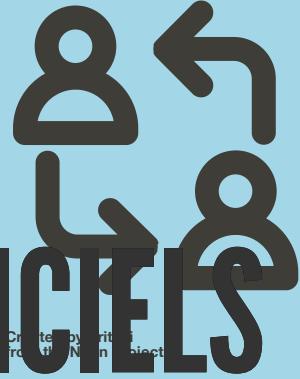


# LOG210 SÉANCE #08

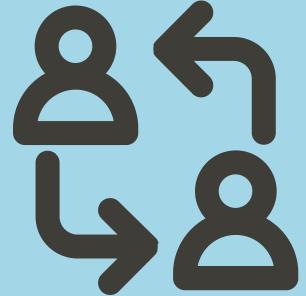
## ANALYSE ET CONCEPTION DE LOGICIELS



1. Administration
2. Équipe
3. Première étude de la conception
4. Outils UML
5. GRASP
6. GOF Révision
7. Diagramme de package
8. TDD en pratique



# ADMINISTRIVIA



- Statistiques Intra
- Consultation individuelle par Zoom/Discord selon mes disponibilités

Created by Prithvi  
from the Noun Project

Élaborer: Outils/patron de réusinage

# LOG210 SÉANCE #08

## ANALYSE ET CONCEPTION DE LOGICIELS



1. Administration
2. Équipe ←
3. Première étude de la conception
4. Outils UML
5. GRASP
6. GOF Révision
7. Diagramme de package
8. TDD en pratique

# TRAVAIL D'ÉQUIPE



- Quels sont les problèmes HRC rencontré par votre équipe?
- Quels solutions avez vous apportées?

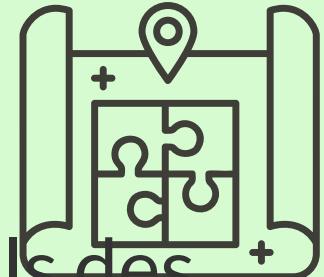
# LOG210 SÉANCE #08

## ANALYSE ET CONCEPTION DE LOGICIELS



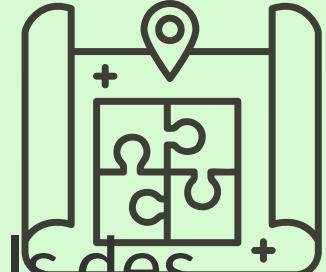
1. Administration
2. Équipe
3. Première étude de la conception 
4. Outils UML
5. GRASP
6. GOF Révision
7. Diagramme de package
8. TDD en pratique

# CONCEPTION D'OBJET 1/2



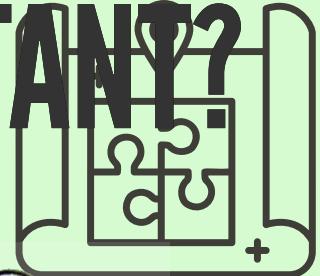
- Comment les développeurs conçoivent-ils des objets?
  - Ils codent. On conçoit en codant (vous avez fait ça au CÉGEP, en INF111, en LOG121, en stage?)
  - Outils de réusinage (refactoring)

# CONCEPTION D'OBJET 2/2

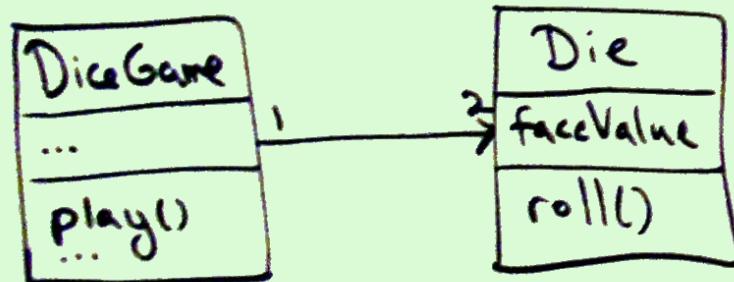


- Comment les développeurs conçoivent-ils des objets?
  - Ils tracent des diagrammes puis ils codent.
  - UML au tableau blanc, puis Java dans Eclipse
- Ils se contentent de tracer des diagrammes.
  - Les outils ne sont pas encore capables de faire le codage à partir des diagrammes... (est-ce vrai?)

# QU'EST-CE QUI EST PLUS IMPORTANT?

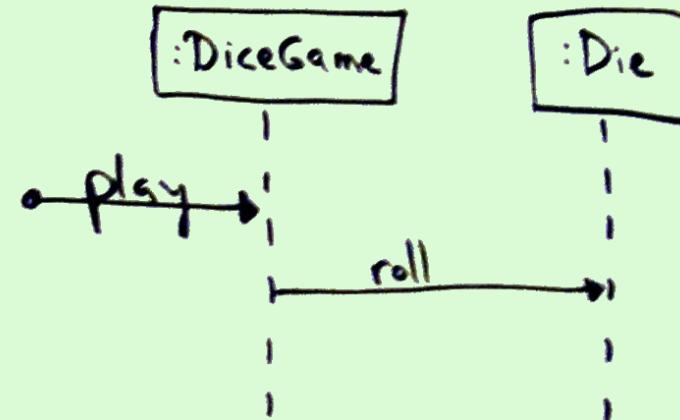


Static model



UML Class Diagram

Dynamic Model



UML Sequence Diagram

# CARTES CRC



- Classe, Responsabilité, Collaborateurs
- Utilisées pour faire le design

## 1. LOG121 (Cay Horstmann, chap: 2.12):

Mailbox	
<i>manage passcode</i>	MessageQueue
<i>manage greeting</i>	
<i>manage new and saved messages</i>	

# EXEMPLE CRC

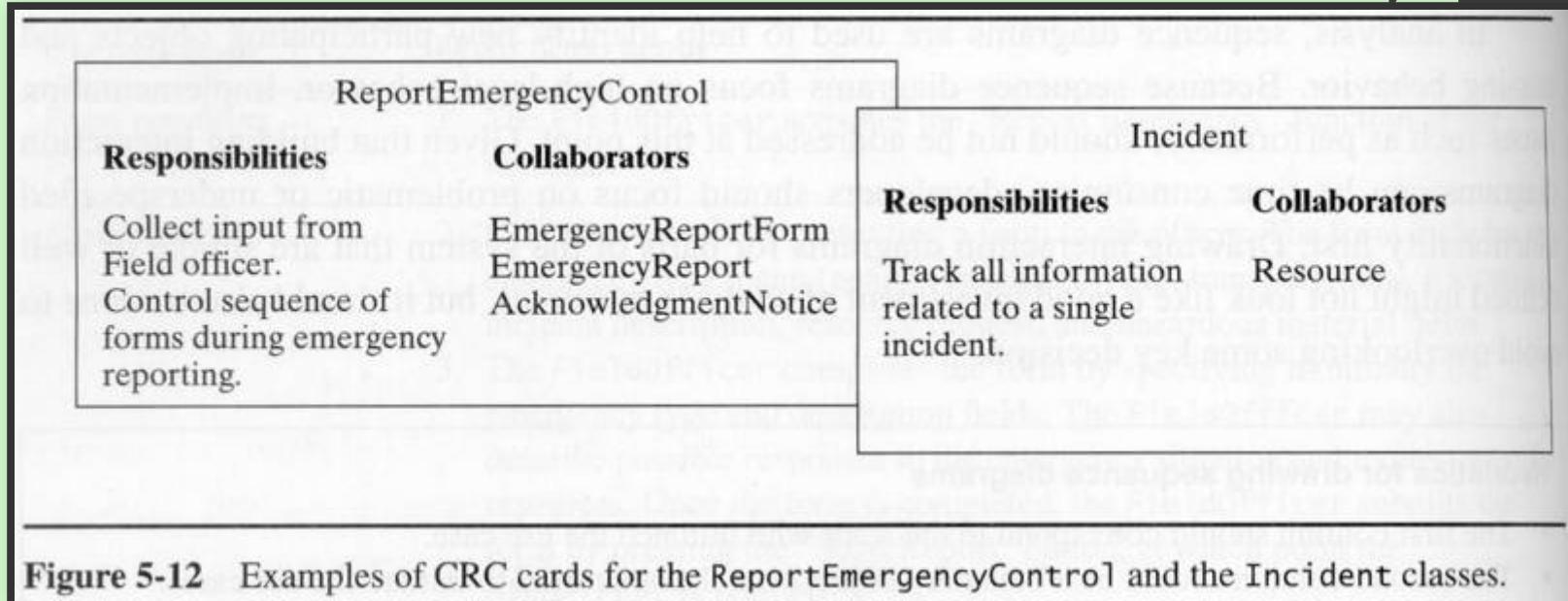
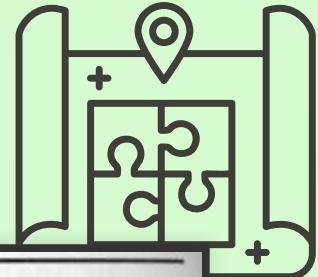
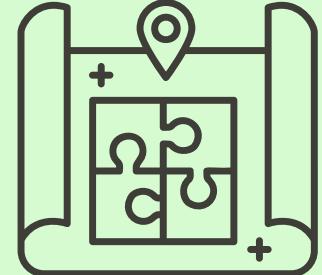


Figure 5-12 Examples of CRC cards for the ReportEmergencyControl and the Incident classes.

ref: Object-Oriented Software Engineering, Bruegge & Dutoit, Prentice-Hall, 2000.

# RÉSUMÉ



- Modélisation agile
  - Caméra pour numériser les esquisses
  - Habilité avec UML
- Modélisation dynamique est importante
- Pas beaucoup de temps pour modéliser
  - Une itération de 3 semaines (120 h)
  - 2 à 8 heures de modélisation, pas plus

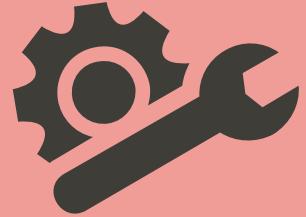
# LOG210 SÉANCE #08



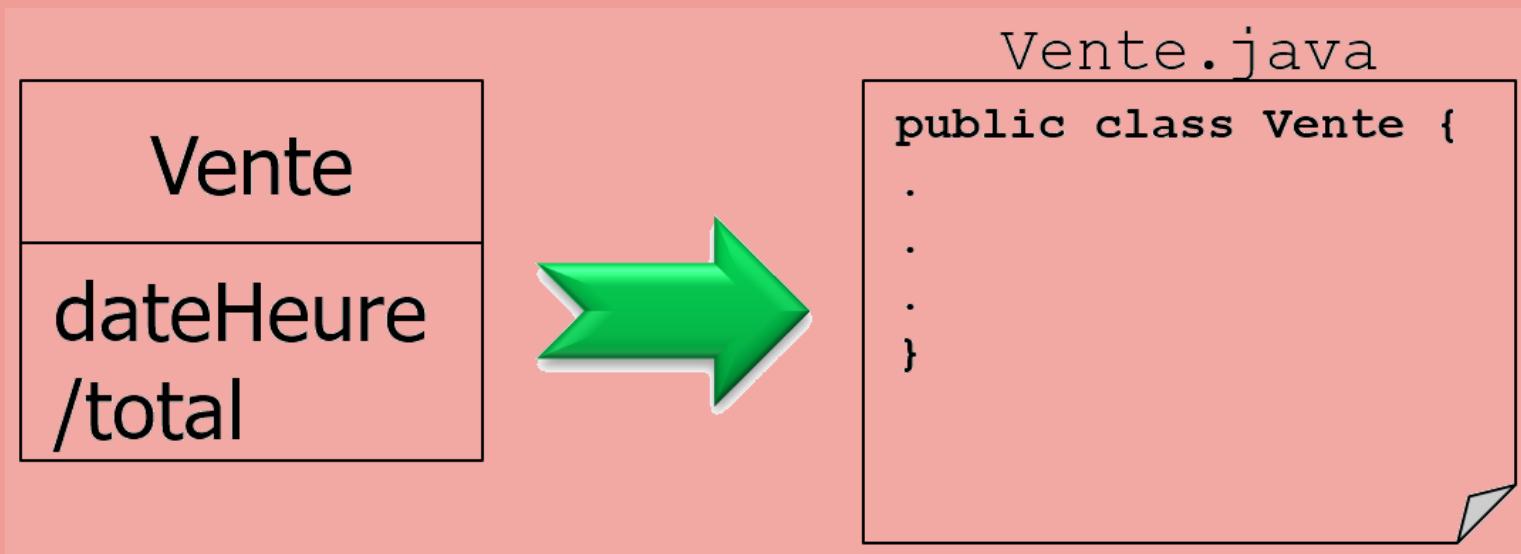
## ANALYSE ET CONCEPTION DE LOGICIELS

1. Administration
2. Équipe
3. Première étude de la conception
4. Outils UML ← S20203
5. GRASP
6. GOF Révision
7. Diagramme de package
8. TDD en pratique

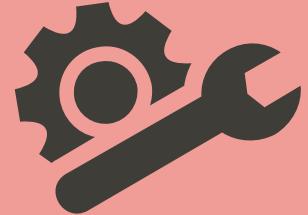
# DÉFINITIONS



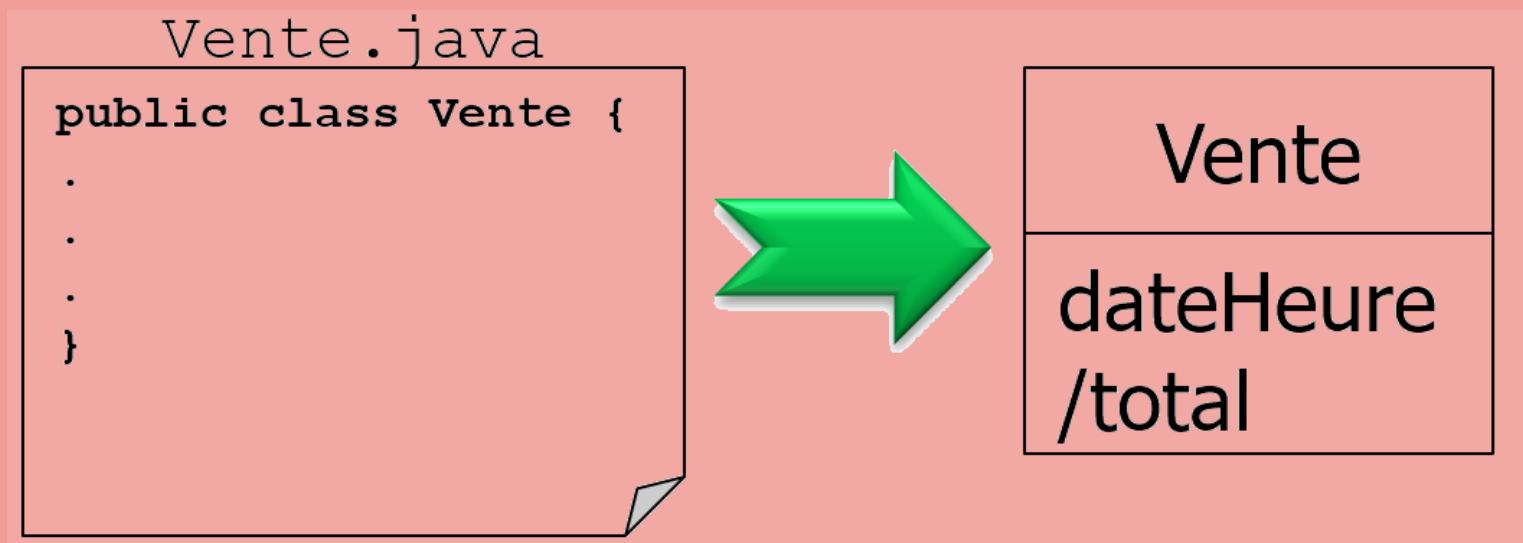
- Pro-ingénierie
  - Générer du code source à partir des diagrammes (UML)



# DÉFINITIONS



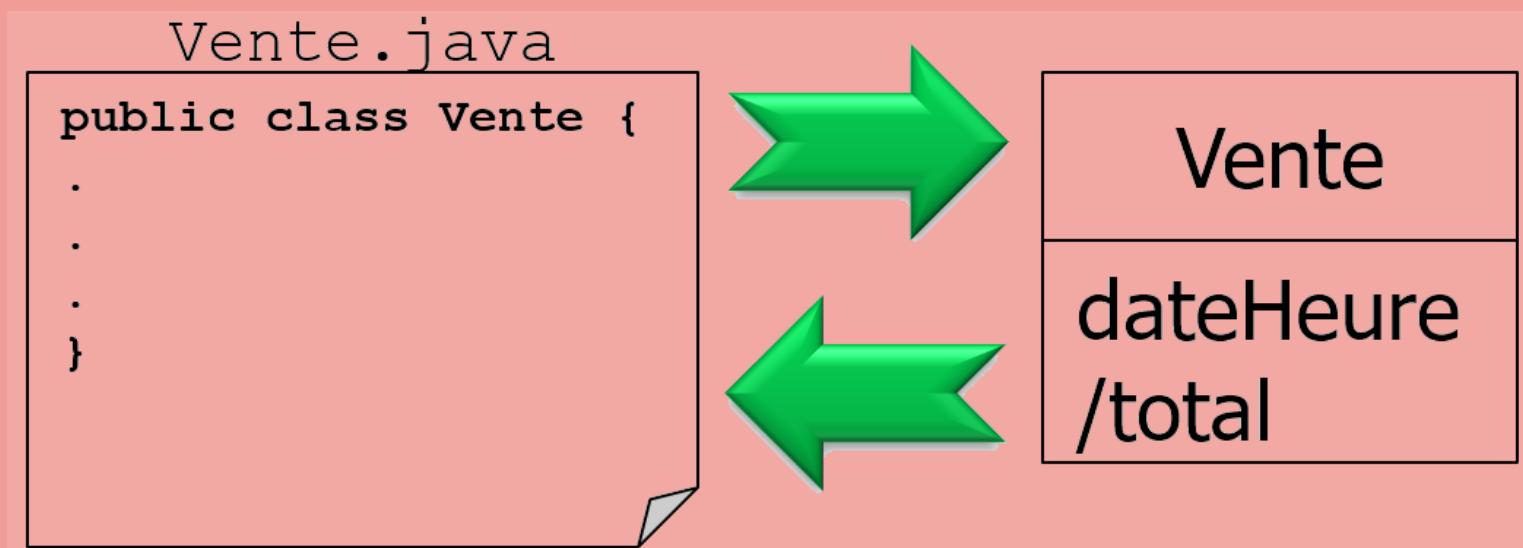
- Rétro-ingénierie
  - Générer des diagrammes (UML) à partir du code



# DÉFINITIONS



- Ingénierie cycle
  - Fermer cette boucle



# QUE DOIT-ON RECHERCHER DANS UN OUTIL?



- Essayez un outil gratuitement avant de l'acheter.
- Testez l'outil sur un projet réel.
- Choisissez un outil qui s'intègre à votre IDE préféré (VS, Eclipse, etc.)
- Rétro-ingénierie est important
- L'impression sur grande feuilles est importante (Larman)

Created by  
From the Ward Project



# LISTE D'OUTILS UML



- Suggestions:
  - Papyrus
  - Modeliosoft
  - PlantUML

## 1. Wikipedia

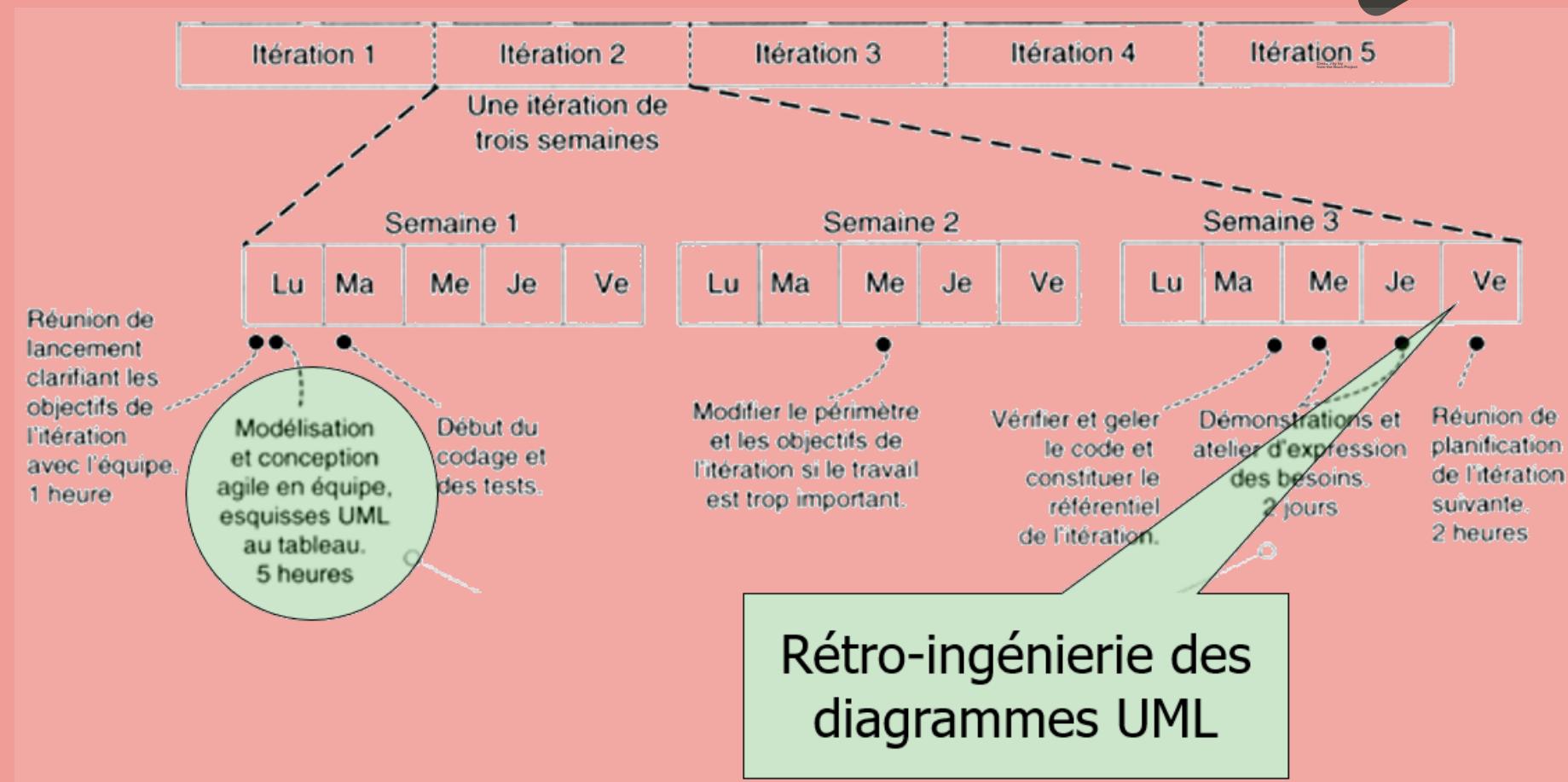
# UML EN MODE ESQUISSE



- Esquisses murales UML au début d'une itération
- Suivies par 3 semaines de codage et de tests
  - Les modèles évolueront avec le code
- Entamer de nouveau la modélisation de nouvelles esquisses
- Générer les diagrammes à partir du code existant le jours précédent cette modélisation

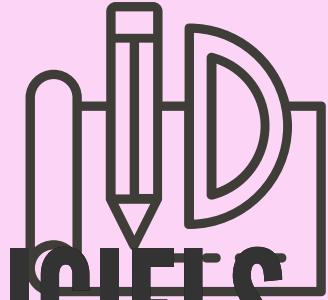


# EXEMPLE D'UNE ITÉRATION



# LOG210 SÉANCE #08

## ANALYSE ET CONCEPTION DE LOGICIELS

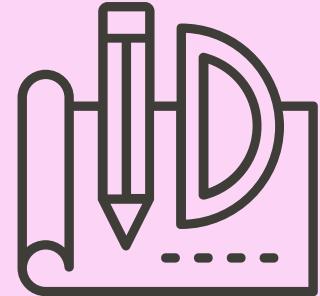


1. Administration
2. Équipe
3. Première étude de la conception
4. Outils UML
5. GRASP ← S20203
6. GOF Révision
7. Diagramme de package
8. TDD en pratique

# AUTRES PATTERNS D'AFFECTATION DES RESPONSABILITÉS

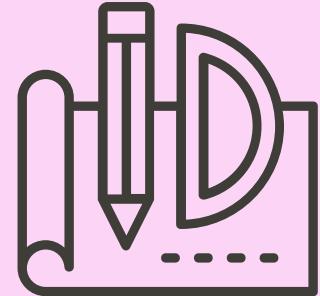


# LES NEUF PATRONS GRASP



- ✓ Expert
- ✓ Créateur
- ✓ Contrôleur
- ✓ Faible Couplage
- ✓ Forte Cohésion
- Polymorphisme
- Fabrication pure
- Indirection
- Protection des variations

# POLYMORPHISME



- Principe de base pour la conception OO
- Cas qui varient mais qui sont similaires
- Dans une famille de classes
  - opération polymorphique
  - dans laquelle les cas sont différents
- Formes géométriques (carré, cercle, triangle, etc.)
  - se dessine avec une méthode draw()
  - chaque classe implémente sa méthode selon son cas...

# POLYMORPHISME

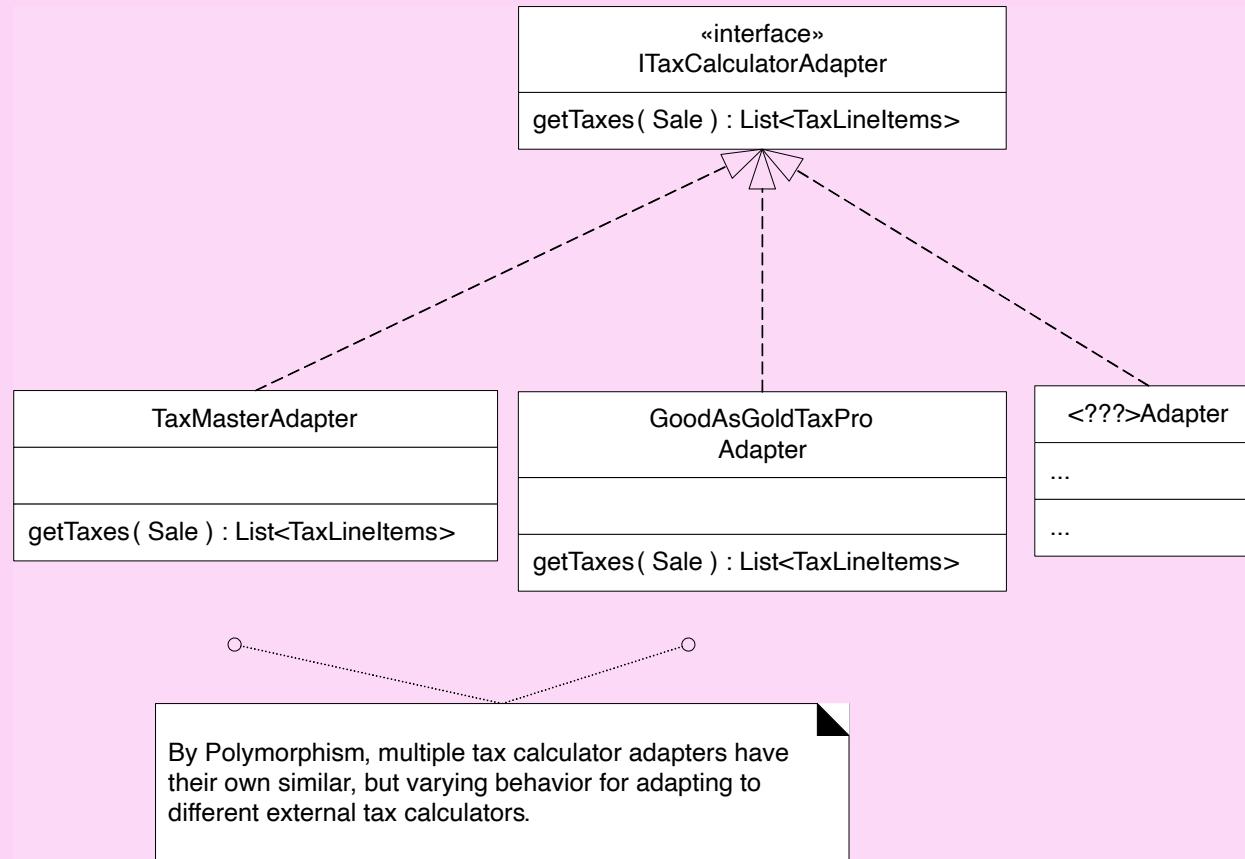
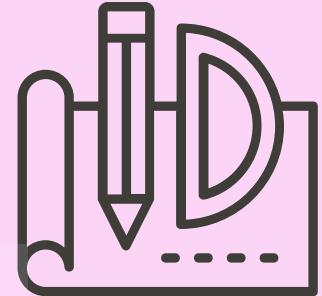


Fig. 25.1 (3e édition)

# POLYMORPHISME

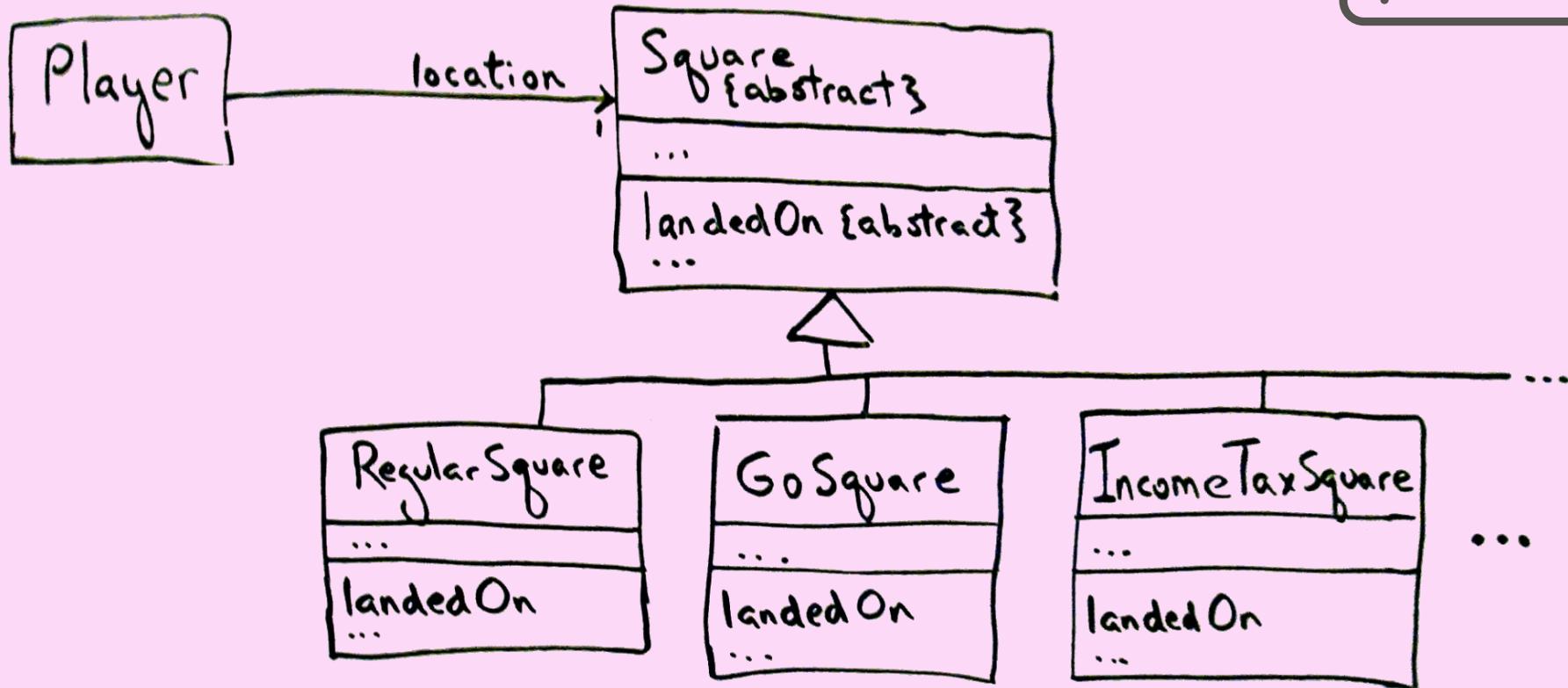
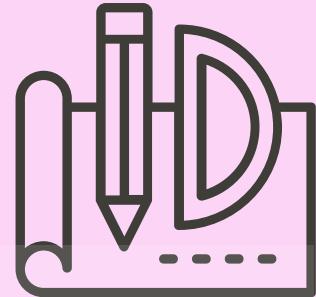


Fig. 25.2 (3e édition)

# POLYMORPHISME

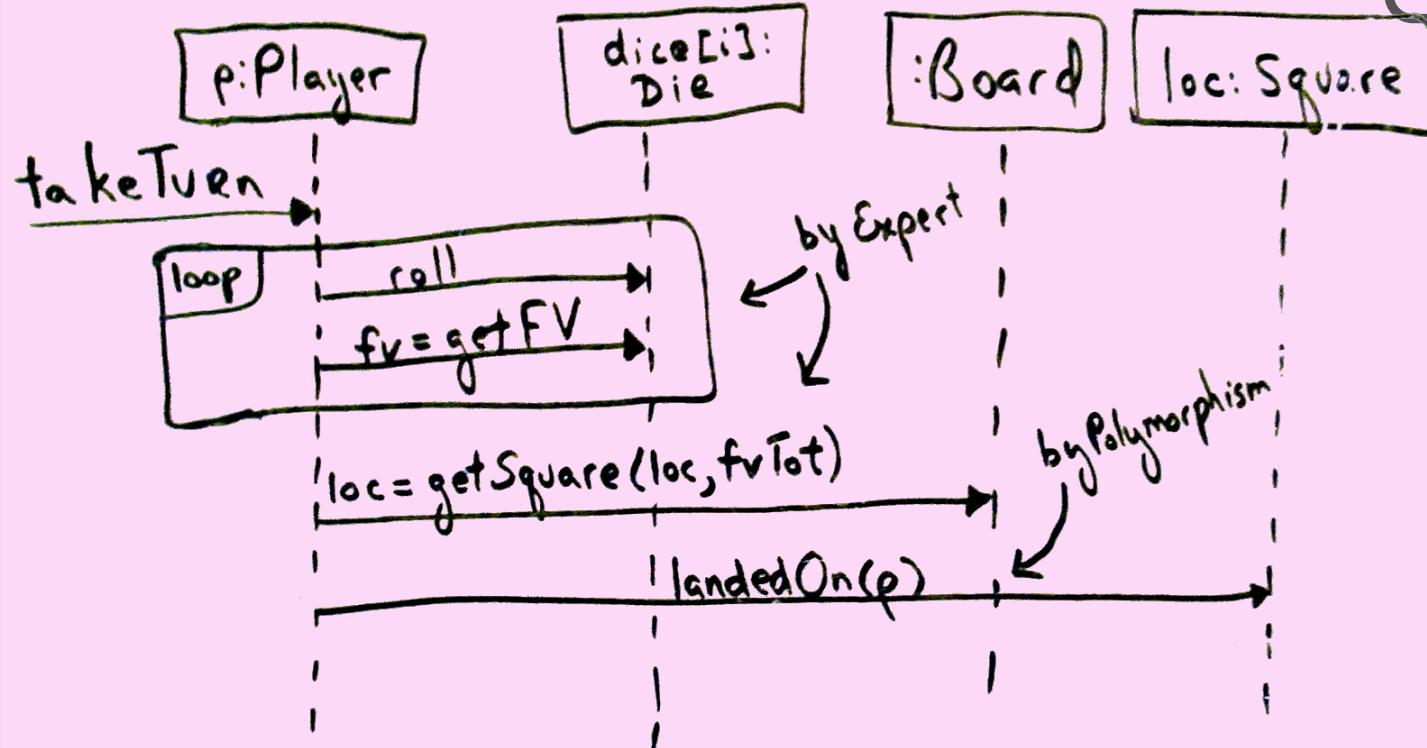
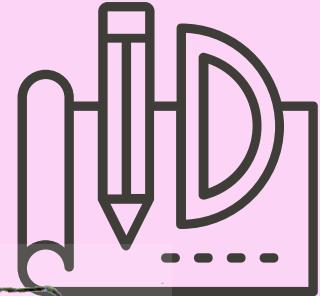


Fig. 25.3 (3e édition)

# POLYMORPHISME

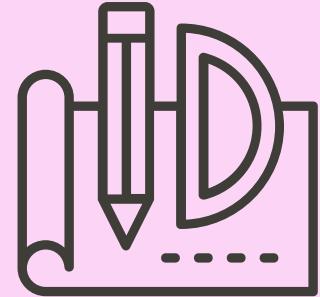
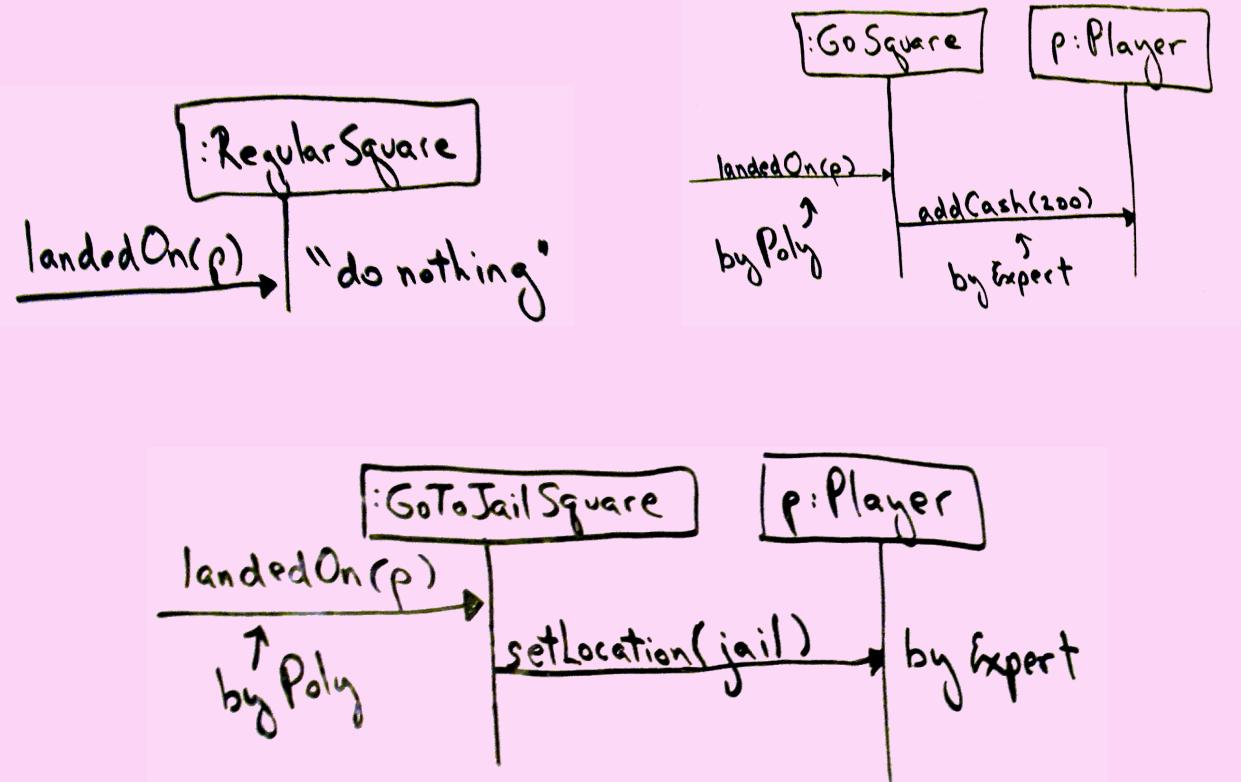
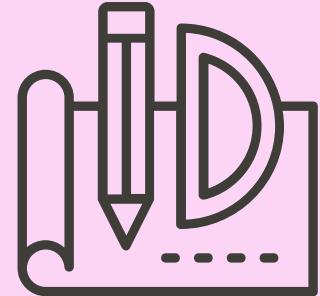


Fig. 25.4, 25.5, 25.7 (3e édition)



# FABRICATION PURE



- Parfois en suivant un patron
  - assigne une responsabilité
  - mais il y a des conséquences non désirables
    - augmentation du couplage
    - diminution de la cohésion
- Solution
  - fabrique une classe « comportementale »
  - qui n'a pas d'équivalent dans le modèle du domaine
  - invention de la part du concepteur

# FABRICATION PURE

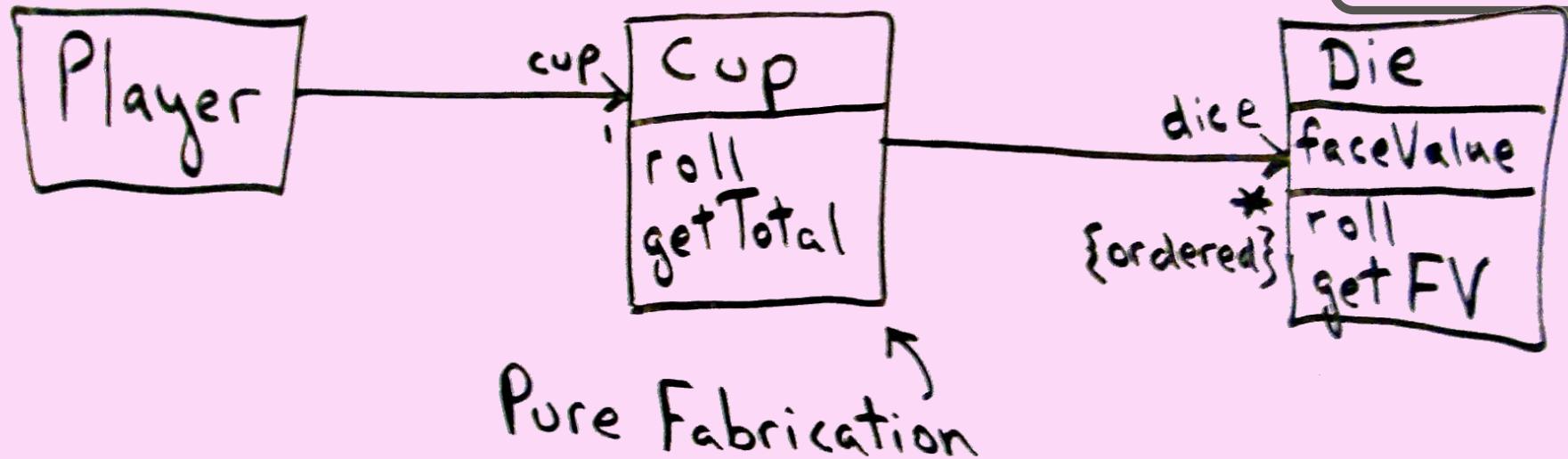
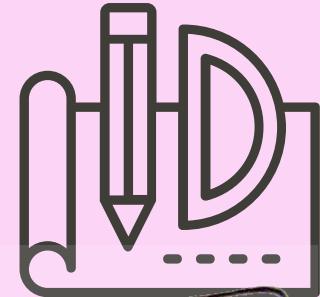


Fig. 25.8 (3e édition)

# FABRICATION PURE

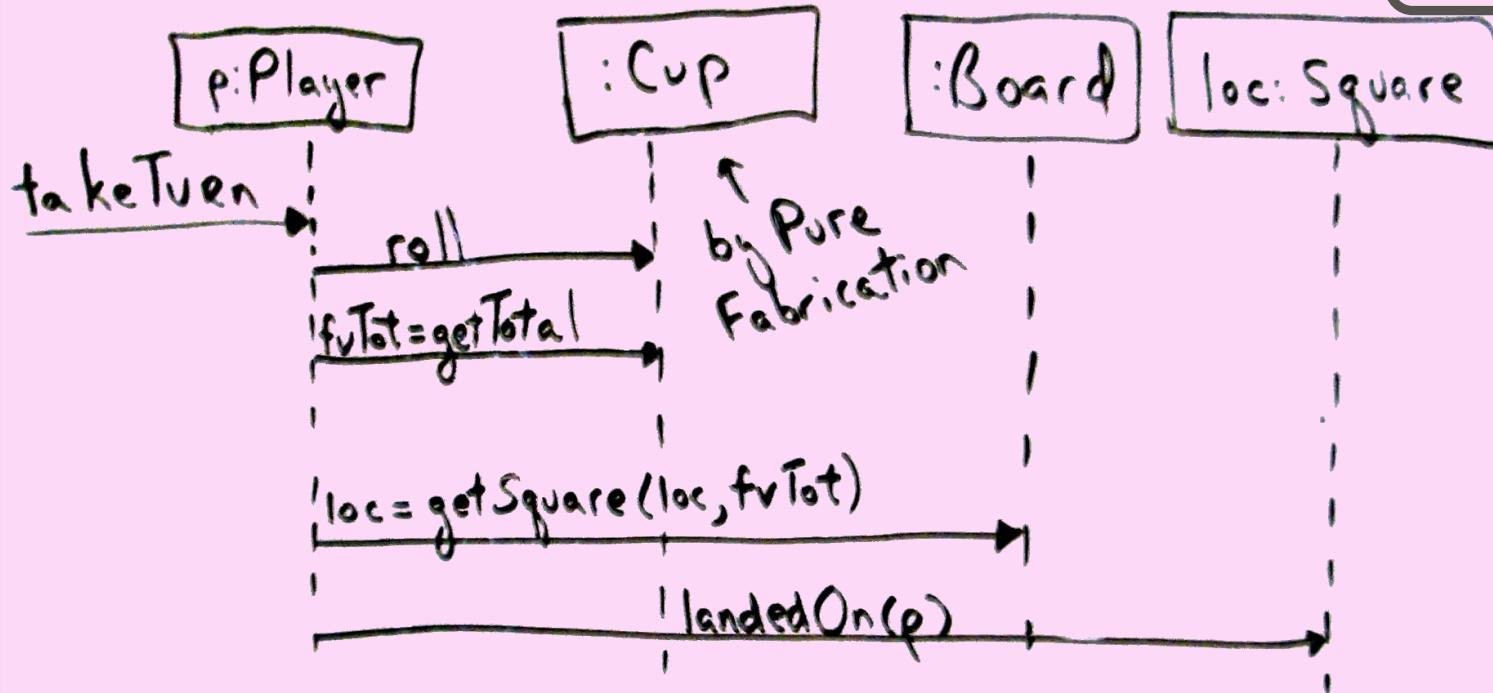
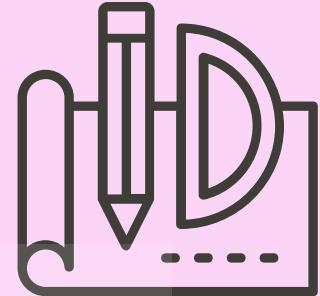
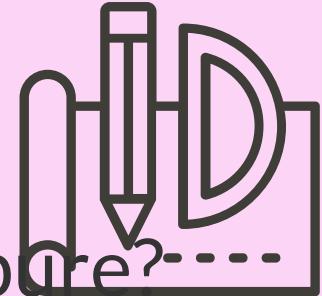


Fig. 25.9 (3e édition)

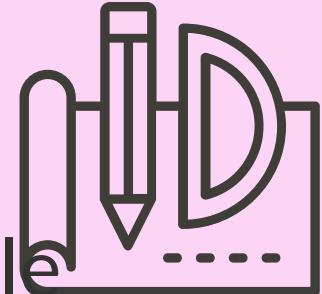
# COMPRÉHENSION



Quel autre GRASP implique une fabrication pure? -----  
Autrement dit, on propose une classe « inventée » ne faisant pas partie du MDD.

1. Créateur
2. Contrôleur
3. Expert
4. Forte cohésion
5. Faible couplage

# INDIRECTION



- Mécanisme souvent utilisé pour réduire le couplage
- Objet intermédiaire
  - découple deux composants

# INDIRECTION

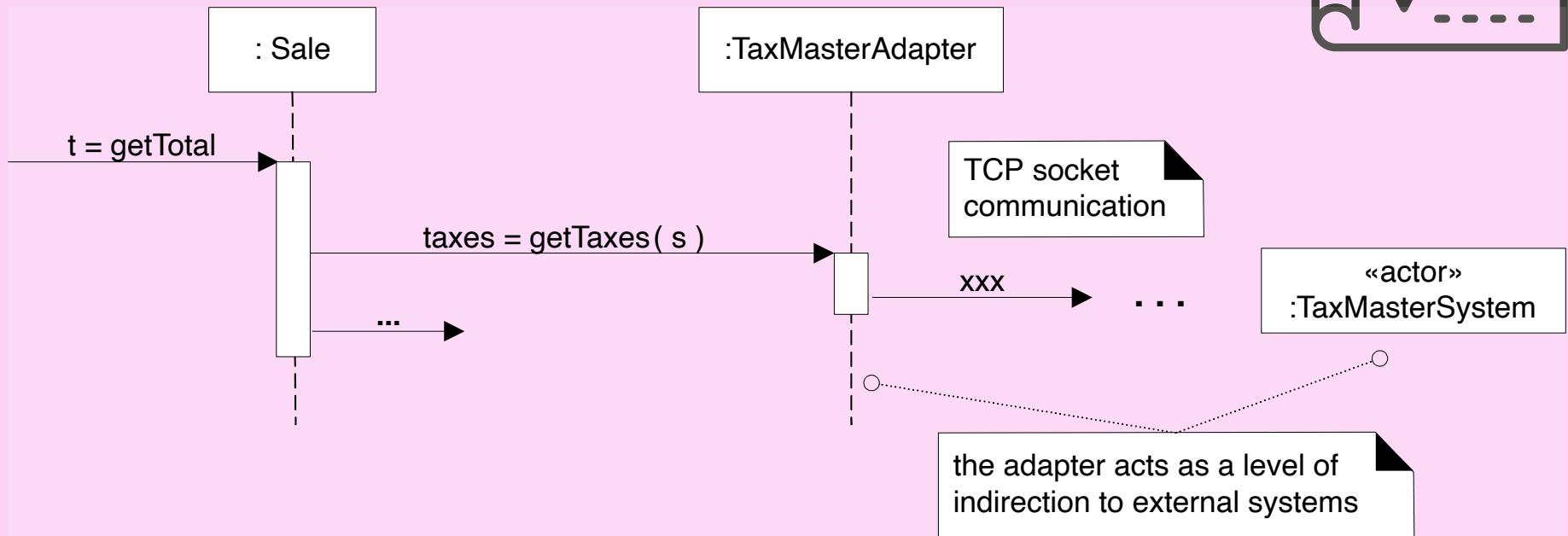
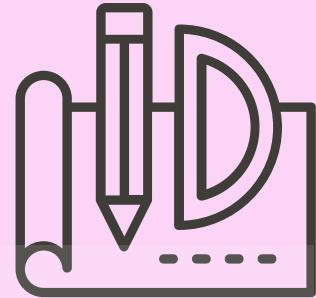
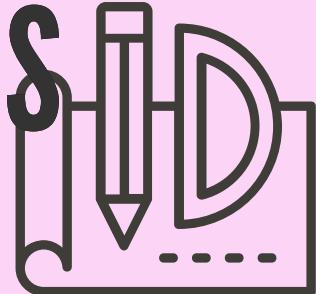


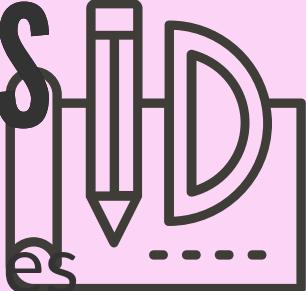
Fig. 25.10 (3e édition)

# PROTECTION DES VARIATIONS

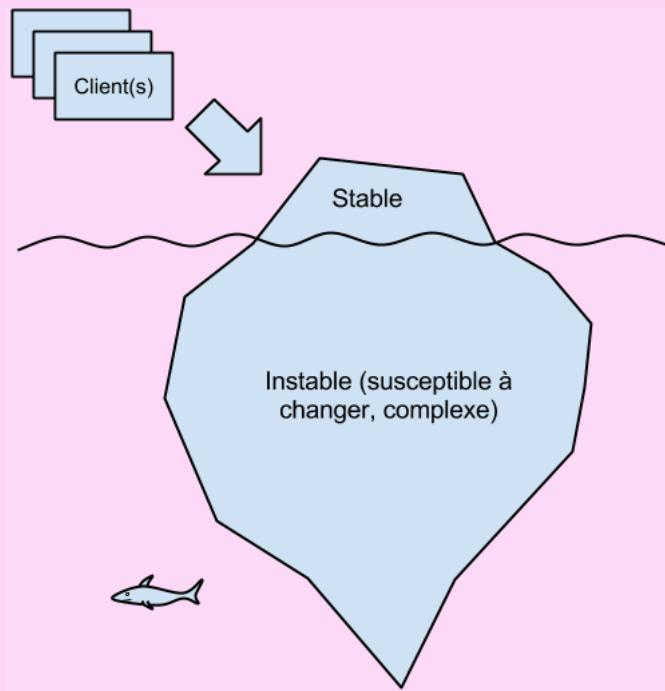


- Composants (classes, interfaces, ...)
  - variations (ou instabilités) de ces composants en affectent d'autres
- Éliminer ces effets indésirables
- Identifier des zones de variabilité

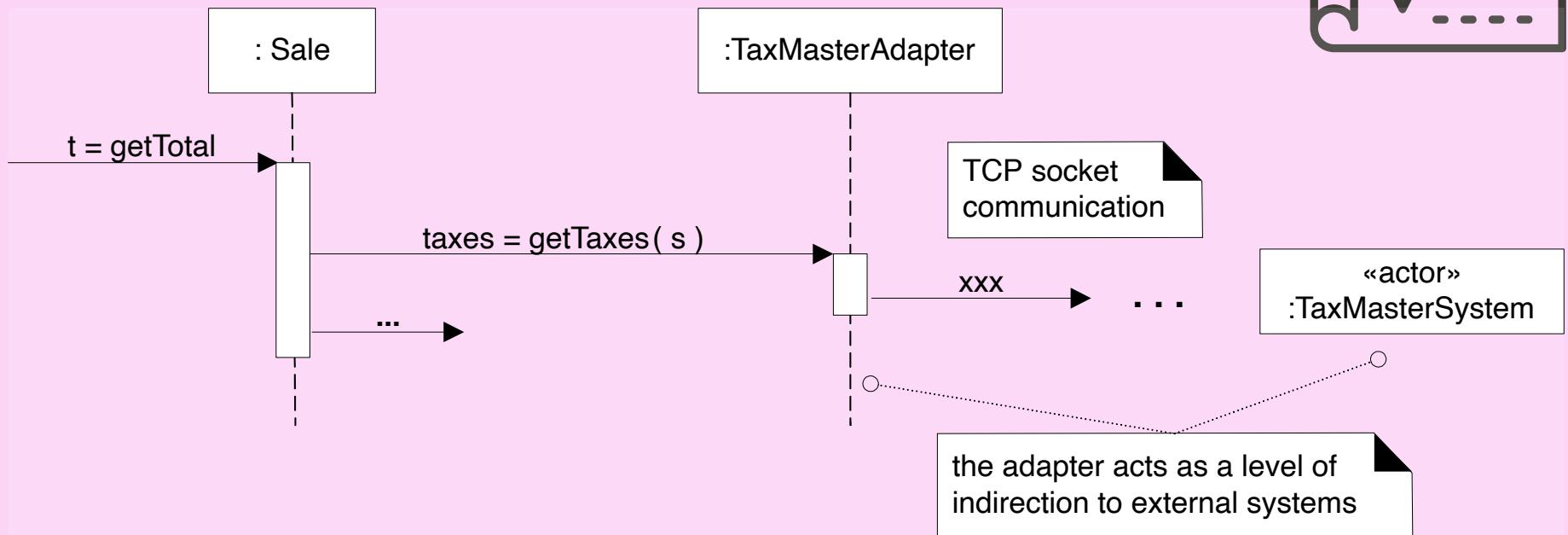
# PROTECTION DES VARIATIONS



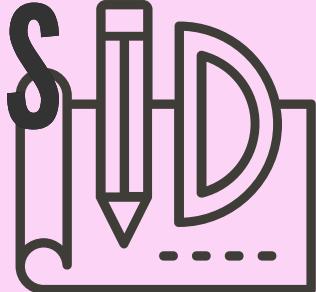
- Créer une interface stable (API) autour des composants instables



# PROTECTION DES VARIATIONS

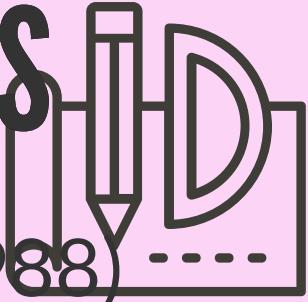


# PROTECTION DES VARIATIONS



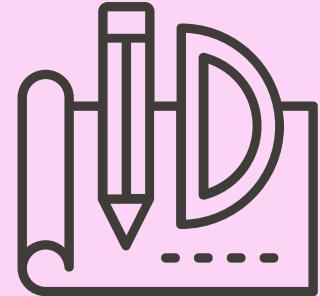
- Vieux principe du génie logiciel
- Principe de « Information hiding » (Parnas, 1972)
  - Encapsulation des données
  - Déterminer
    - choix difficiles du design
    - choix du design qui auront tendance à changer
  - Concevoir chaque module afin de cacher ces choix « variables »

# PROTECTION DES VARIATIONS



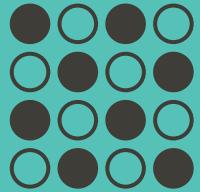
- Principe de « Open-closed » (Meyer, 1988)
- Composants devraient être
  - ouverts à l'extensibilité (Implémentations)
  - fermés à la modification (API)
- Ces contraintes paraissent contradictoires

# RÉSUMÉ



- D'autres patrons GRASP
  - Polymorphisme
  - Fabrication pure
  - Indirection
  - Protection des variations
- Vocabulaire
  - concepts de base pour les ingénieurs en logiciel
  - améliore la communication dans une équipe

# LOG210 SÉANCE #08



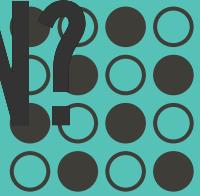
Created by Jonathan Li  
from the Noun Project

## ANALYSE ET CONCEPTION DE LOGICIELS

1. Administration
2. Équipe
3. Première étude de la conception
4. Outils UML
5. GRASP
6. GOF Révision ← S20203
7. Diagramme de package
8. TDD en pratique



# QU'EST CE QU'UN DESIGN PATTERN?

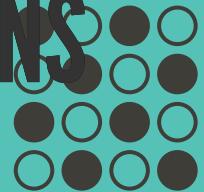


Created by Jonathan Li  
from the Noun Project

- Solution à un problème récurrent dans un contexte particulier

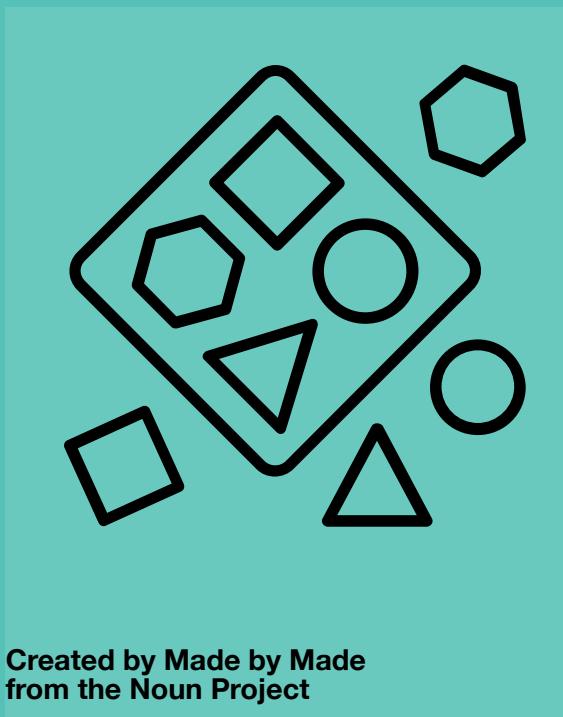


# TROIS DÉFIS POUR BIEN UTILISER LES PATTERNS

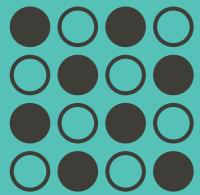


Created by Jonathan Li  
from the Noun Project

1. (Re)connaître le problème
2. Comprendre le patron (la solution)
3. Appliquer la solution



# 1.(RE)CONNAÎTRE LE PROBLÈME



Created by Jonathan Li  
from the Noun Project

- Comprendre le problème
- Exercice de « croyance » (on débute en OO)
- Certains problèmes sont plus « récurrents » que d'autres...
- Auteurs (GoF) avaient beaucoup d'expérience en OO



# 2. COMPRENDRE LE

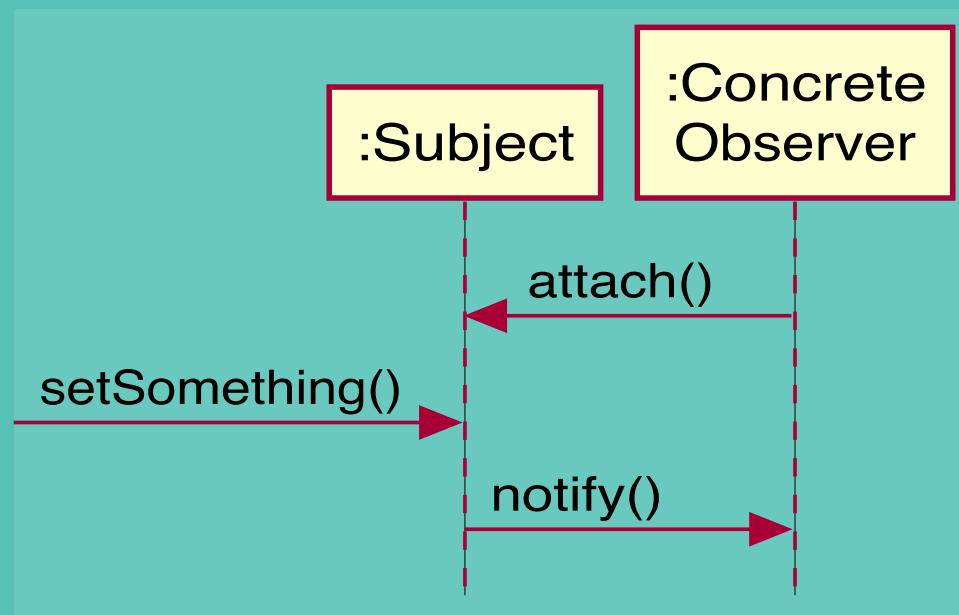
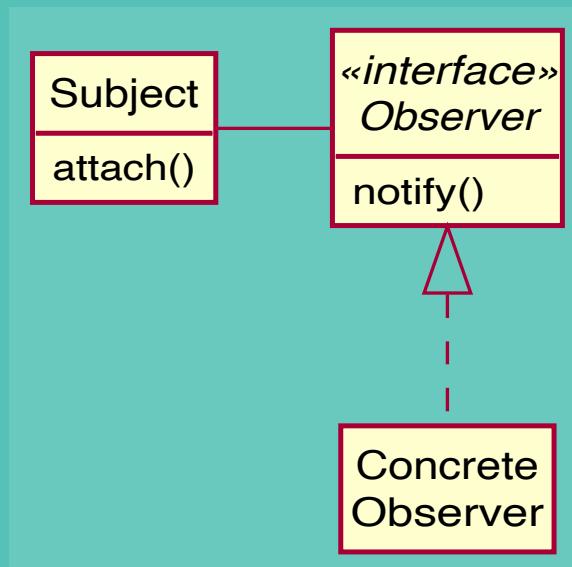
Created by Jonathan Li  
from the Noun Project

## PATRON

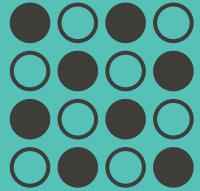
(LA SOLUTION)

- abstractions
- notation UML est importante
- aspects statique et dynamique





# 3. APPLIQUER LA SOLUTION

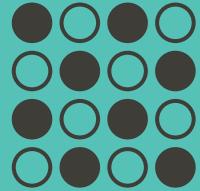


Created by Jonathan Li  
from the Noun Project

- Étape concrète
- Mise en contexte particulier



# 3. APPLIQUER LA SOLUTION



Created by Jonathan Li  
from the Noun Project

Nom dans le  
patron

Vrai nom (Bouton Swing)

Subject

JButton

Observer

ActionListener

ConcreteObserver

la classe implémentant  
l'interface ActionListener

attach()

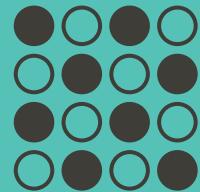
addActionListener()

notify()

actionPerformed()



# PATRONS VUS DANS LE LIVRE DE LOG121

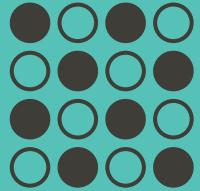


Created by Jonathan Li  
from the Noun Project

- Adaptateur
- Commande
- Composite
- Décorateur
- Itérateur
- Méthode fabrique
- Observateur
- **Proxy**
- Singleton
- Stratégie
- Template
- Visiteur



# PERSPECTIVE ZEN



**BEGINNER MIND**  
*« J'ai besoin d'un pattern pour Bonjour, le Monde »*



**INTERMEDIATE MIND**  
*« Peut-être un singleton va-t-il bien ici? »*



*« Un décorateur va naturellement ici. »*

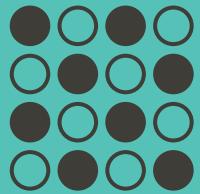


**ZEN MIND**

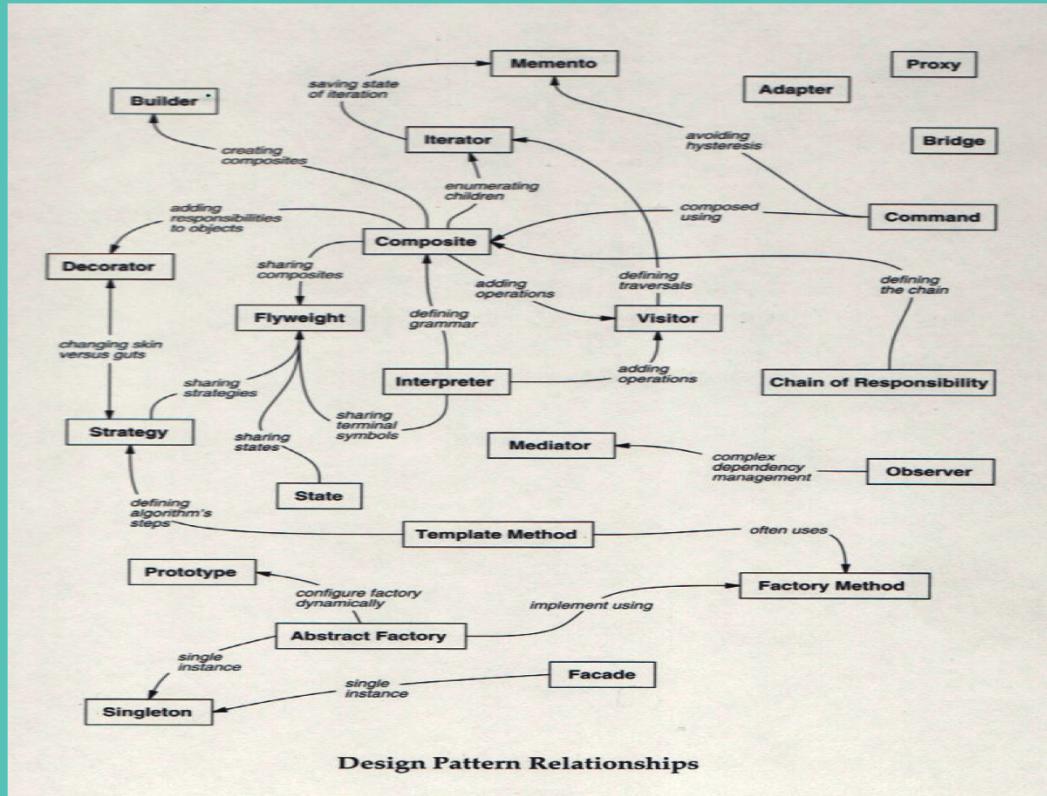
Created by Jonathan Li  
from the Head First Project

Freeman, Elisabeth, Eric Freeman,  
Bert Bates, Kathy Sierra, and Elisabeth  
Robson. 2004. *Head First Design  
Patterns*. 1st ed. O'Reilly Media.

# DESIGN PATTERN RELATIONSHIPS

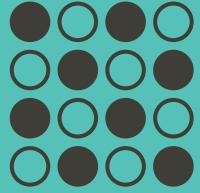


Created by Jonathan Li  
from the Noun Project



Ref: Design Patterns, Element of Reusable Object-oriented Software

# RÉSUMÉ

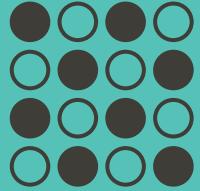


Created by Jonathan Li  
on the Coursera Project

- Reconnaître des problèmes récurrents est difficile faute d'expérience adéquate



# RÉSUMÉ

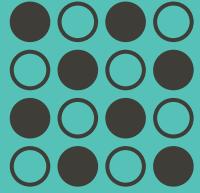


Created by Jonathan Li  
on the 'four' Project

- Reconnaître des problèmes récurrents est difficile faute d'expérience adéquate
- Expérimenter avec les patterns (dans une version du projet qui n'est pas critique)



# RÉSUMÉ

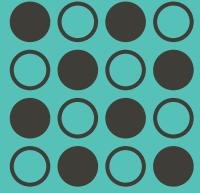


Created by Jonathan Li  
on the 'four' Project

- Reconnaître des problèmes récurrents est difficile faute d'expérience adéquate
- Expérimenter avec les patterns (dans une version du projet qui n'est pas critique)
- Un patron ne s'enlève pas facilement!



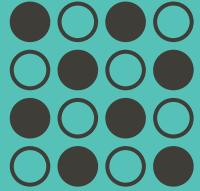
# RÉSUMÉ



Created by Jonathan Li  
on the 'four' Project

- Reconnaître des problèmes récurrents est difficile faute d'expérience adéquate
- Expérimenter avec les patterns (dans une version du projet qui n'est pas critique)
- Un patron ne s'enlève pas facilement!
- Discuter de leurs avantages et leurs inconvénients avec des collègues

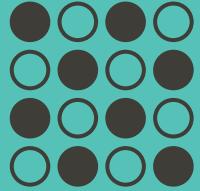
# RÉSUMÉ



Created by Jonathan Li  
on the 'four' Project

- Reconnaître des problèmes récurrents est difficile faute d'expérience adéquate
- Expérimenter avec les patterns (dans une version du projet qui n'est pas critique)
- Un patron ne s'enlève pas facilement!
- Discuter de leurs avantages et leurs inconvénients avec des collègues
- Vérifier après l'application du pattern que les bénéfices sont vraiment là

# RÉSUMÉ

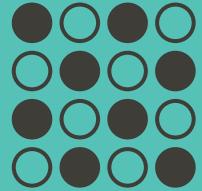


Created by Jonathan Li  
on the 'four' Project

- Reconnaître des problèmes récurrents est difficile faute d'expérience adéquate
- Expérimenter avec les patterns (dans une version du projet qui n'est pas critique)
- Un patron ne s'enlève pas facilement!
- Discuter de leurs avantages et leurs inconvénients avec des collègues
- Vérifier après l'application du pattern que les bénéfices sont vraiment là
  - p.ex. il est plus facile d'étendre le logiciel



# PATRON GOF LOG121

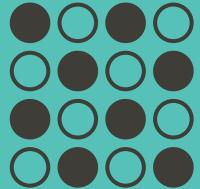


Created by Jonathan Li  
from the Noun Project

- Décorateur
- Itérateur
- Commande
- Façade
- Singleton
- Template method
- Factory method
- Proxy
- Observateur
- Visiteur



# PATRONS GOF



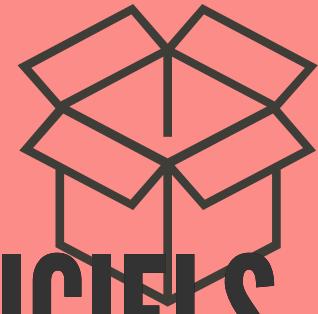
Created by Jonathan Li  
from the Noun Project

## Quiz patron GOF



# LOG210 SÉANCE #08

## ANALYSE ET CONCEPTION DE LOGICIELS



1. Administration
2. Équipe
3. Première étude de la conception
4. Outils UML
5. GRASP
6. GOF Révision
7. Diagramme de package ← S20203
8. TDD en pratique S20203

# PACKAGING?



Created by Christophe Argelès  
from the Box Project

# PACKAGING?

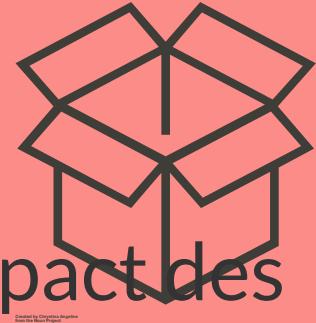
## ATTRIBUT DE QUALITÉ



Created by Chaitanya Agrawal  
from the Noun Project

- Performance
- Disponibilité
- Testabilité
- Interopérabilité
- Sécurité
- Usabilité
- Modifiabilité

# OBJECTIFS



- Organiser les packages pour réduire l'impact des modifications
- Apprendre d'autres notations des structures de packages UML

# PRINCIPES

- Organiser les packages par **cohésion**
- Packager une **famille d'interface**
- Créer un package par **tâche** et par **goupe de classes instables** (Branches)
- Les plus responsables sont les plus stables
- Factoriser les type indépendants
- Utiliser des **fabrications** pour limiter la dépendance aux packages concrets
- Pas de cycles dans les packages **DSM**



# GROUPEMENT PAR COHÉSION FONCTIONNELLE

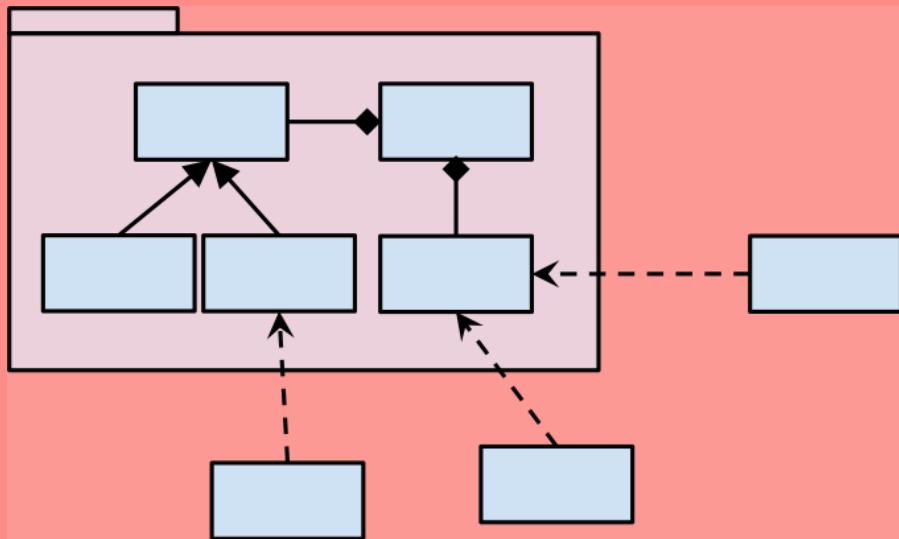


- On groupe les types qui sont fortement apparentés, parce qu'ils participent à une finalité, un service, des collaborations, une politique et une fonction qu'ils ont en commun.
- P.ex. le package “Tarification” contient des classes liées à la tarification des produits.
- C'est intuitif.

# GROUPEMENT PAR COHÉSION RELATIONNELLE



- Mettre dans un package des classes qui se sont fortement couplées les unes aux autres.
- $CR = (\text{NombreDeRelationsInternes} + 1) / \text{NombreDeTypes}$



- GOF Facade pour améliorer la cohésion relationnelle avec les packages externes

# COHÉSION RELATIONNELLE - EXEMPLE



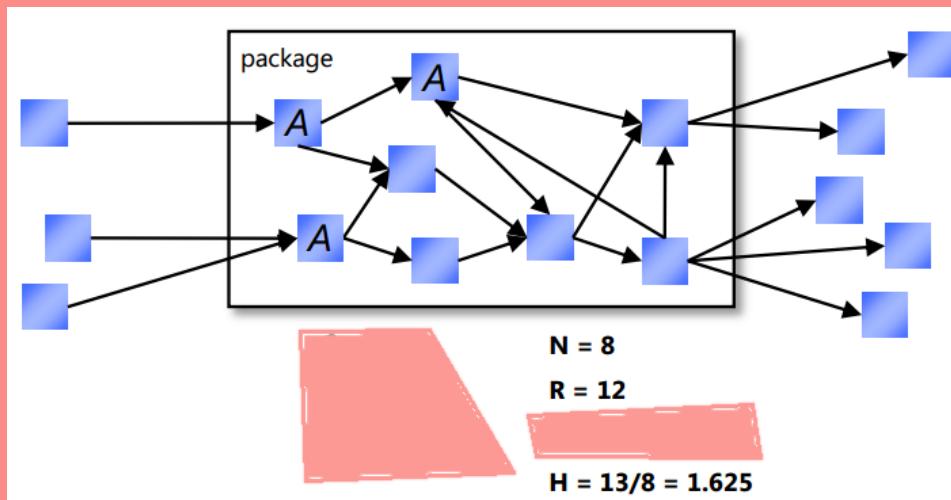
## cohesion

**Relational Cohesion (H):** average number of internal relationships per type:

$$H = (R + 1) / N, \text{ where}$$

**R** = number of type relationships internal to the package,  
and

**N** = number of types in the package.



ndepend: métrique de cohésion

# PACKAGER UNE FAMILLE D'INTERFACES

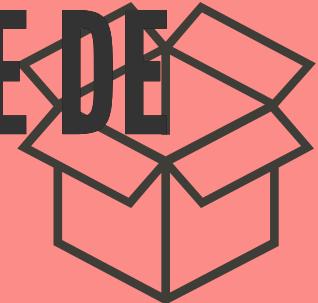


- Placez une famille d'interfaces fonctionnellement liées dans un package à part
- Cela vaut pour les familles de trois interfaces ou plus
- Exemple: javax.ejb comprend au moins douze

```
- EJBContext
- EJBHome
- EJBLocalHome
- EJBLocalObject
- EJBMetaData
- EJBObject
- EnterpriseBean
- EntityBean
- EntityContext
- Handle
- HomeHandle
- MessageDrivenBean
- MessageDrivenContext
- SessionBean
- SessionContext
- SessionSynchronization
- TimedObject
- Timer
- TimerHandle
- TimerService
```



# CRÉEZ UN PACKAGE PAR TÂCHE DE LIVRAISON



Created by Christopher Arguello  
from the Noun Project

- Les packages sont l'unité de base de développement et de livraison (on ne développe rarement qu'une seule classe)
- Créer un package par tâche de livraison



# ... ET PAR GROUPE DE CLASSES INSTABLES [2/2]



- Si un ensemble de classes dans un package a une tendance à évoluer ensemble, p.ex. on remarque qu'à chaque itération, c'est toujours les mêmes 5 ou 10 classes dans un package qui ont été modifiées.
- Exemple concret – chaque fois que l'on change le schéma de la BD il y a des changements dans plusieurs classes, surtout les appels à la méthode “sqlquery()”
  - Créez un package dans le package (sous package) pour isoler les classes ayant une tendance à changer ensemble
  - Notez que la tendance de ce genre de changement n'est pas visible dans les premières itérations.

# LES PLUS RESPONSABLES SONT LES PLUS STABLES

Les packages « les plus responsables » engendrent le plus de dépendances et devraient être plus stables.

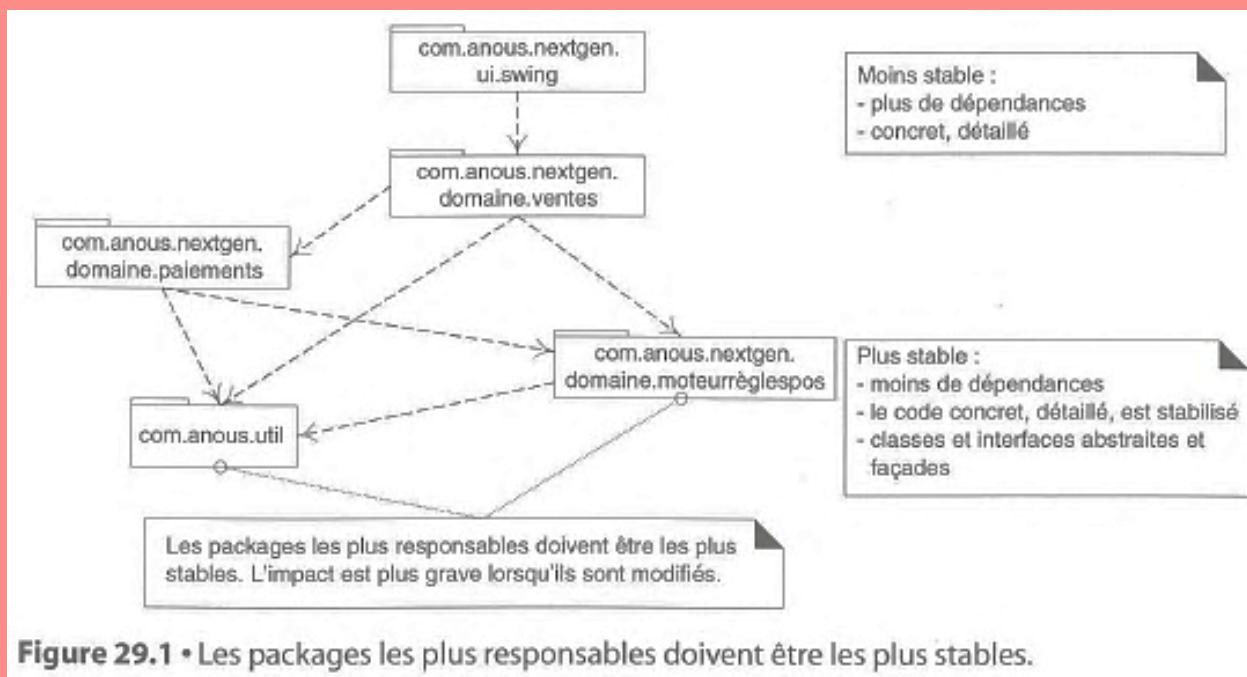
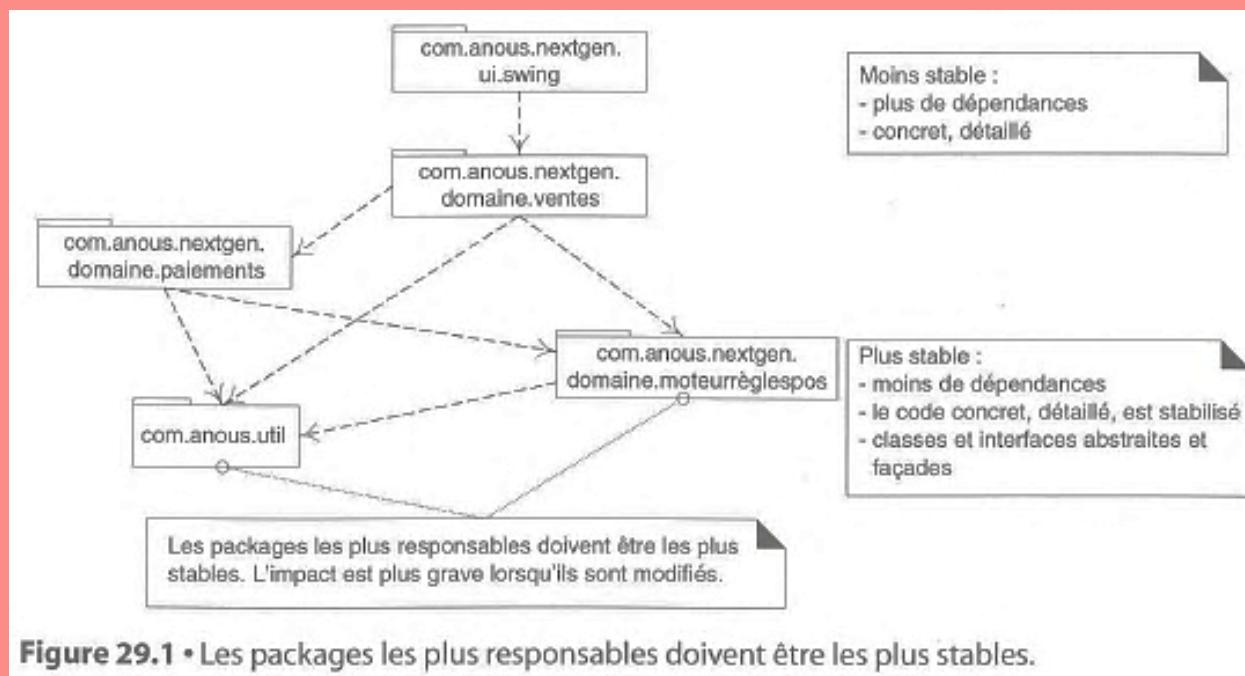


Figure 29.1 • Les packages les plus responsables doivent être les plus stables.

# LES PLUS RESPONSABLES SONT LES PLUS STABLES

Les packages « les plus responsables » engendrent le plus de dépendances et devraient être plus stables.

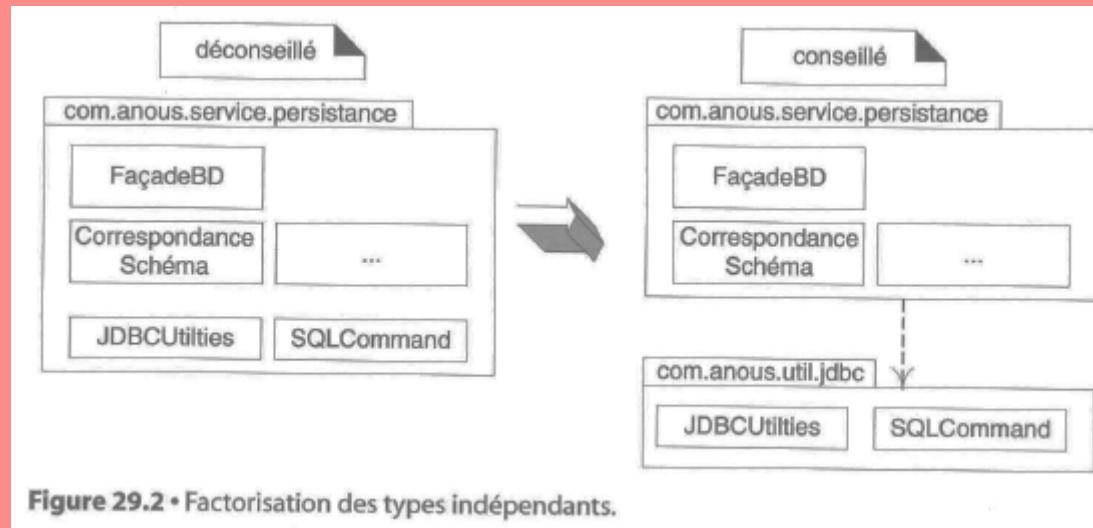


## Quel GRASP pour rendre un package plus stable?

# FACTORIZER LES TYPES



Organiser les types qui peuvent être utilisés indépendamment ou dans des contextes différents en packages distincts.



# FABRIQUES POUR DÉCOUPLER DES CLASSES CONCRÈTES



Renforcer la stabilité des packages consiste à réduire leur dépendance à des classes concrètes qui appartiennent à d'autres packages

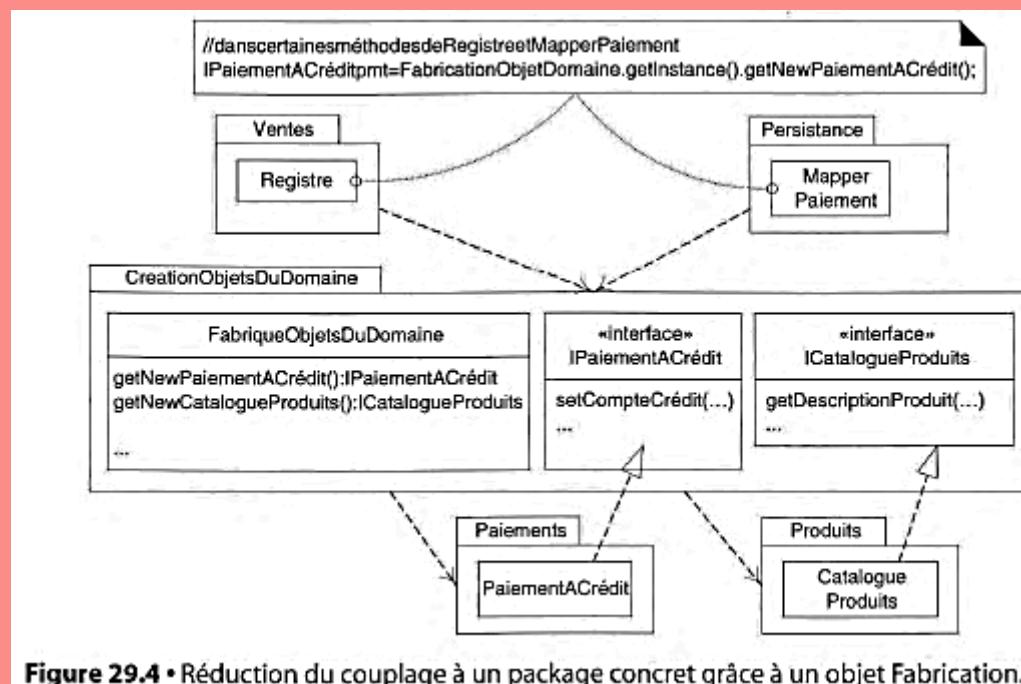


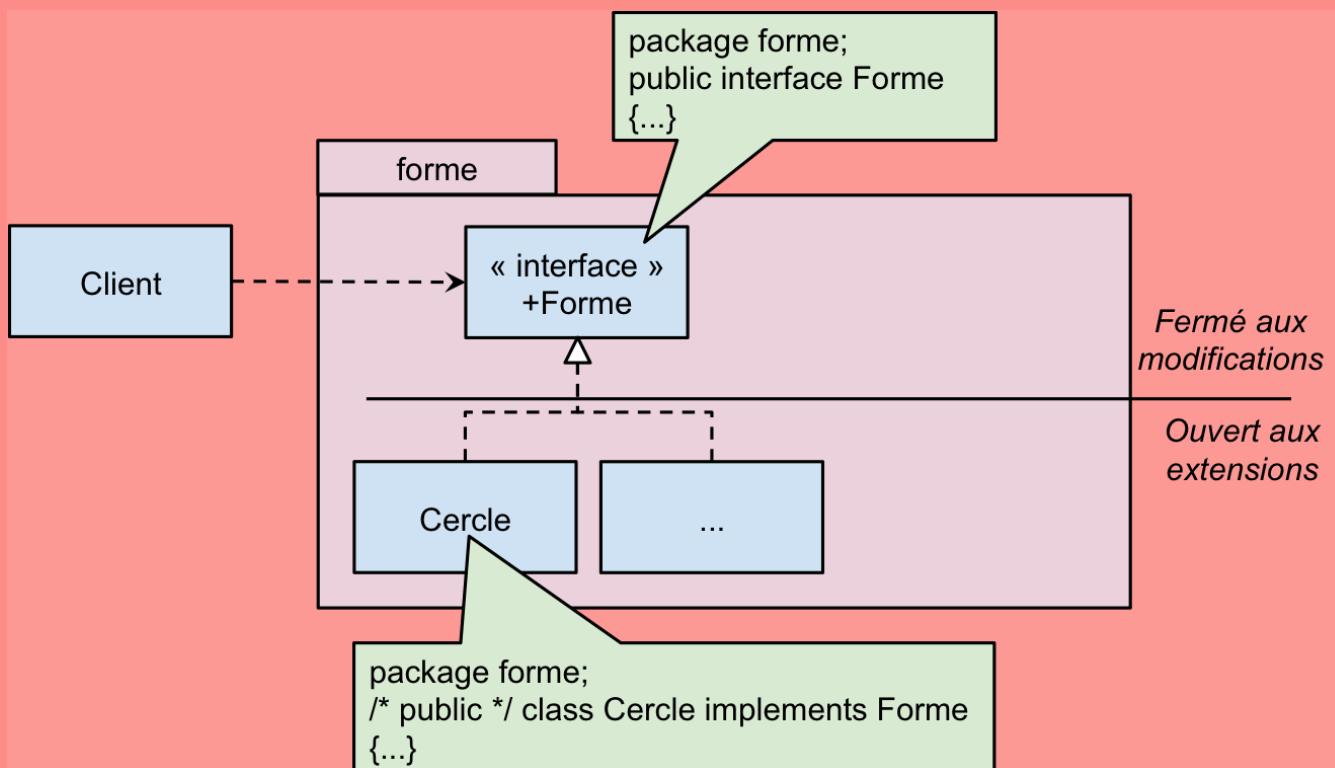
Figure 29.4 • Réduction du couplage à un package concret grâce à un objet Fabrication.



# VISIBILITÉ ET ENCAPSULATION DANS LES PACKAGES



## Organiser les classes dans un package avec la bonne visibilité

Created by Christophe Argotin  
from the Blue Project

# VISIBILITÉ ET ENCAPSULATION DANS LES PACKAGES



- Bloquer l'accès à un attribut ou une méthode d'une classe avec `private`
- Bloquer l'accès à une classe dans un package?
  - Déclarer la classe `class X { ... }` (package protection, sans mot clé)
  - GRASP Protection des variations
    - Client ne devrait pas accéder aux implémentations de Forme.

# VISIBILITÉ ET ENCAPSULATION DANS LES PACKAGES



- Cette visibilité empêche les transgressions de l'encapsulation

The screenshot shows a Java code editor with the following code:

```
package client;

public class Client {
    /**
     * @param args
     */
    public static void main(String[] args) {
        forme.Forme maForme = new forme.Cercle();
        maForme.dessiner();
    }
}
```

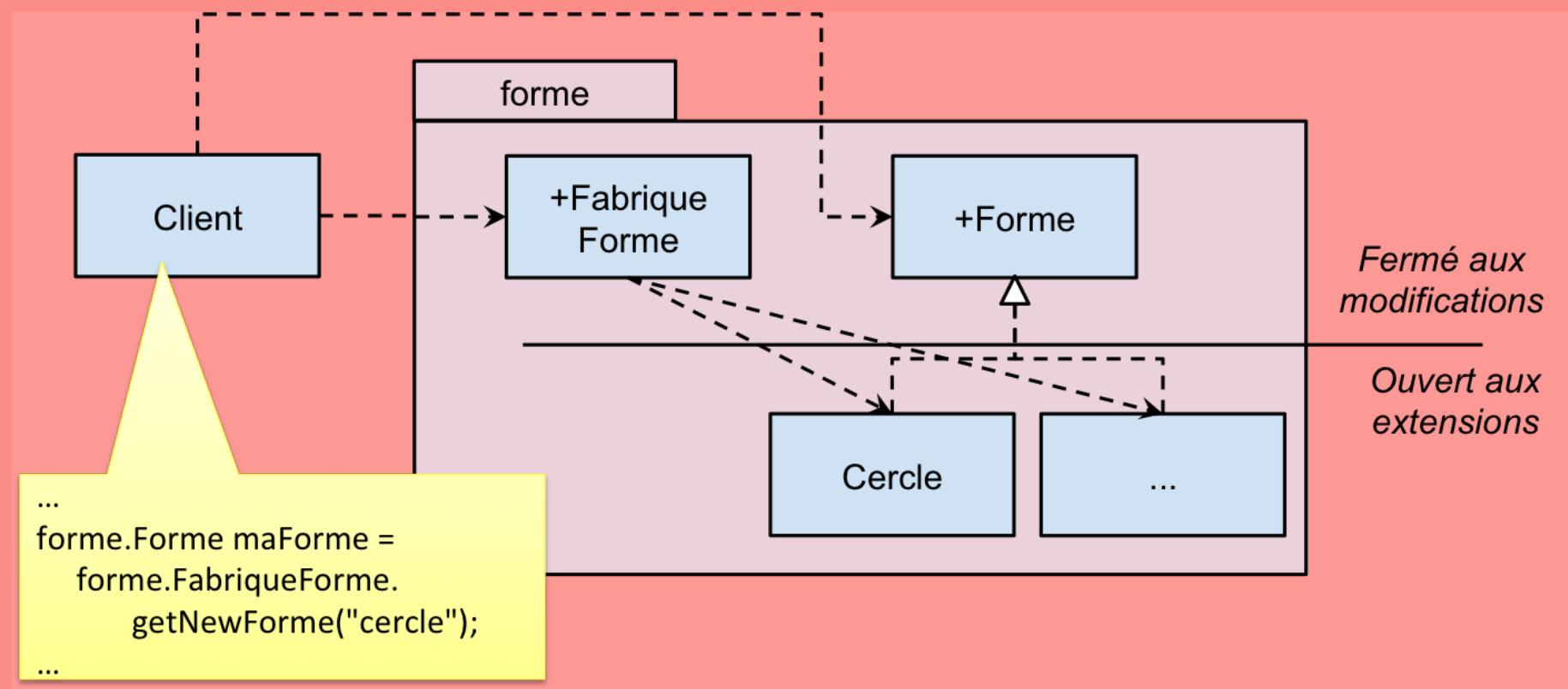
A code analysis tooltip is displayed over the line `forme.Forme maForme = new forme.Cercle();`. The tooltip says:

**The type forme.Cercle is not visible**  
1 quick fix available:  
Change visibility of 'Cercle' to 'public'

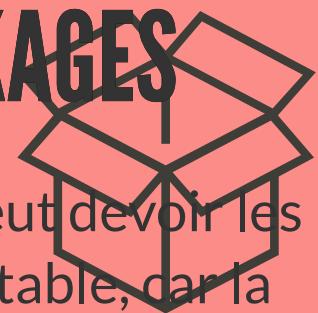
A large red circular 'no' symbol is overlaid on the bottom right of the tooltip.

# COMMENT INSTANCIER LES CLASSES INVISIBLES?

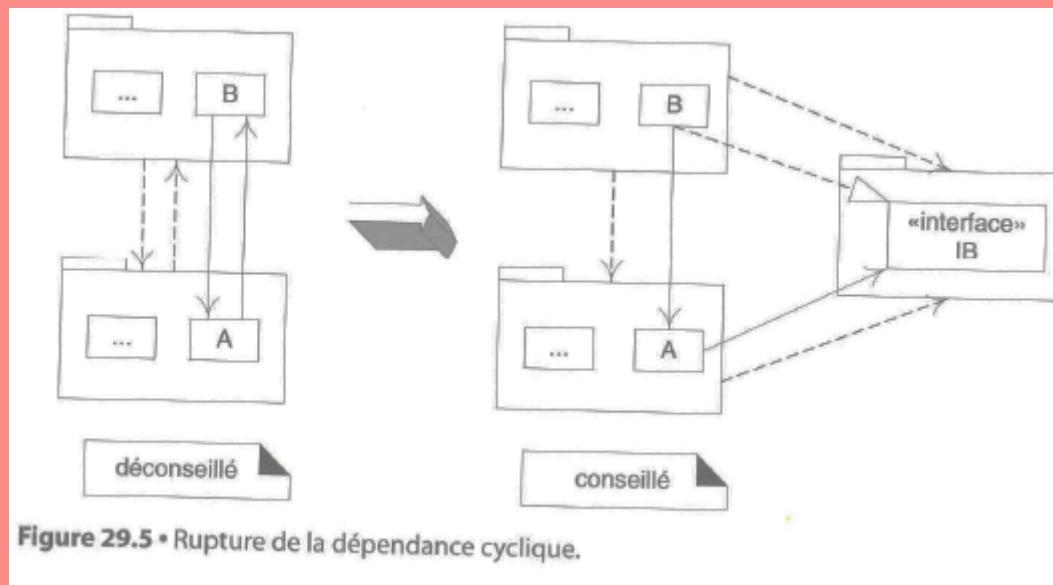
- FabriqueForme permet à Client d'instancier les Formes concrètes, bien qu'elles lui soient invisibles



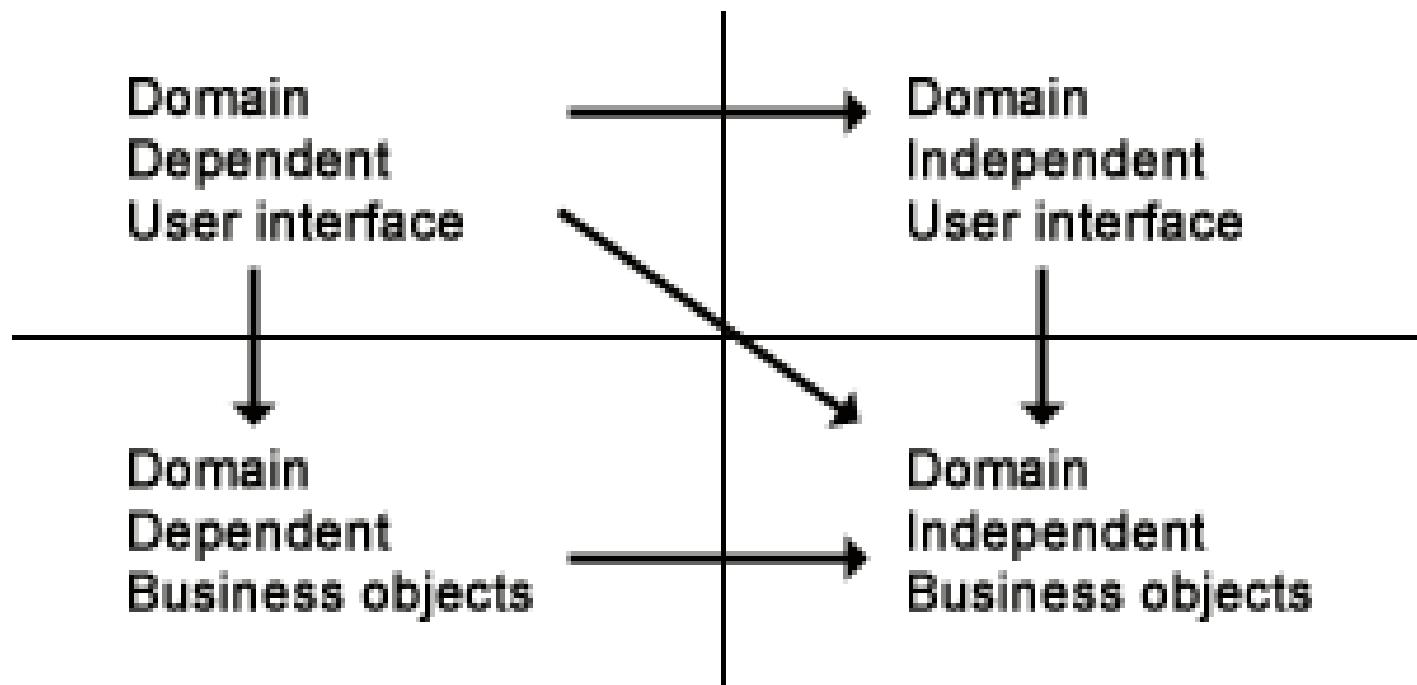
# PRINCIPE: PAS DE CYCLES DANS LES PACKAGES



- Si un groupe de packages a une dépendance cyclique, on peut devoir les traiter comme un seul grand package. Cela n'est pas souhaitable, car la livraison de packages volumineux (ou d'agrégats de packages) aggrave le risque d'effets néfastes.
- Factoriser les types participant au cycle dans un package plus petit.
- Rompre le cycle à l'aide d'une interface.



# COMBINAISON DES PATRONS DE PACKAGING



ref: Commonly used architectural patterns in Java applications

# QU'EST-CE QU'UNE COUCHE?



- Groupement à forte granularité de classes, packages et sous-systèmes
- Possède un thème « cohésif »
- Une couche plus « haute » fait appel aux couches plus « basses », mais pas l'inverse
- Couches OO typiques
  - Présentation (IHM)
  - Logique applicative (objets du domaine)
  - Services techniques (persistance, journalisation, etc.)

# CONCEVOIR UNE ARCHITECTURE EN COUCHES



- Séparation nette et cohésive
- Collaboration et couplage vont du haut vers le bas

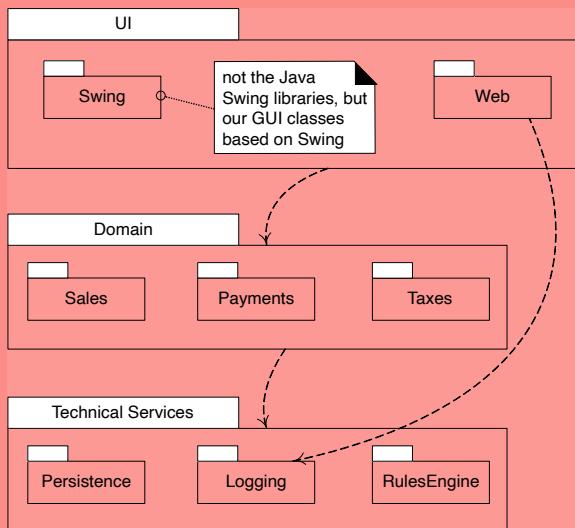


fig F12.2, A13.2

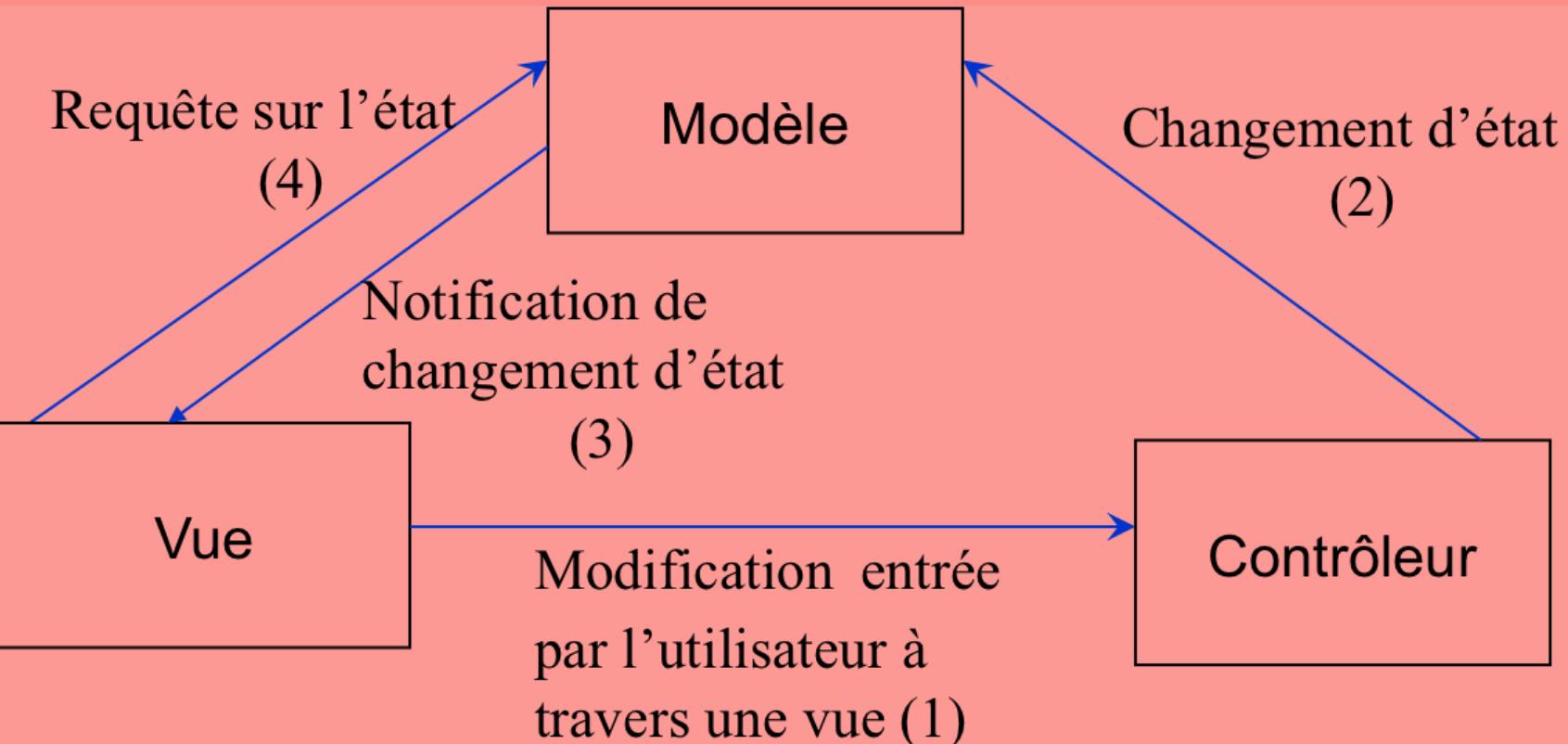
- Séparation Modèle-Vue est un cas simple



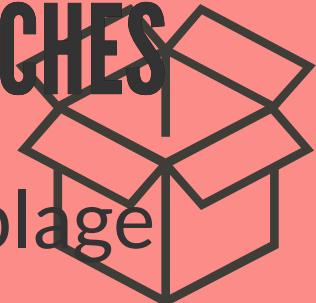
# ARCHITECTURE - MODÈLE / VUE / CONTRÔLEUR



- Vues/contrôleurs modifient le modèle
- Modèle informe les vues des changements
- Vues se mettent à jour de nouveau en conséquence

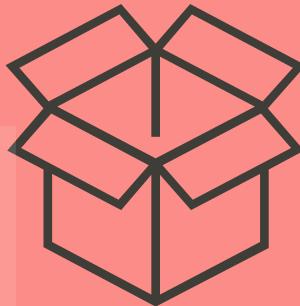
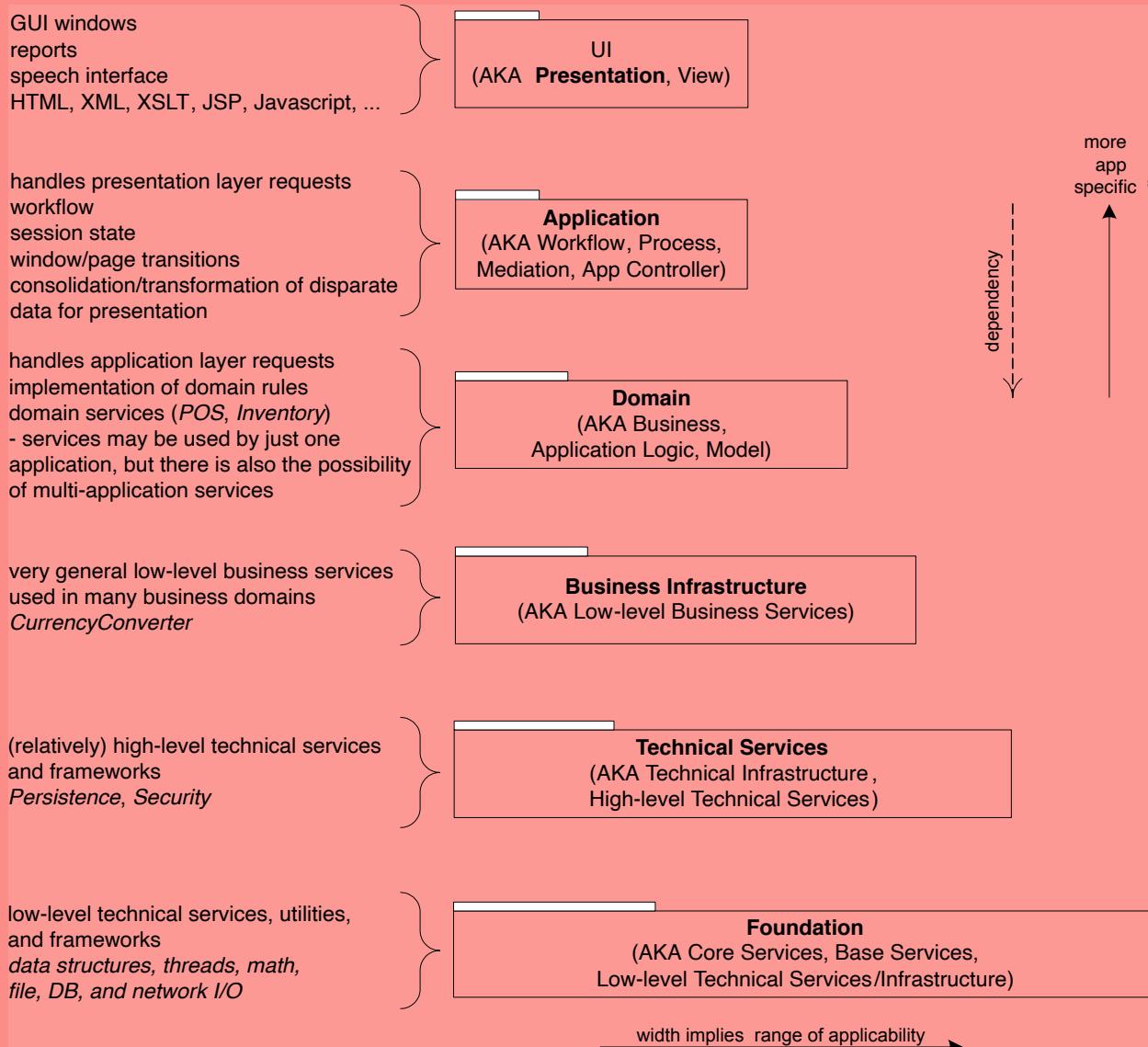


# AVANTAGES DE LA MODÉLISATION EN COUCHES



- Séparation de fonctions/services -> couplage réduit, cohésion améliorée
- La complexité est encapsulée (décomposable)
- On peut remplacer certaines couches
- Les couches basses contiennent des fonctions réutilisables
- Certaines couches (Domaine) peuvent être distribuées
- Segmentation logique facilite développement en équipe

# ARCHITECTURE MULTI-COUCHES



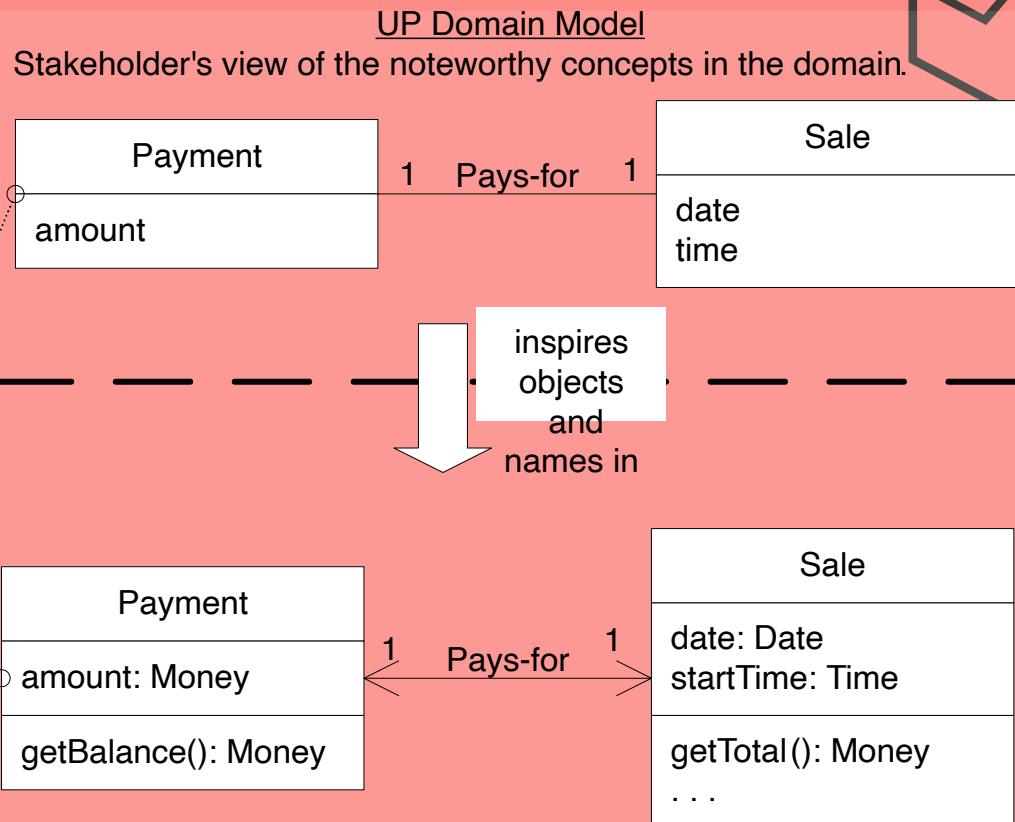
# RELATION ENTRE COUCHE DOMAINE ET DCE



A Payment in the Domain Model is a concept, but a Payment in the Design Model is a software class. They are not the same thing, but the former *inspired* the naming and definition of the latter.

This reduces the representational gap.

This is one of the big ideas in object technology.



## Domain layer of the architecture in the UP Design Model

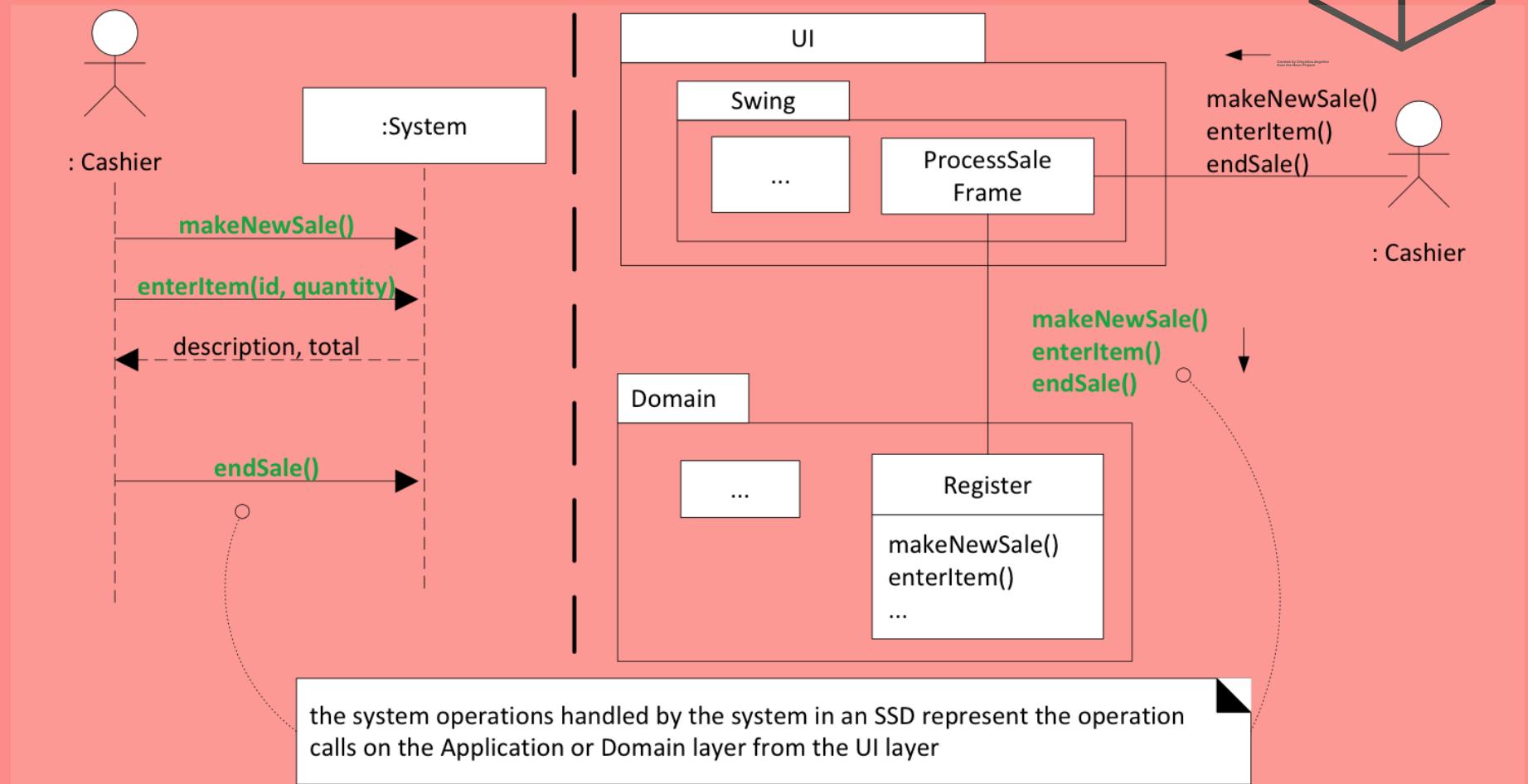
The object-oriented developer has taken inspiration from the real world domain in creating software classes.

Therefore, the representational gap between how stakeholders conceive the domain, and its representation in software, has been lowered.

# RELATION ENTRE DSS ET COUCHES



Created by Christophe Argenton  
From the book Project



# INFLUENCE DES ARTEFACTS DU PROCESSUS UNIFIÉ

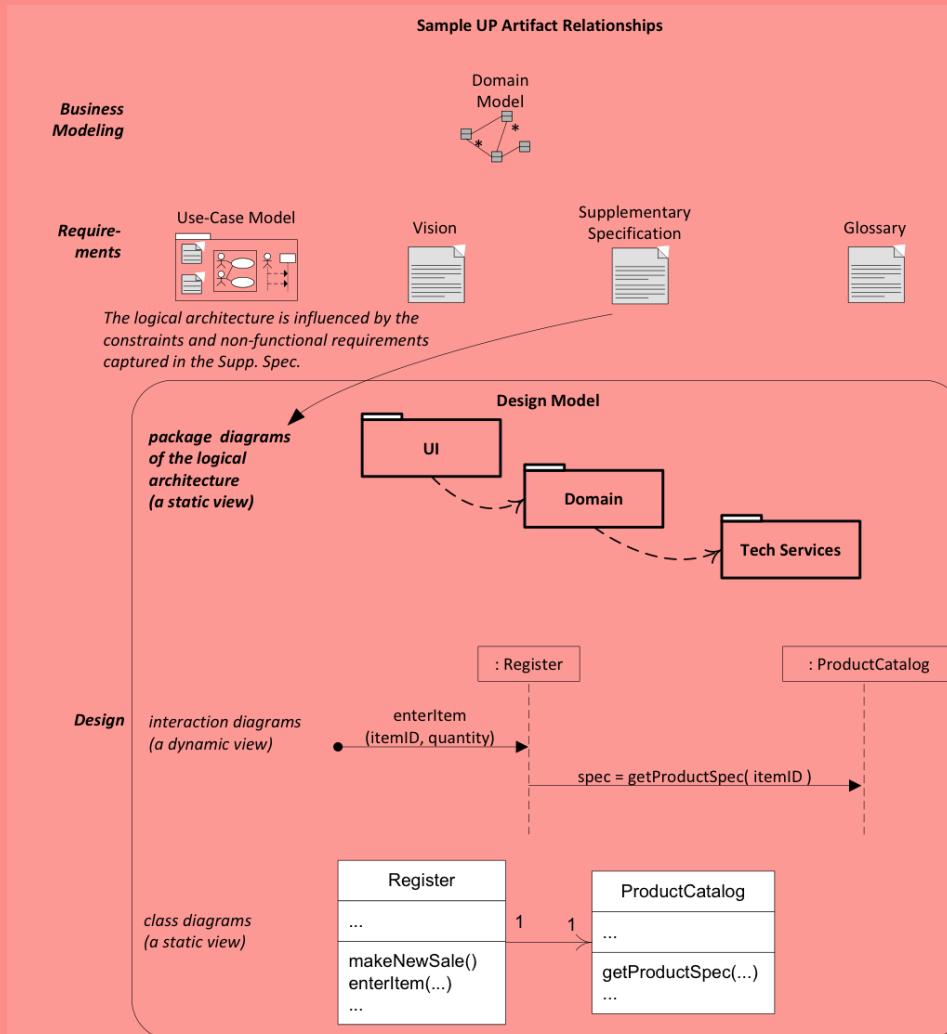


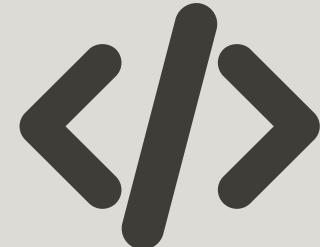
fig A13.1, F12.1

# RÉSUMÉ



- Architecture logique est importante pour les conceptions de systèmes complexes
- Modèle en couches est populaire
  - Extension du principe de séparation Modèle-Vue
  - Java est organisé en couches
  - Android aussi
- DSS relie la couche présentation avec la couche du domaine à travers les opérations système

# LOG210 SÉANCE #08



## ANALYSE ET CONCEPTION DE LOGICIELS

1. Administration
2. Équipe
3. Première étude de la conception
4. Outils UML
5. GRASP
6. GOF Révision
7. Diagramme de package
8. TDD en pratique 



# TDD



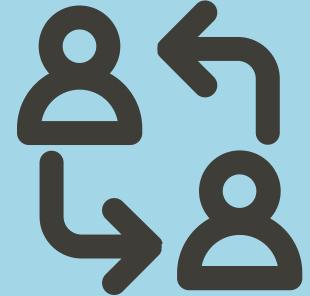
## 1. Kata TDD avec TypeScript

- Développer un fonction permettant d'extraire les mots uniques d'une phrase
- Développer une fonction permettant de faire l'intersetion des mots de deux phrases



# SÉANCE #08

## RÉTROACTION: PAGE D'UNE MINUTE



Created by Prithvi  
from the Noun Project

1. Quels sont les deux [trois, quatre, cinq] plus importants [utiles, significatives, surprenantes, dérangeantes] choses que vous avez apprises au cours de cette session?
2. Quelle (s) question (s) reste (s) en tête dans votre esprit?
3. Y a-t-il quelque chose que tu n'as pas compris?

<https://1drv.ms/u/s!An6-F73ulxAOhVyiCB46jTeINVLS>

