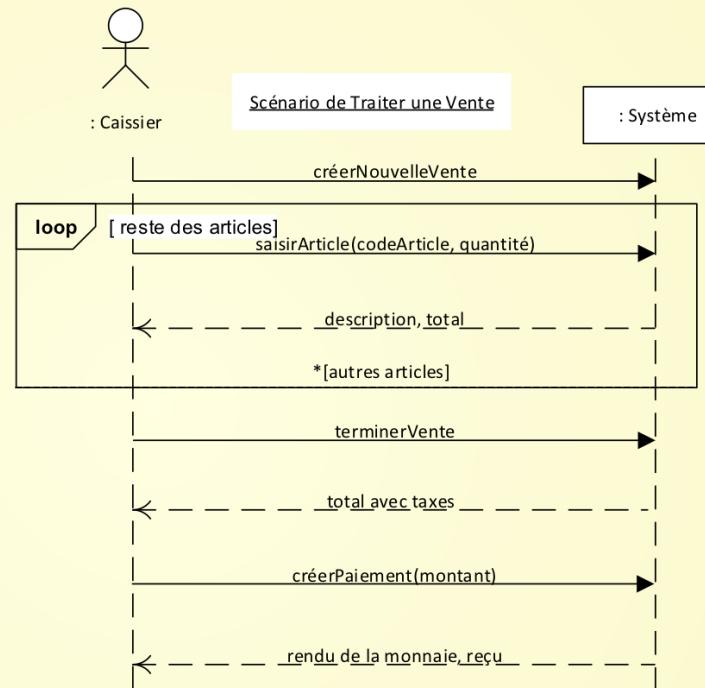
Scénario simple de paiement en espèces
de Traiter une Vente:

1. Le Client arrive à la caisse avec ses achats.
2. Le Caissier entame une nouvelle vente.

3. Le Caissier saisit le code des articles.
4. Le Système enregistre les références d'article et présente leur description, leur prix et le total courant.

Le caissier répète l'opération (étapes 3 et 4) pour tous les articles.

5. Le Système présente le total avec taxes.
6. Le Caissier indique le montant total au client et l'invite à payer.
7. Le Client paie et l'opération est traitée par le Système.
- ...

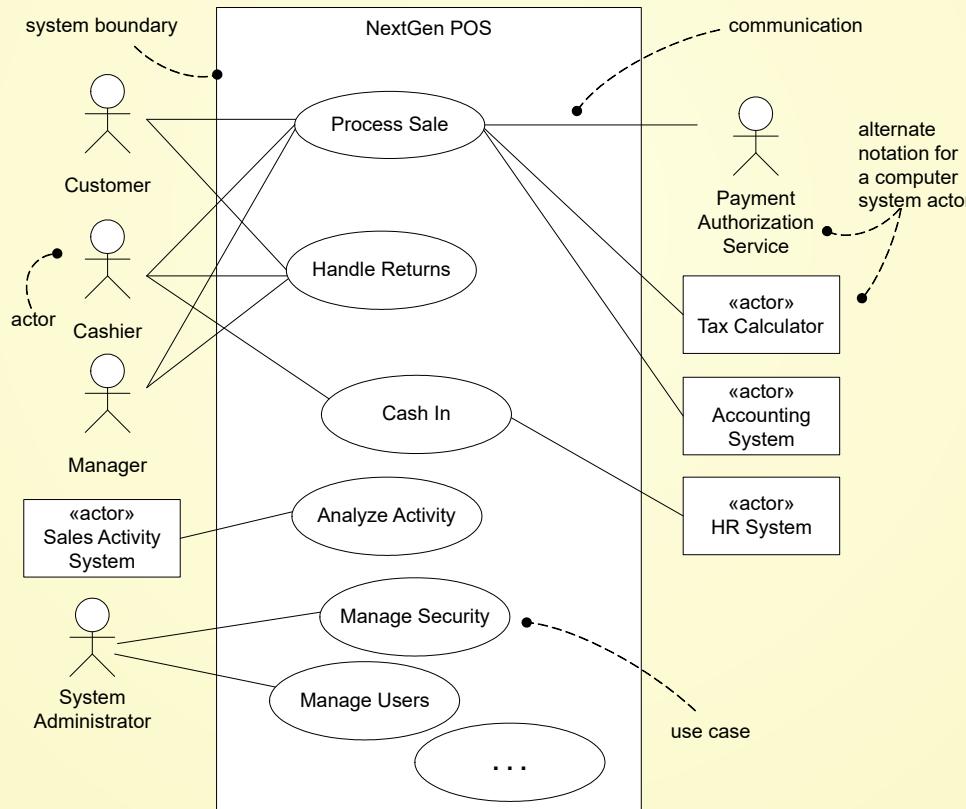


“Traiter vente” nécessite des opérations systèmes envoyées à Registre (équipement, GRASP Contrôleur)

REGISTRE “EXPLOSE” DE RESPONSABILITÉS



Tous les cas d'utilisation qui se déroulent sur la caisse
(Registre) → situation désespérée 😭

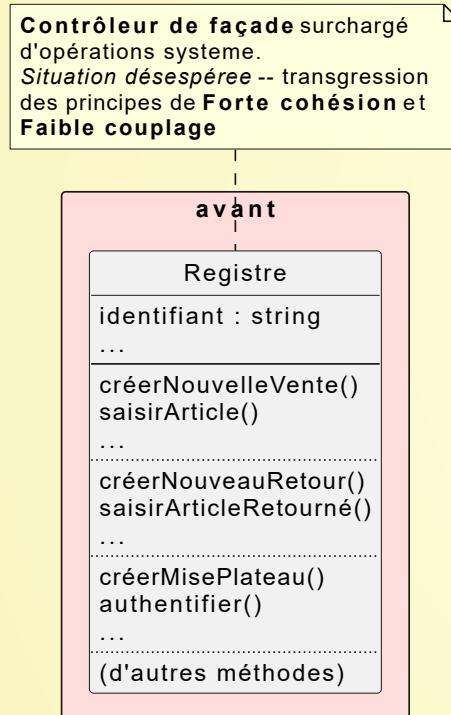


FABRICATION PURE

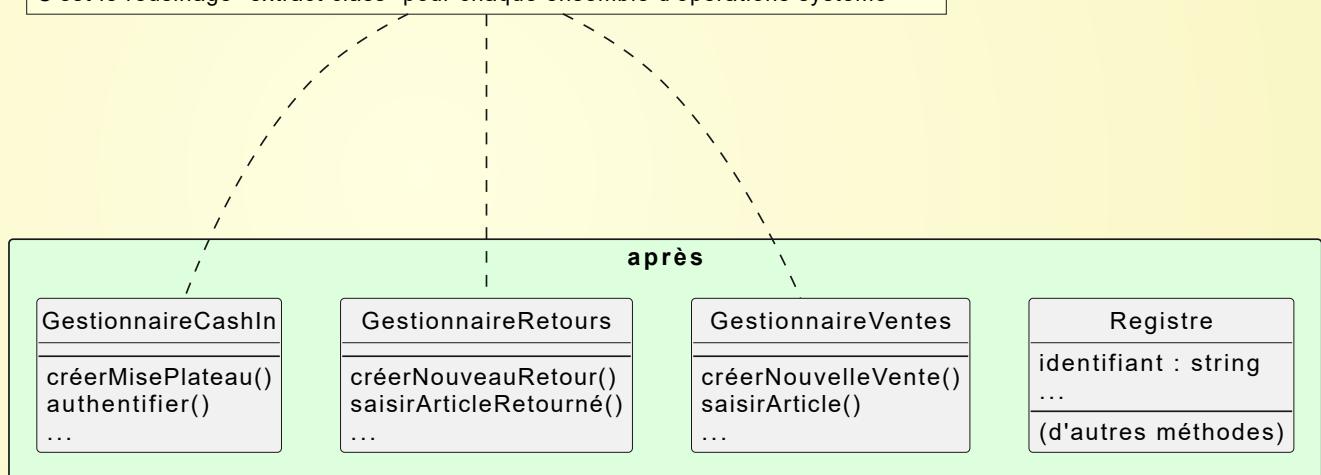
RÉUSINAGE

Extract (methods) to Class

SOLUTION : FABRICATION PURE



Fabrications Pures avec Contrôleurs de session (de cas d'utilisation).
C'est le réusinage "extract class" pour chaque ensemble d'opérations système



FORCES PARFOIS OPPOSÉES

1. Principe: Réduire l'écart des représentations
2. Principe: Faible couplage et Forte cohésion

Fabrication Pure → Gestionnaire Ventes
Représentation facile à comprendre? 😐

Peut-on “équilibrer” ces forces?

PATTERNS GOF ET FABRICATION PURE

VISITOR, MODERATOR, OBSERVER, ETC.

- Abstractions (classes ou types interface) ayant un ensemble cohésif de responsabilités.
- Augmentent l'écart des représentations:
 - Solutions au problèmes de design (plutôt que du domaine).
 - Complexité circonstancielle.



INDIRECTION

INDIRECTION

- **Problème:**

- Où affecter une responsabilité pour éviter le couplage entre deux entités ou plus?
- Comment découpler les objets pour maintenir le potentiel de réutilisation?

- **Solution:**

- Affecter la responsabilité à un objet qui sert d'intermédiaire entre d'autres composants ou services pour éviter de les coupler directement.
- Le intermediaire crée une indirection entre les autres composants.

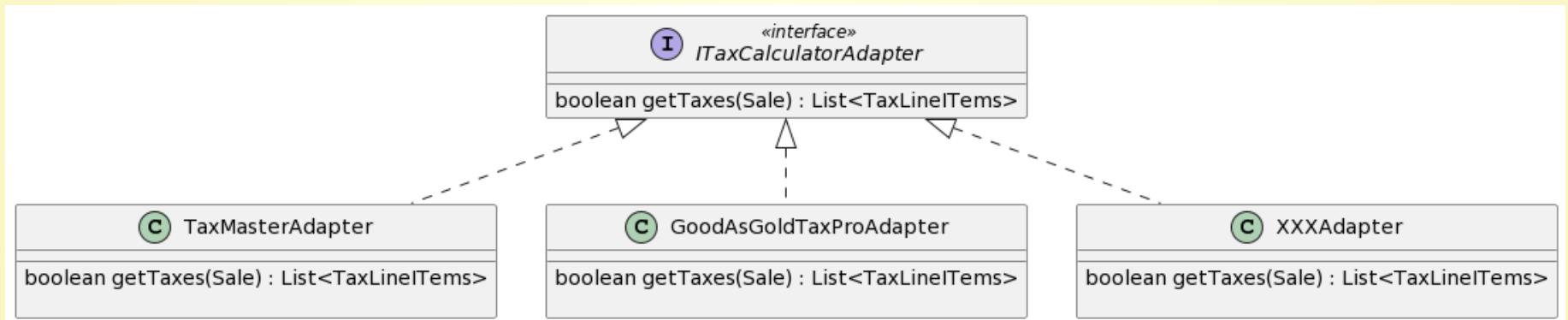
INDIRECTION

Exemples du livre :

- AdaptateurCalculateurTaxes
- StockagePersistant

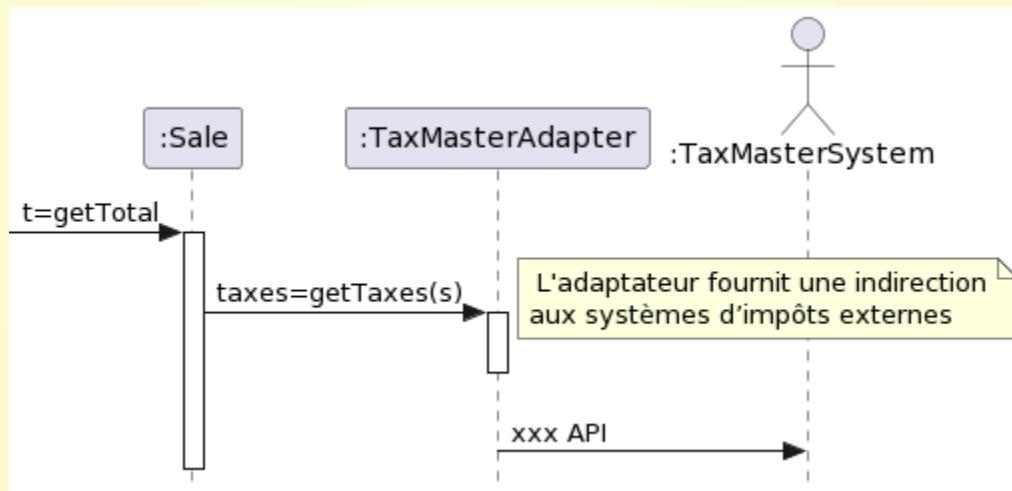
INDIRECTION - TAXADAPTER

DIAGRAMME DE CLASSES



INDIRECTION - TAXADAPTER

DIAGRAMME DE SÉQUENCES





→ ETSLOG210

Qu'est-ce qui est faux dans le problème de design des calculateurs de taxes?

- A. On ne peut changer l'API des calculateurs de taxes.
- B. On ne veut pas que le “core” de NextGenPOS soit couplé aux API différentes des calculateurs.
- C. Pour intégrer un nouveau calculateur de taxes, on n'à qu'à écrire son adaptateur.
- D. Chaque adaptateur a une API différente pour fonctionner avec son calculateur externe.
- E. Aucune de ces réponses.

INDIRECTION – D'AUTRES EXEMPLES

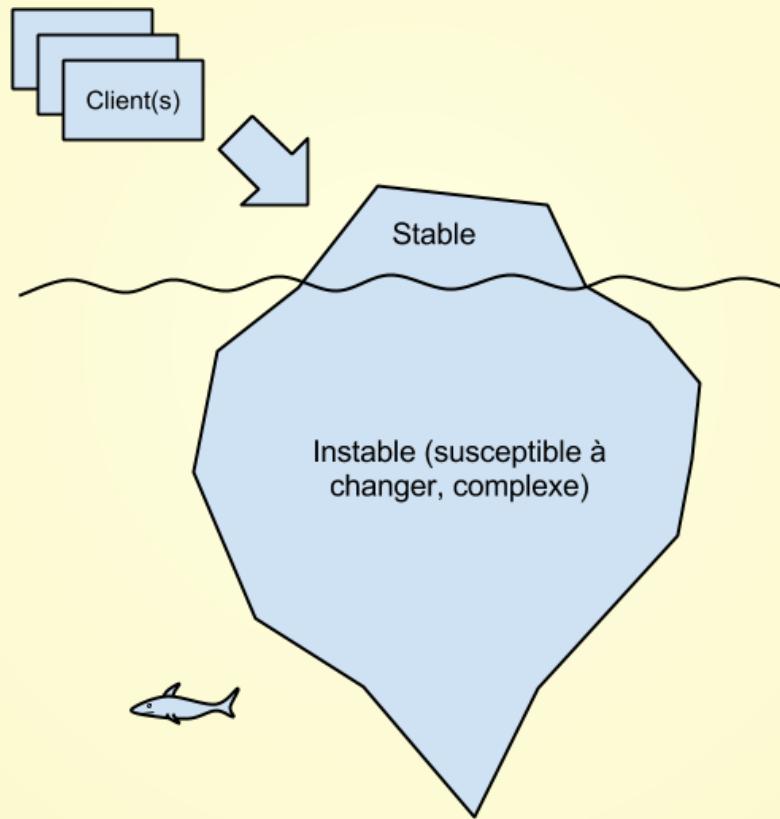
- Contrôleur de cas d'utilisation (de session) est une indirection qui
 - « découpe les objets pour maintenir le potentiel de réutilisation. »
 - une application du patron GoF « Façade »
- Architecture en couches
 - <https://log210-cfuhrman.github.io/log210-valider-architecture-couches/#/0/2>

PROTECTION DES VARIATIONS

PROTECTION DES VARIATIONS

- **Problème:**
 - Comment concevoir des objets, des sous-systèmes, ou de systèmes de telle façon que les variations ou l'instabilité de ces éléments n'aient pas d'impact indésirable sur d'autres éléments?
- **Solution:**
 - Identifier les points de variation ou d'instabilité prévisibles. Affecter les responsabilités pour créer une interface (sens large) stable autour d'eux.

PROTECTION DES VARIATIONS



PROTECTION DES VARIATIONS - EXEMPLE

- Problème du calculateur de taxes
 - Plusieurs variations - calculs de taxes
- Solution:
 - Adaptateur avec la méthode polymorphe
getTaxes
 - Une classe pour chaque variation qui implémente
la méthode getTaxes

D'AUTRES EXEMPLES DE PV

- Conceptions pilotées par les données
 - `tsconfig.json` permet de configurer le générateur de Javascript selon beaucoup de variations
- Une machine virtuelle (JVM)
 - programme en “bytecode” qui peut exécuter sur plusieurs plateformes (qui varient)
- Langages standard:
 - Requêtes SQL qui fonctionnent sur plusieurs BD



PV SPÉCULATIF

Deux types de points de changement:

- **Points de variations:** spécifiés **explicitement** dans les besoins (cahier des charges)
- **Points d'évolution:** points de variation « spéculatifs » **absents des besoins existants**

Principe YAGNI: « You Ain't Gonna Need It »
Il vaut mieux ne pas spéculer.

RÉSUMÉ GRASP

- Faible couplage (évaluation)
- Forte cohésion (évaluation)
- Polymorphisme
- Fabrication Pure
- Indirection
- Protection des variations

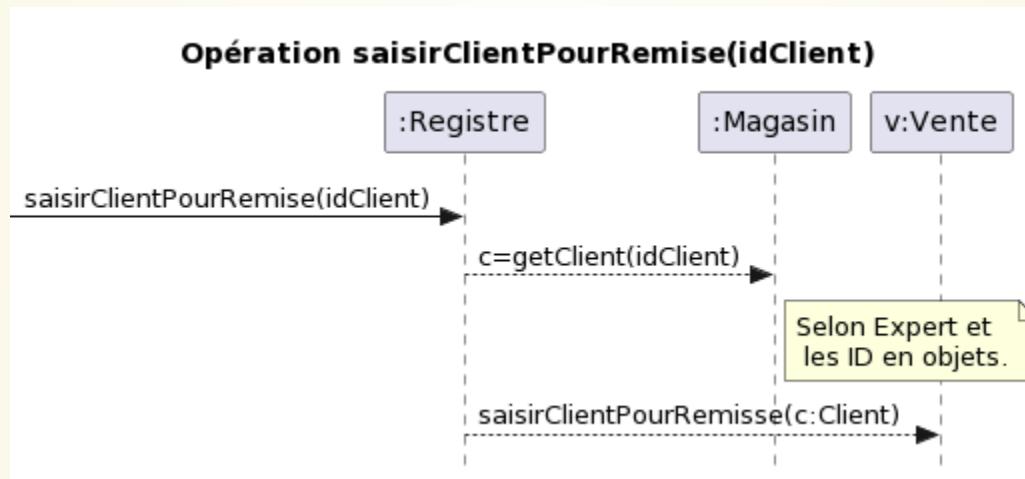
PATRON TRANSFORMER ID EN OBJETS

(NDC 9.4)

- Définition :
 - Obtenir la référence d'un objet basé sur une valeur d'un attribut comme ID.
- Pour l'implémentation on utilise souvent un tableau associatif (Map).

PATRON TRANSFORMER ID EN OBJETS

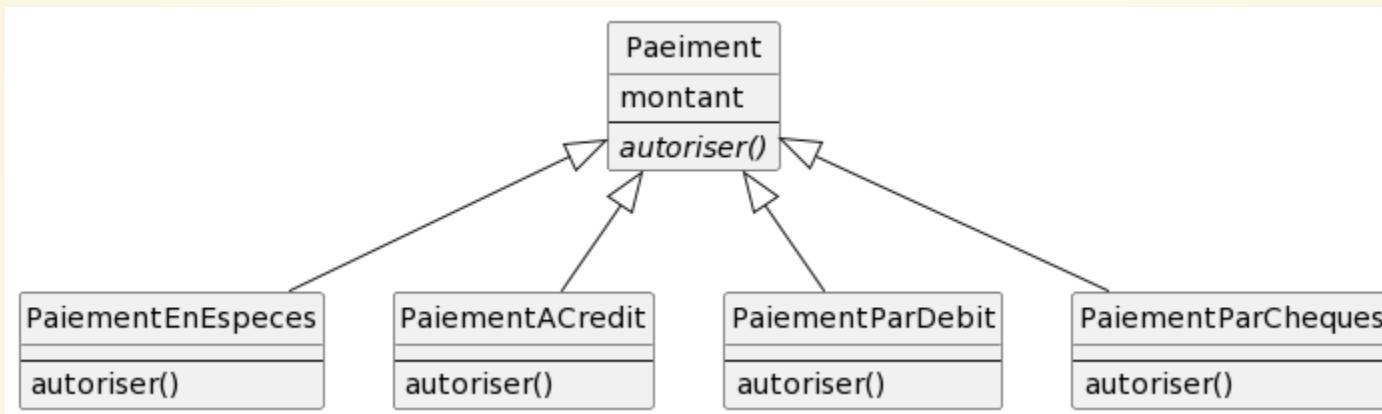
EXEMPLE



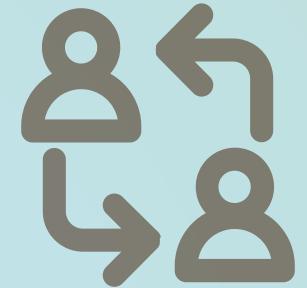
PATRON “FAIRE SOI-MÊME” (NDC 9.8)

- Définition :
 - Moi, objet logiciel, je fais moi-même ce qu'on fait normalement à l'objet réel dont je suis une abstraction.
- Typiquement appliqué ensemble avec GRASP Polymorphisme

PATRON “FAIRE SOI-MÊME” EXEMPLE



Intuition: Chaque sous-classe implémente elle-même sa propre autorisation.



Created by Prithvi
from the Noun Project

FEUILLE D'UNE MINUTE

SVP m'écrire un courriel pour dire ce qu'étaient les points les moins clairs de la séance.