

# LOG210 SÉANCE #05

## ANALYSE ET CONCEPTION DE LOGICIELS



1. **Equipe** - Culture d'équipe ← S20203
2. **MDD** - Modèle du domaine - Révision
3. **TypeScript** - Exemples TypeScript (SGB)
4. **Complexité** - Inhérente et accidentelle
5. **DSS** - Diagramme de séquence système
6. **Contrats** - Contrats pour réaliser les RDCU
7. **RDCU** - Exercice avancé
8. **DCL** - Diagramme de classes logiciel
9. **Exercice** - Exercice

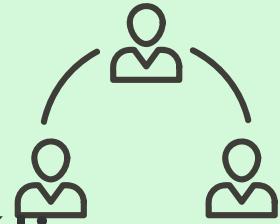
# CULTURE D'ÉQUIPE



- Qu'est-ce qu'une culture d'équipe?
  - Ensemble de valeurs, d'objectifs, d'expériences qui est unique à chaque équipe.
  - Éléments « codage » : revues de code, développement piloté par les tests, documentation de la conception, etc.
  - Éléments sociaux: sushi à midi, ou un 5 à 7 le vendredi, etc.
- Qu'elle soit bonne ou mauvaise, la culture existera
- Le leader ne décide pas la culture; il s'en occupe.



# CULTURE FORTE D'ÉQUIPE



- Culture ouverte au changement qui l'améliore, mais résistant à un changement radical qui lui fait mal.
- Celle qui concentre l'effort sur **la livraison de logiciel génial** est la mieux réussite.
- Efforts pour souder une équipe ne mènent pas toujours à la productivité:
  - faire la fête, surenchère de programmation, faire des rencontres

# CULTURE FORTE D'ÉQUIPE



- Une culture forte sera une culture d'auto-selection
- Culture de code propre et facile à maintenir attirera des développeurs appréciant ces valeurs...
- Culture d'agressivité, d'initiation, de dérapages verbaux, etc. attirera des développeurs appréciant ces valeurs...

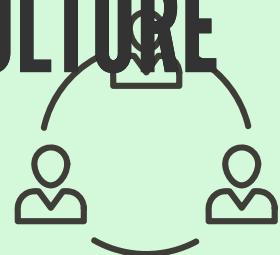
# ÉLÉMENTS POUR LES CULTURES D'ÉQUIPE RÉUSSITES



- Énoncé de mission d'équipe
- Culture de HRC
- favorise une participation des coéquipiers extrovertis et introvertis
- team.égo > coéquipier[i].égo
- La critique constructive
  - facile à recevoir
  - difficile à donner : ne pas oublier « respect »
- Bonne communication est essentielle
- Moyens de communication diversifiés
- Listes de diffusion
- Documentation de design
- Communication synchrone (rencontres) vs asynchrone (courriel)



# COMPORTEMENTS MENAÇANT UNE BONNE CULTURE

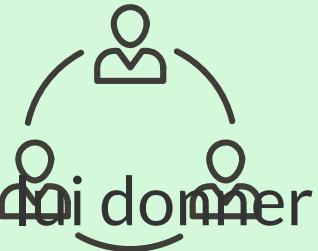


- Ne pas respecter le temps des autres
- Ne pas respecter une décision prise par l'équipe
- Ne pas écouter ou respecter les autres
- Ne pas faire de compromis
- Être perfectionniste
- Être provocateur (troll) / Répondre aux provocateurs (trolls)
- Devenir trop affectif

*L'attention et la concentration sont primordiales.*

# COMMENT AGIR FACE À CES COMPORTEMENTS

- Chercher des faits dans le drame
- Si quelqu'un se plaint, même avec trop d'émotion, lui donner le bénéfice du doute et chercher les causes (malgré le manque de respect, etc.)
- Amener la discussion sur un plan technique si possible.
- Souvent il y a des choses à améliorer dans la situation.
- La gentillesse peut chasser les trolls en fin de compte...





# COMMENT AGIR FACE À CES COMPORTEMENTS



- Se concentrer sur l'objectif à long terme
- Un témoin de comportement délétère se demande:
  - Malgré la perte de concentration de l'équipe à court terme, une résolution du drame sera-t-elle bénéfique à l'équipe à long terme?
  - Est-ce que la situation se résoudra d'elle-même?
- Si la réponse est « non » à une de ces questions, mettre fin au comportement immédiatement (sans résolution).



# COMMENT AGIR FACE À CES COMPORTEMENTS



- Savoir quand abandonner
- Parfois le comportement d'un coéquipier ne s'améliore pas malgré beaucoup d'efforts. Il faut dans ce cas l'isoler de l'équipe.

*Avant de demander un changement d'équipe, il faudra avoir essayé une approche HRC et être en mesure de l'expliquer à l'enseignant.*

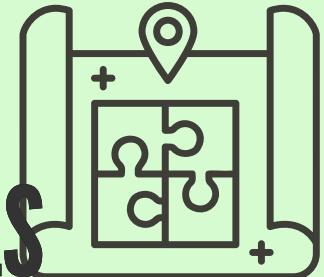
# CULTURE D'ÉQUIPE



- La plupart des gens ne sont pas des imbéciles!
- Cependant, il est naïf de penser aux gens comme « bons » ou « mauvais ».
- La malice qui menace une bonne culture d'équipe est souvent expliquée par l'ignorance, le besoin d'être reconnu, ou un manque d'empathie...
- Il faut toujours être tolérant *envers les gens*, mais *ne pas tolérer les comportements* qui nuisent à une bonne culture d'équipe (selon la norme HRC).

# LOG210 SÉANCE #05

## ANALYSE ET CONCEPTION DE LOGICIELS



1. **Equipe** - Culture d'équipe
2. **MDD** - Modèle du domaine - Révision  S20203
3. **TypeScript** - Exemples TypeScript (SGB)
4. **Complexité** - Inhérente et accidentelle
5. **DSS** - Diagramme de séquence système
6. **Contrats** - Contrats pour réaliser les RDCU
7. **RDCU** - Exercice avancé
8. **DCL** - Diagramme de classes logiciel
9. **Exercice** - Exercice

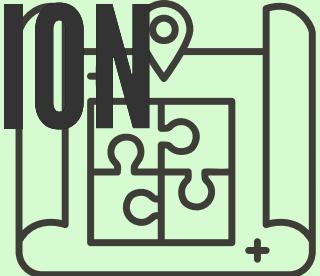


# MDD?



- Agrégation vs composition?
- Comment interpréter une multiplicité de plusieurs

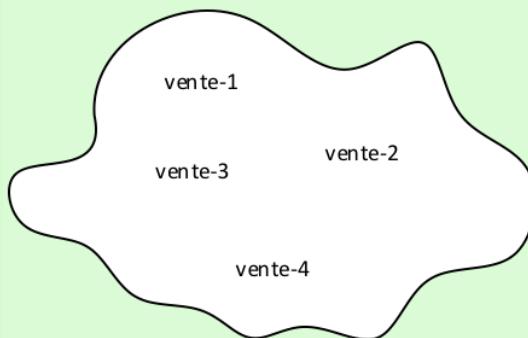
# SYMBOLE, INTENSION, EXTENSION



symbol du concept

"Une vente représente  
l'événement d'une transaction  
d'achat. Elle a une date et une  
heure."

intension du concept



extension du concept

# LOG210 SÉANCE #05



## ANALYSE ET CONCEPTION DE LOGICIELS

1. **Equipe** - Culture d'équipe
2. **MDD** - Modèle du domaine - Révision
3. **TypeScript** - Exemples TypeScript (SGB) 
4. **Complexité** - Inhérente et accidentelle
5. **DSS** - Diagramme de séquence système
6. **Contrats** - Contrats pour réaliser les RDCU
7. **RDCU** - Exercice avancé
8. **DCL** - Diagramme de classes logiciel
- ≡ 9. **Exercice** - Exercice

# EXEMPLES TYPESCRIPT

Dépôt github

<https://github.com/profcfuhrmanets/exemples-ts/>



# ARROW FUNCTION /> (JAVASCRIPT)

src/arrow\_function.ts



# ARRAY.PROTOTYPE.MAP ()

src/array\_map\_arrow.ts



# TABLEAU ASSOCIATIF

## MAP() (JAVASCRIPT)

src/tableau\_associatif.ts



# ACCÈS SGB </>

src/get\_students\_fetch.ts



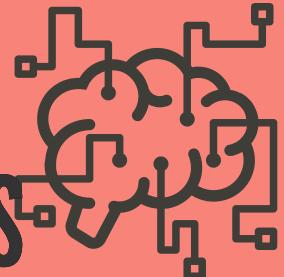


# DEUX “MAP” </>

Le sens est différent:

- Tableau associatif Map<> ( )
  - <https://howtodoinjava.com/typescript/maps/>
- Array.prototype.map( )
  - [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Array/map](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map)

## ANALYSE ET CONCEPTION DE LOGICIELS

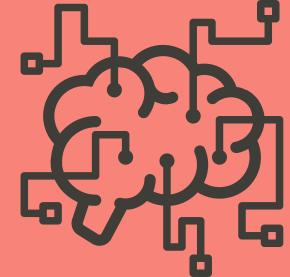


Created by Weltenraiser  
from the Noun Project

1. **Equipe** - Culture d'équipe
2. **MDD** - Modèle du domaine - Révision
3. **TypeScript** - Exemples TypeScript (SGB)
4. **Complexité** - Inhérente et accidentelle  **S20203**
5. **DSS** - Diagramme de séquence système
6. **Contrats** - Contrats pour réaliser les RDCU
7. **RDCU** - Exercice avancé
8. **DCL** - Diagramme de classes logiciel
9. **Exercice** - Exercice



# COMPLEXITÉS



INHÉRENTE ET

ACCIDENTELLE

Created by Weltenraser  
from the Noun Project

# OBJECTIF

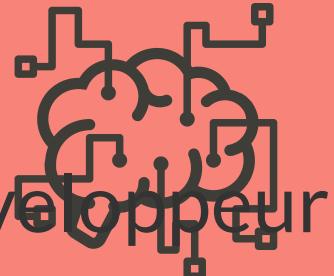


Created by Weltenraser  
from the Noun Project

Vous sensibiliser aux conséquences de vos choix en conception sur le plan de la complexité.



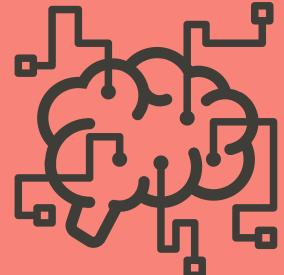
# DÉFIS POSÉS PAR LA COMPLEXITÉ



- La complexité est l'adversaire de tout développeur logiciel
- On peut distinguer entre deux sources de complexité (F. Brooks)
  - Le domaine du problème
    - « complexité inhérente (essentielle) »
  - La solution (logicielle)
    - « complexité accidentelle »

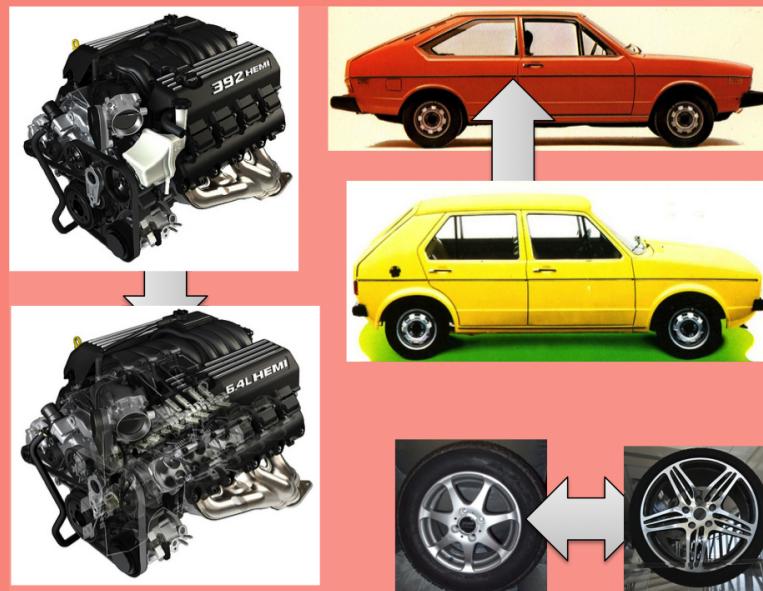
Created by Weltenraser  
from the Noun Project

# COMPLEXITÉS



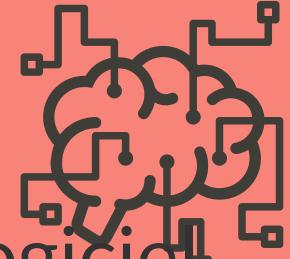
- Inhérente (essentielle)
  - « Essentielle » selon Aristote
  - Pour « être » une voiture, une voiture a un moteur, des roues, des portes, etc.
- Accidentelle (due à des choix)

Created by Weltenraiser  
from the Noun Project





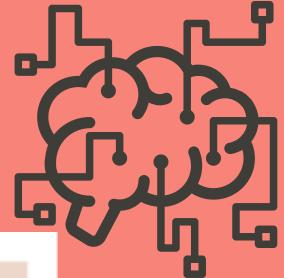
# COMPLEXITÉ INHÉRENTE



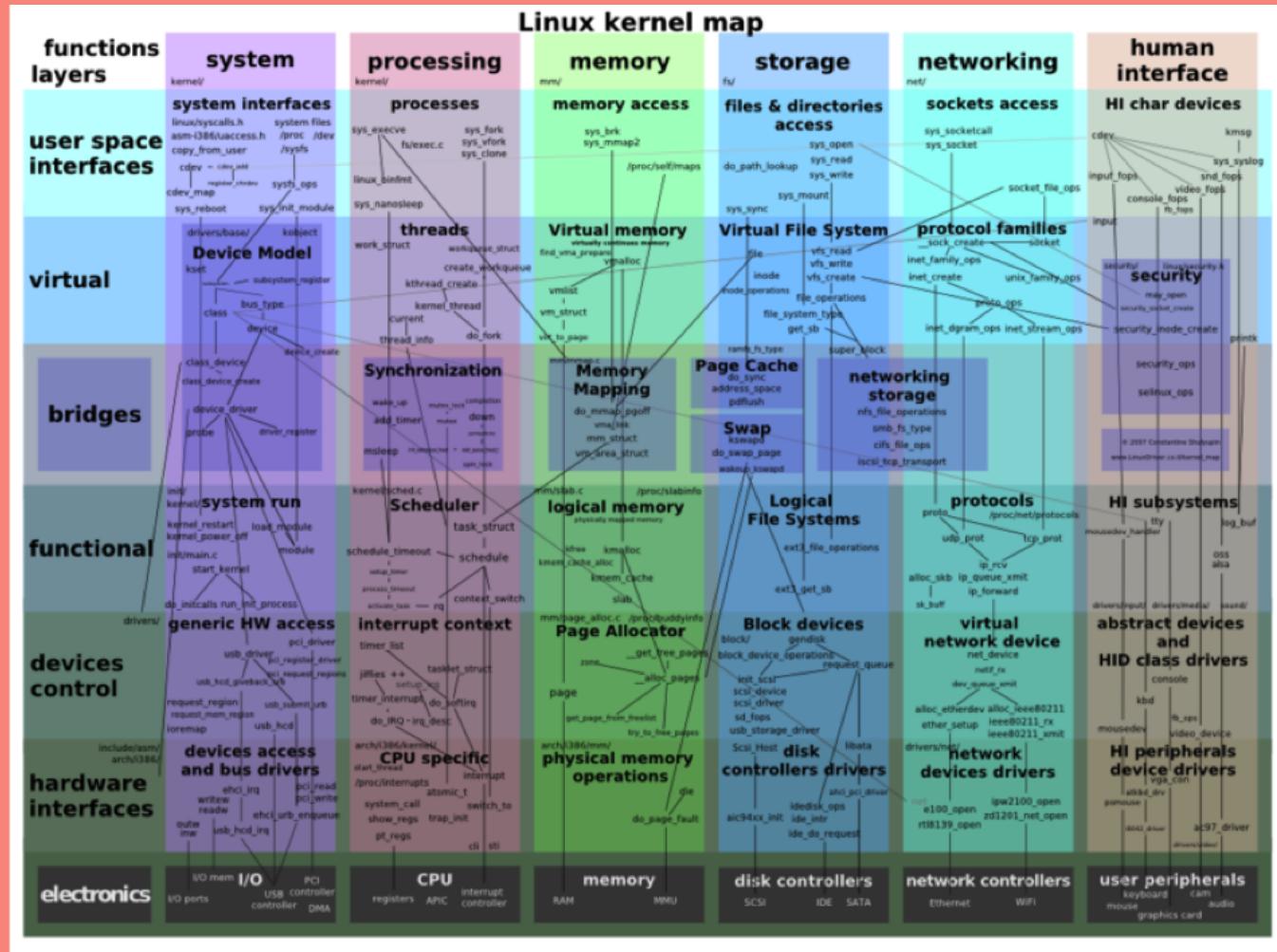
- Le domaine du problème concernant le logiciel peut être complexe
  - Exemple : système d'exploitation pour ordinateur
- La complexité inhérente est due au « problème » et non à la solution
  - Linux est complexe parce qu'il doit supporter plein de fonctionnalités...
- Synonyme : « complexité essentielle »

Created by Weltenraser  
from the Noun Project

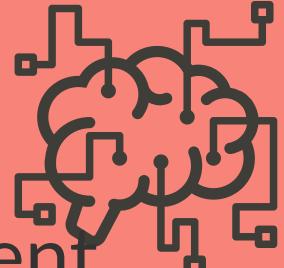
# COMPLEXITÉ INHÉRENTE



Created by Weltenraser  
The Noun Project



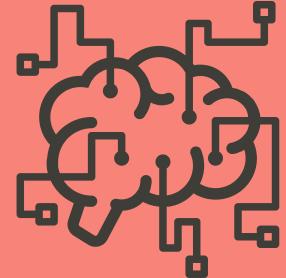
# COMPLEXITÉ ACCIDENTELLE



Created by Weltenraiser  
from the Noun Project

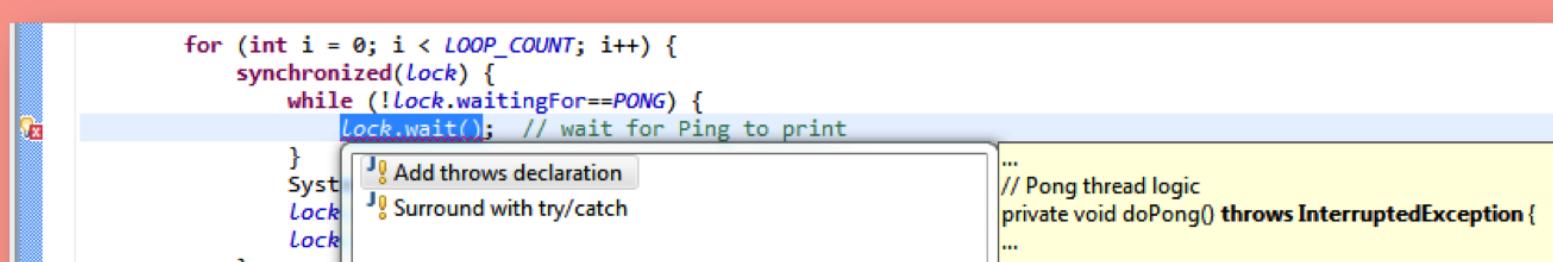
- Les choix proposés pour la solution peuvent introduire une complexité « accidentelle »
  - Langage de développement à bas niveau (p. ex. C ou assembleur)
  - Outils de débogage limités
- La complexité accidentelle peut être plus facilement gérée
  - Meilleurs outils (IDE), meilleures techniques, langage plus haut niveau, solution plus simple (si possible)

# EXEMPLE : GESTION DE COMPLEXITÉ ACCIDENTELLE EN JAVA



- Compilation en ligne de commande
- Utilisation de IDE (Eclipse)

```
C:\>javac PingPong.java
PingPong.java:104: error: unreported exception
InterruptedException; must be caught or declared to
be thrown
                                lock.wait();
                                         ^
1 error
```



The screenshot shows a Java code editor in Eclipse. A tooltip is displayed over the line `lock.wait();`. The tooltip contains two items:

- ! Add throws declaration
- ! Surround with try/catch

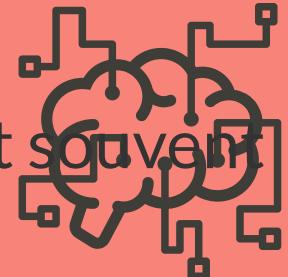
The code snippet is part of a loop that waits for a lock to be released:

```
for (int i = 0; i < LOOP_COUNT; i++) {
    synchronized(lock) {
        while (!lock.waitingFor==PONG) {
            lock.wait(); // wait for Ping to print
        }
    }
}
```

Below the code, there is a partial implementation of a method:

```
... // Pong thread logic
private void doPong() throws InterruptedException {
    ...
}
```

# DISTINCTION N'EST PAS TOUJOURS FACILE



Created by Weltenraiser  
from the Noun Project

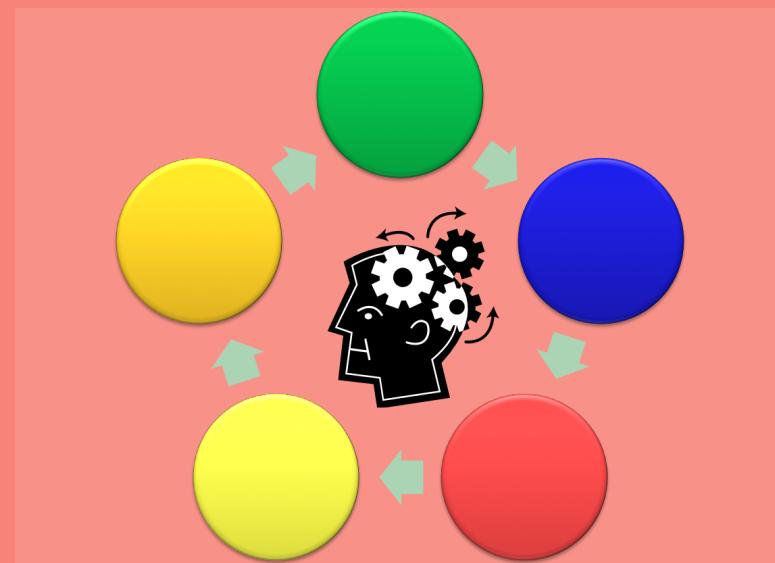
- Dans une conception logicielle, les éléments sont souvent complexes.
- La source de la complexité n'est pas toujours facile à identifier.
  - Exemple: logiciels de réseau
    - Difficultés rencontrées: délais, pertes d'informations, timeout, communication asynchrone , etc.
  - Sont-elles dues aux complexités inhérentes ou accidentielles?
- Communication asynchrone vs complexité inhérente
- Pertes d'informations vs complexité accidentelle (p.ex. avec protocole UDP)

# GÉRER LA COMPLEXITÉ



Created by Weltenraser  
from the Noun Project

- Réduire l'effort cognitif en gérant la complexité inhérente
  - Abstractions (OO)
  - Encapsulation (OO)
  - Masquage de l'information (OO)



# ABSTRACTION ET

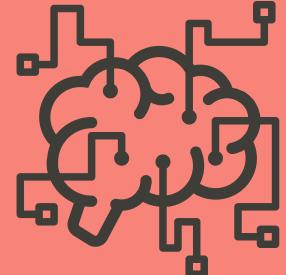
Created by Weltenraiser  
from the Noun Project

# ENCAPSULATION



# ABSTRACTION

## Vue plus simple

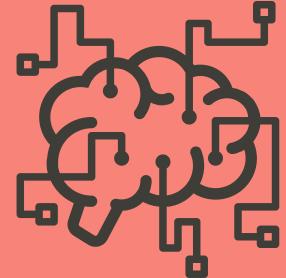


Created by Weltenraiser  
from the Noun Project



# ENCAPSULATION

## Accès limité



Created by Weltenraiser  
from the Noun Project

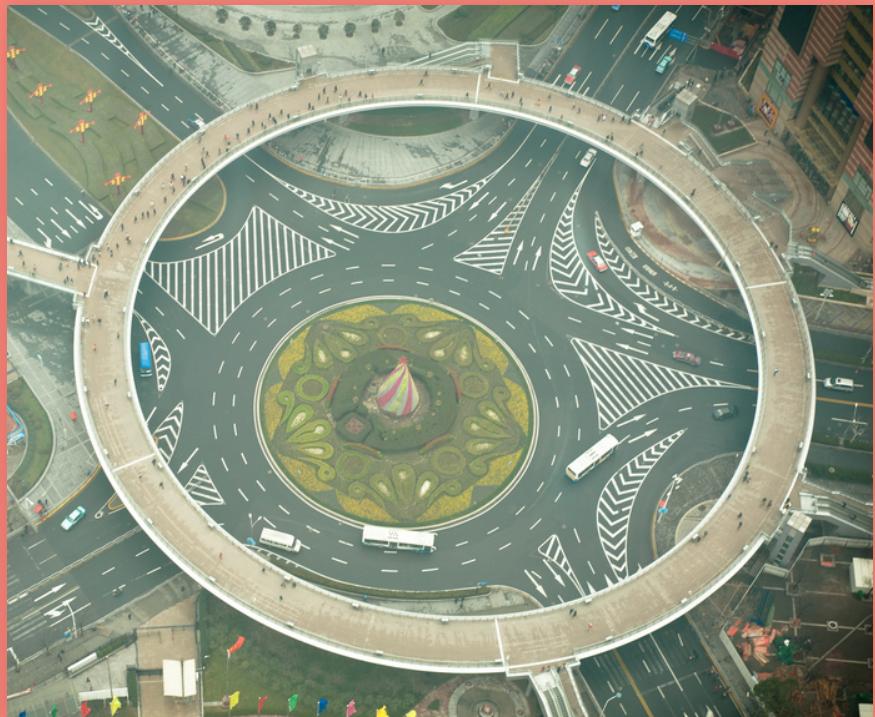


# PATTERNS POUR GÉRER LA COMPLEXITÉ



- Proposer des solutions aux problèmes récurrents dans un contexte particulier

Created by Weltenraser  
from the Noun Project



# DESIGN PATTERNS SONT UNE PARTIE DE LA SOLUTION

- Proposer des solutions aux problèmes récurrents dans un contexte particulier

Created by Weltenraser  
from the Noun Project



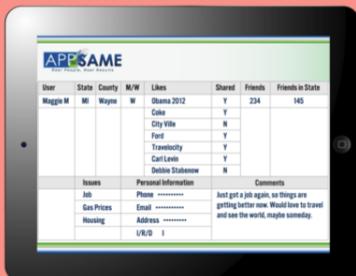
Appareils mobiles



Avionique







Automobile

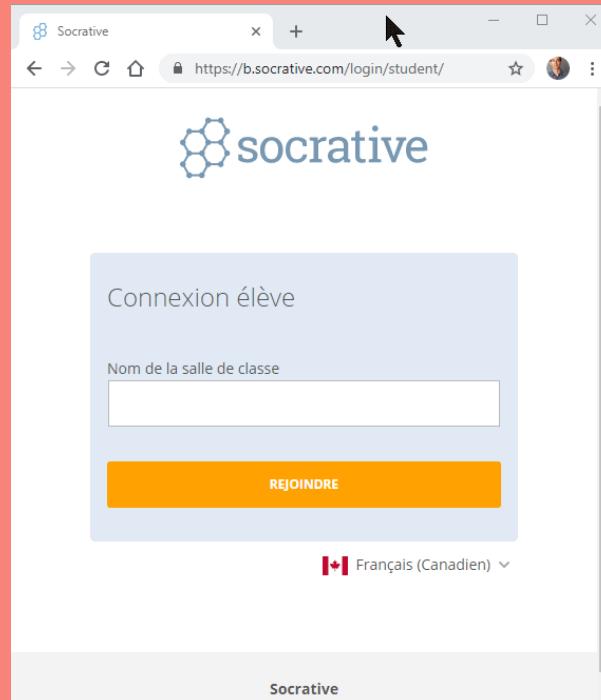


**SIGNETS**  
Guichet interactif

# SOCRATIVE OU ZOOM

Created by Weltenraser  
from the Noun Project

Nom de la salle de class: **ETSDESIGN**



# VALIDATION DE LA COMPRÉHENSION



- En développement logiciel, la complexité inhérente est due au domaine du problème et cause des complexités dans la solution.
  - Vrai?
  - Faux?

Created by Weltenraiser  
from the Noun Project

# VALIDATION DE LA COMPRÉHENSION



- La complexité inhérente est de mieux en mieux gérée grâce aux innovations technologiques.
  - Vrai?
  - Faux?

Created by Weltenraiser  
from the icons Project

# VALIDATION DE LA COMPRÉHENSION



- Le déverminage d'un algorithme de parcours d'arbre représentant une expression mathématique est difficile à cause de la complexité accidentelle
  - Vrai?
  - Faux?

Created by Weltenraiser  
from the Noun Project

# VALIDATION DE LA COMPRÉHENSION



- Created by Weltenraiser from the Noun Project
- Les outils de développement comme Eclipse répondent principalement aux difficultés dues à la complexité inhérente.
    - Vrai?
    - Faux?

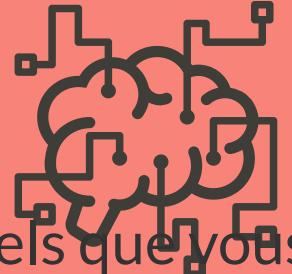
# VALIDATION DE LA COMPRÉHENSION



- Qu'est-ce qui n'est pas une technique pour gérer la complexité inhérente?
  1. Masquage de l'information
  2. Encapsulation
  3. Ramasse-miettes (récupérateur de mémoire)
  4. Abstraction
  5. Généralisation

Created by Weltenraser  
from the Noun Project

# RÉSUMÉ

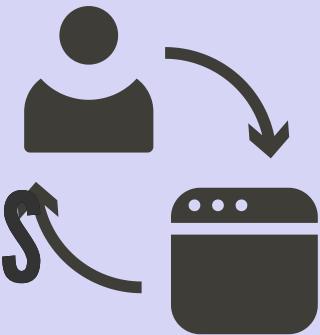


Created by Weltenraser  
from the Noun Project

- Soyez attentif aux sources de complexité dans les logiciels que vous développez
- La complexité inhérente ne peut pas être éliminée si votre problème est complexe
- Cependant, elle peut être gérée avec un bon choix d'abstractions.
- Patrons utilisent des abstractions et de l'encapsulation pour gérer la complexité
- Cependant, ils amènent aussi de la complexité accidentelle
- Vérifier:
  - Le problème récurrent que le patron est censé résoudre existe-t-il? (facile)
  - Le coût de la complexité due au patron est acceptable par rapport aux bénéfices (moins facile)

# LOG210 SÉANCE #05

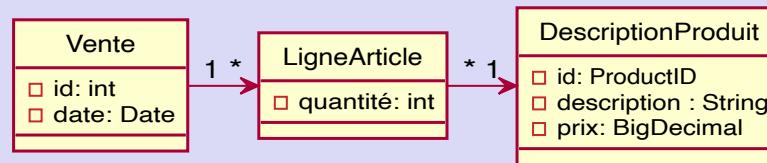
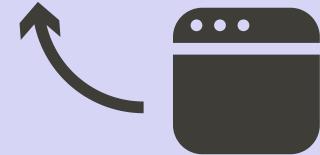
## ANALYSE ET CONCEPTION DE LOGICIELS



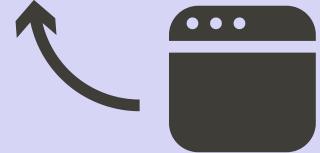
1. **Equipe** - Culture d'équipe
  2. **MDD** - Modèle du domaine - Révision
  3. **TypeScript** - Exemples TypeScript (SGB)
  4. **Complexité** - Inhérente et accidentelle
  5. **DSS** - Diagramme de séquence système 
- S20203
6. **Contrats** - Contrats pour réaliser les RDCU
  7. **RDCU** - Exercice avancé
  8. **DCL** - Diagramme de classes logiciel
  - ≡ 9. **Exercice** - Exercice

# RETOUR D'UNE OPÉRATION SYSTÈME (DSS)

- Du cas d'utilisation...
  - Le système présente le total incluant les taxes calculées.
- L'affichage du total => couche présentation
- Mais le calcul du total?
  - Aucune classe n'en a la responsabilité
  - Affectez-la avec « Expert »
  - (il y a un attribut dérivé / total dans le MDD)



# EXPERT POUR CALCULER LE TOTAL



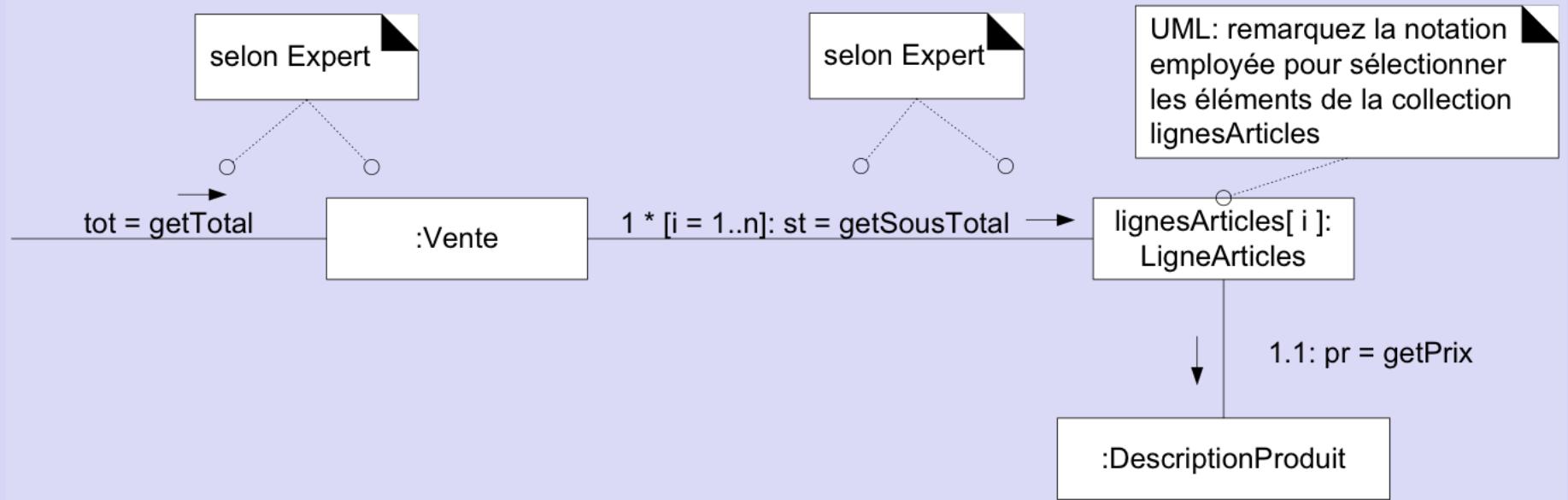
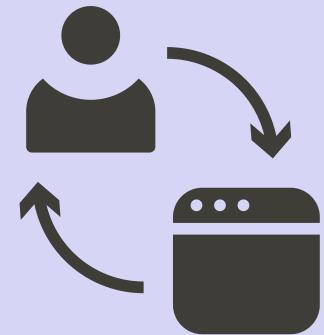
1. Énoncer la responsabilité
    - Qui doit connaître le montant total de la vente?
  2. Récapituler les informations nécessaires
    - total est la somme de tous les sous-totaux des lignes
    - sous-total de ligne := quantité \* prix
  3. Dresser la liste des informations nécessaires et des classes qui les possèdent.

Informations nécessaires au total de la vente	Expert
Produit.prix	Produit
LigneArticles.quantité	LigneArticles
Toutes les LigneArticles de la vente en cours	Vente

Analyse détaillée: p.F337

# CONCEPTION

## Vente.getTotal()



(diagramme de communication)

# LOG210 SÉANCE #05

## ANALYSE ET CONCEPTION DE LOGICIELS



Created by Mohammed Ra

1. **Equipe** - Culture d'équipe
2. **MDD** - Modèle du domaine - Révision
3. **TypeScript** - Exemples TypeScript (SGB)
4. **Complexité** - Inhérente et accidentelle
5. **DSS** - Diagramme de séquence système
6. **Contrats** - Contrats pour réaliser les RDCU 
7. **RDCU** - Exercice avancé
8. **DCL** - Diagramme de classes logiciel
- ☰ 9. **Exercice** - Exercice

# CONTRATS

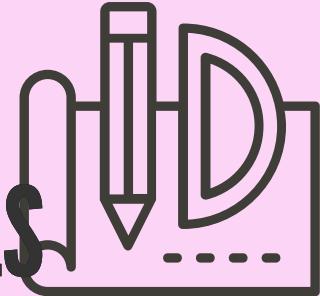


Created by Mohammed Ra

- sur la base de correspondance avec “Clé”
  - implique l’utilisation d’un tableau associatif (Map<>)

# LOG210 SÉANCE #05

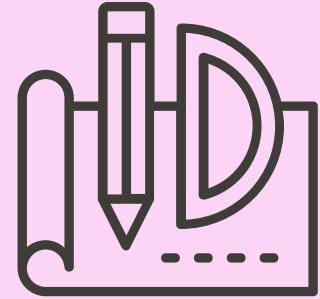
## ANALYSE ET CONCEPTION DE LOGICIELS



1. **Equipe** - Culture d'équipe
2. **MDD** - Modèle du domaine - Révision
3. **TypeScript** - Exemples TypeScript (SGB)
4. **Complexité** - Inhérente et accidentelle
5. **DSS** - Diagramme de séquence système
6. **Contrats** - Contrats pour réaliser les RDCU
7. **RDCU** - Exercice avancé  **S20203**
8. **DCL** - Diagramme de classes logiciel
9. **Exercice** - Exercice

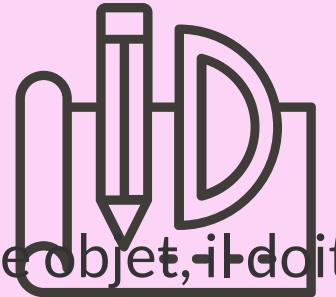


# RDCU-GRASP

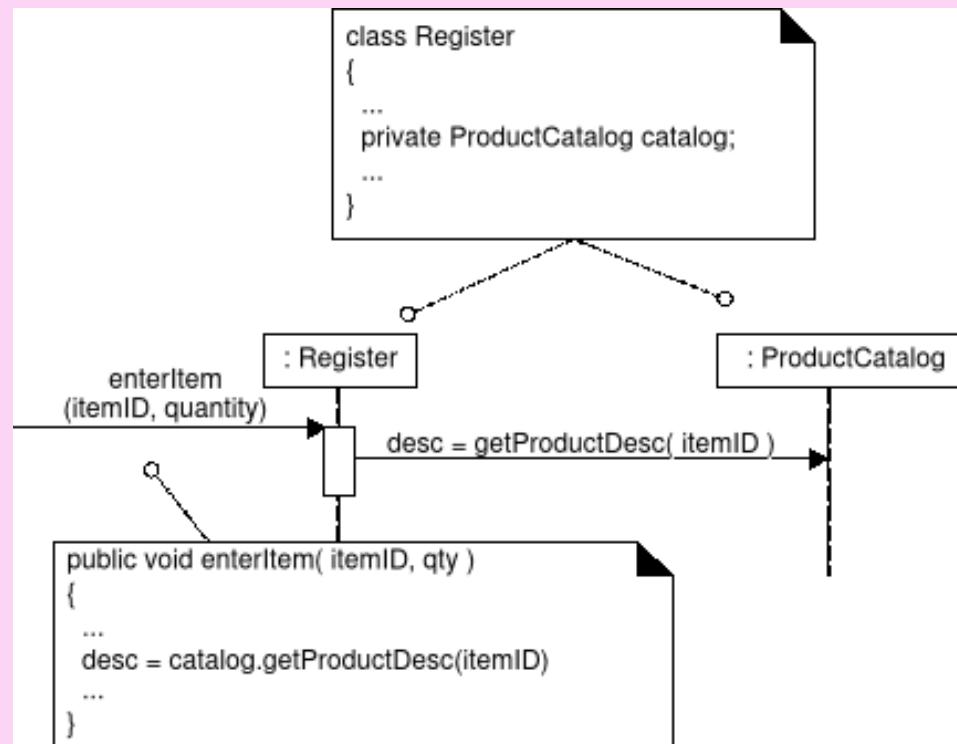


- 1. Contrôleur**
- 2. Créateur**
- 3. Expert en information**
- 4. Faible couplage**
- 5. Forte cohésion**
  - Polymorphisme
  - Indirection
  - Protection de variation
  - Fabrication pure

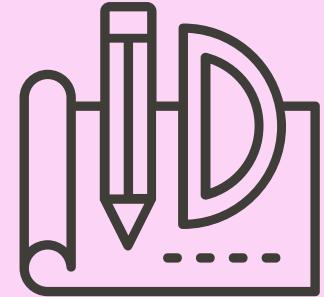
# VISIBILITÉ



- Avant qu'un objet puisse envoyer un message à un autre objet, il doit « voir » ce dernier
- Avoir une visibilité = posséder une référence (en Java) F18.1

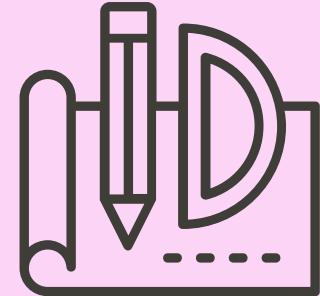


# TYPES DE VISIBILITÉ

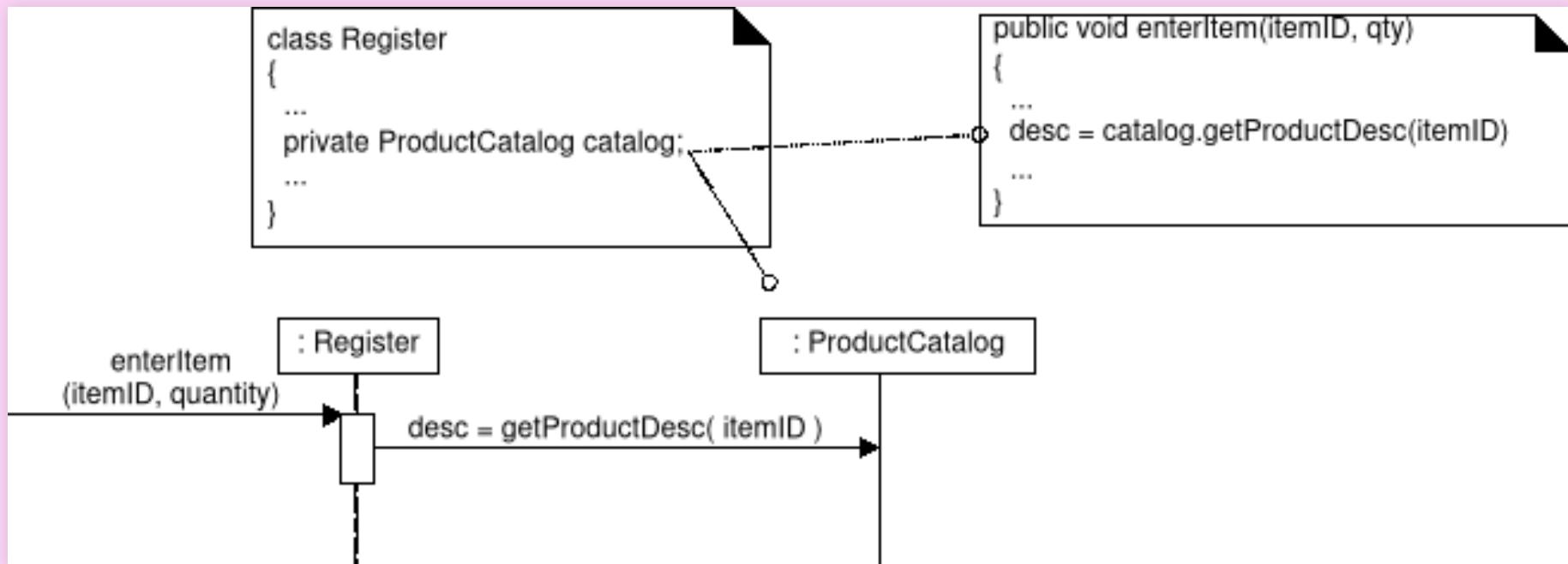


- Attribut
- Paramètre
- Locale
- Globale

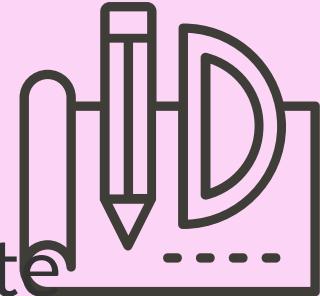
# VISIBILITÉ D'ATTRIBUT



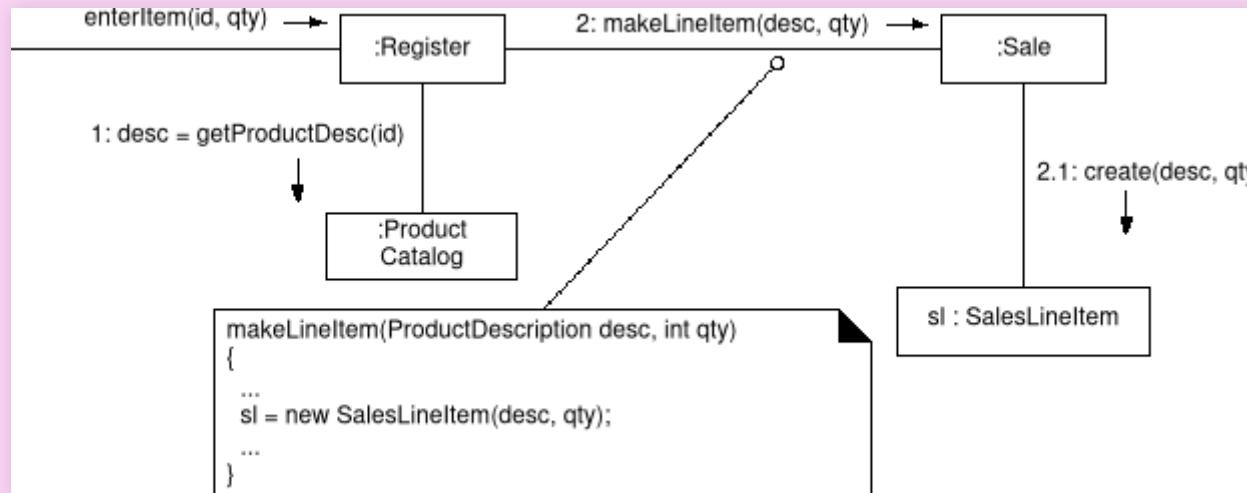
- Relativement permanente
- Très commune



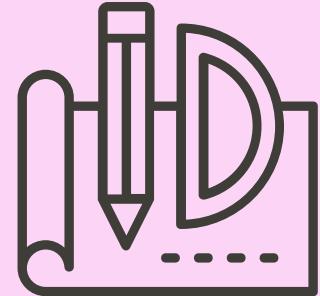
# VISIBILITÉ DE PARAMÈTRE



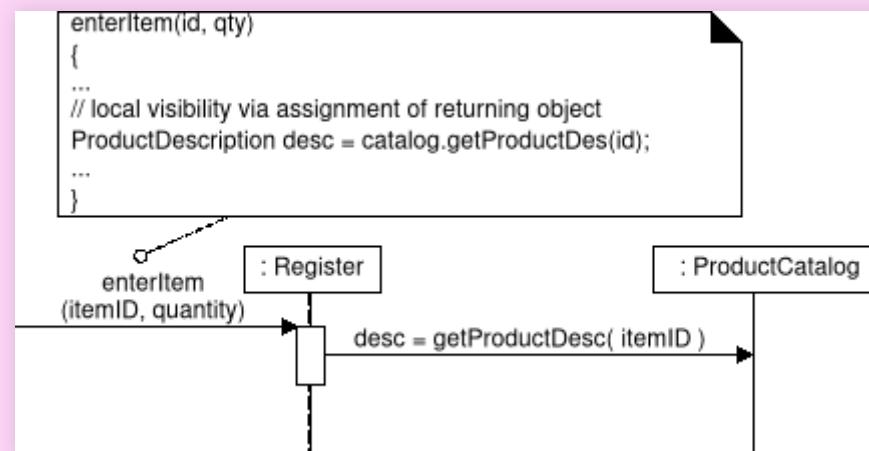
- Durée de vie relativement courte
- Paramètre peut devenir attribut



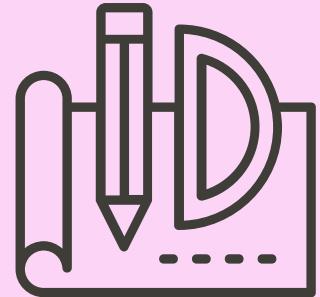
# VISIBILITÉ LOCALE



- Durée de vie relativement courte
- Peut être définie via
  - Instance créée localement
  - Instance retournée par une méthode



# VISIBILITÉ GLOBALE



- Relativement permanente
- Moins commune
- Meilleure définition via un singleton (GoF)

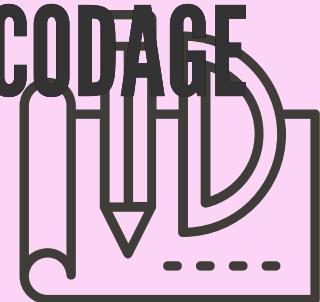
```
public class LogManager {  
    private java.io.PrintStream sout ;  
    private static LogManager lmInstance ;  
    private LogManager ( java.io.PrintStream out )  
    {  
        sout = out;  
    }  
    public void log( String msg )  
    {  
        sout.println ( msg );  
    }  
    public static LogManager getInstance () {  
        if ( lmInstance == null ) {  
            lmInstance = new LogManager ( System.out );  
        }  
        return lmInstance ;  
    }  
}
```

Usage:

```
LogManager.getInstance().log( "haaaaaaaah haaaaaahhaa");
```

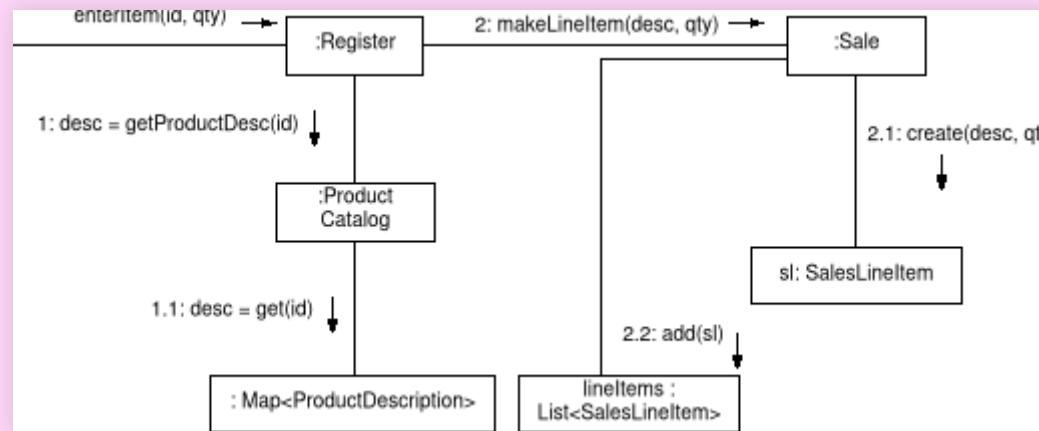


# CRÉATIVITÉ ET CHANGEMENTS PENDANT LE CODAGE

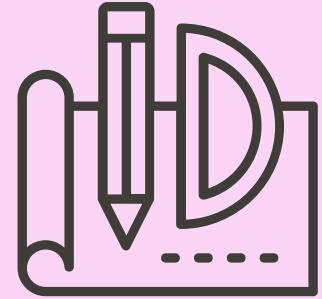


- Les artefacts de conception affectent l'implémentation
- Pendant la programmation et les tests
  - multitudes de changements peuvent surgir
- S'attendre à des déviations et les prévoir
  - Favoriser le changement
- Le codage n'est pas une activité triviale
  - Pourquoi?

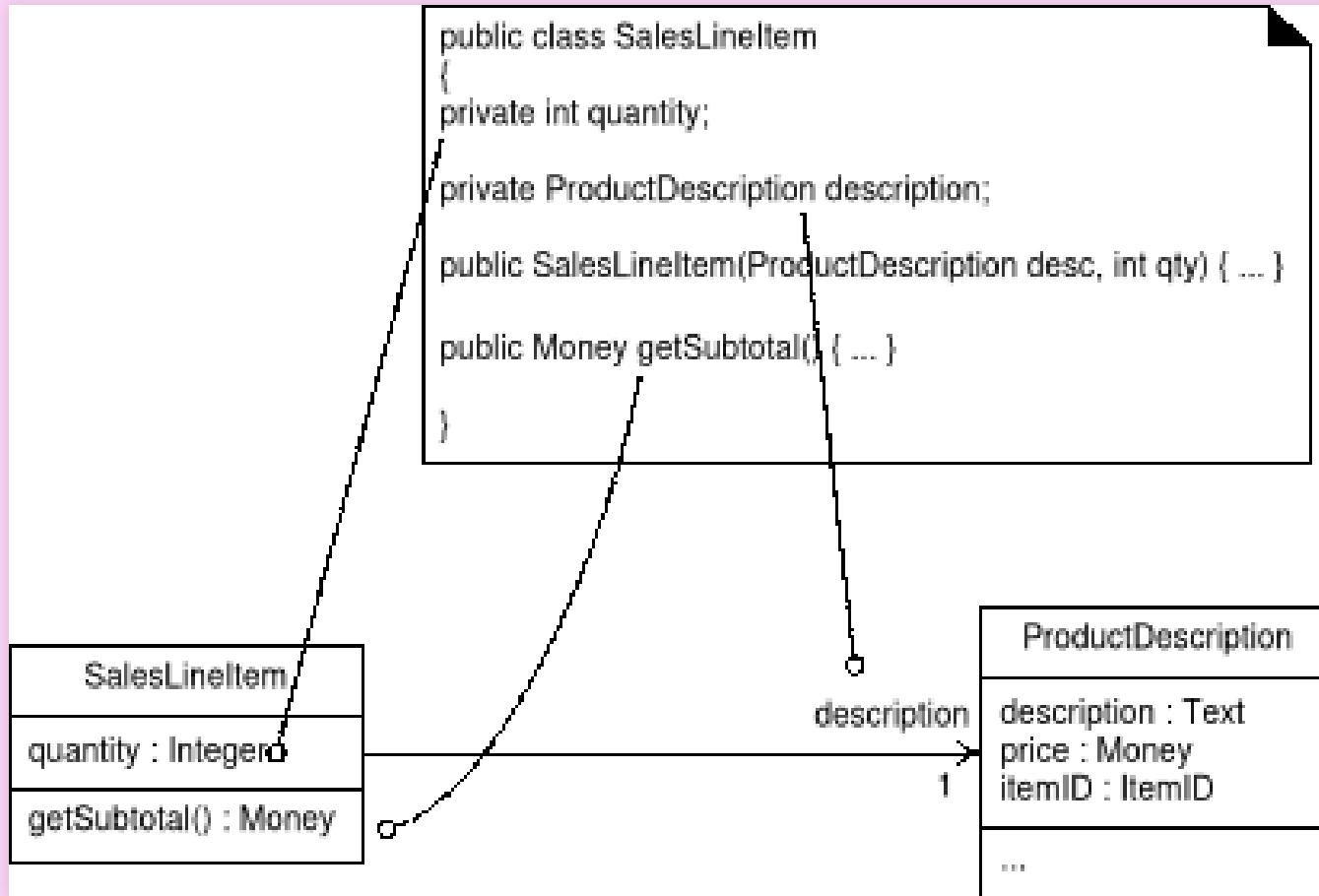
# MÉTHODES À PARTIR DES DIAGRAMMES D'INTERACTION



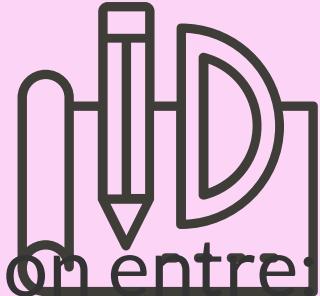
# SALESLINEITEM



- SalesLineItem en Java

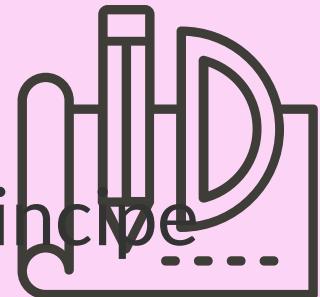


# RÉSUMÉ



- Nous avons vu des processus de traduction entre:
  - les diagrammes de classes UML et les définitions de classes
  - les diagrammes d'interaction UML et les corps des méthodes
- Mais le travail de programmation laisse encore beaucoup de place à l'exploration

# VALIDATION DE LA COMPRÉHENSION



- Quelle est la motivation principale du principe GRASP Créateur?
  1. Pour avoir une création rapide (performante) d'instances
  2. Pour obtenir une conception plus facile à comprendre
  3. Pour avoir une conception qui prend moins de mémoire
  4. Pour identifier des occasions d'appliquer le patron Fabrique

# VALIDATION DE LA COMPRÉHENSION



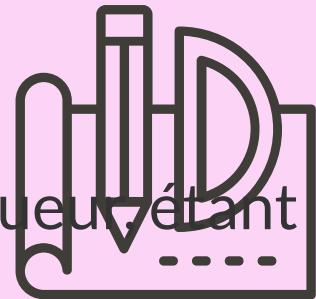
- Le pattern Contrôleur GRASP sert à trouver la bonne classe...
  1. de la couche présentation pour réaliser une opération système
  2. de la couche domaine pour réaliser une opération système
  3. de l'IHM pour implémenter une logique applicative
  4. de l'IHM pour contrôler le système

# VALIDATION DE LA COMPRÉHENSION

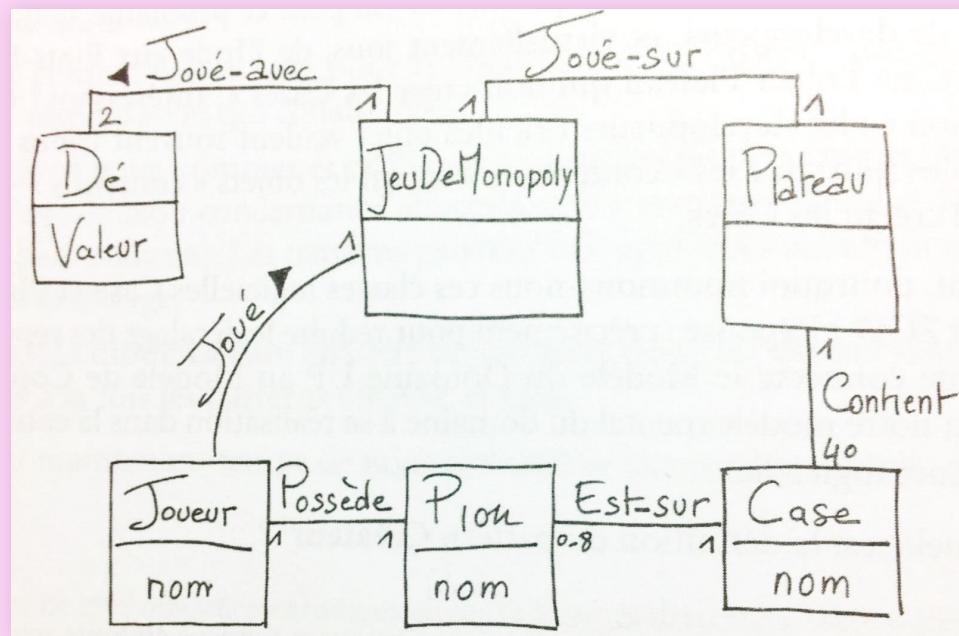


- Quel genre de contrôleur GRASP est JeuDeMonopoly dans l'exemple?
  1. Contrôleur de jeu
  2. Contrôleur MVC
  3. Contrôleur de session
  4. Contrôleur de façade

# VALIDATION DE LA COMPRÉHENSION

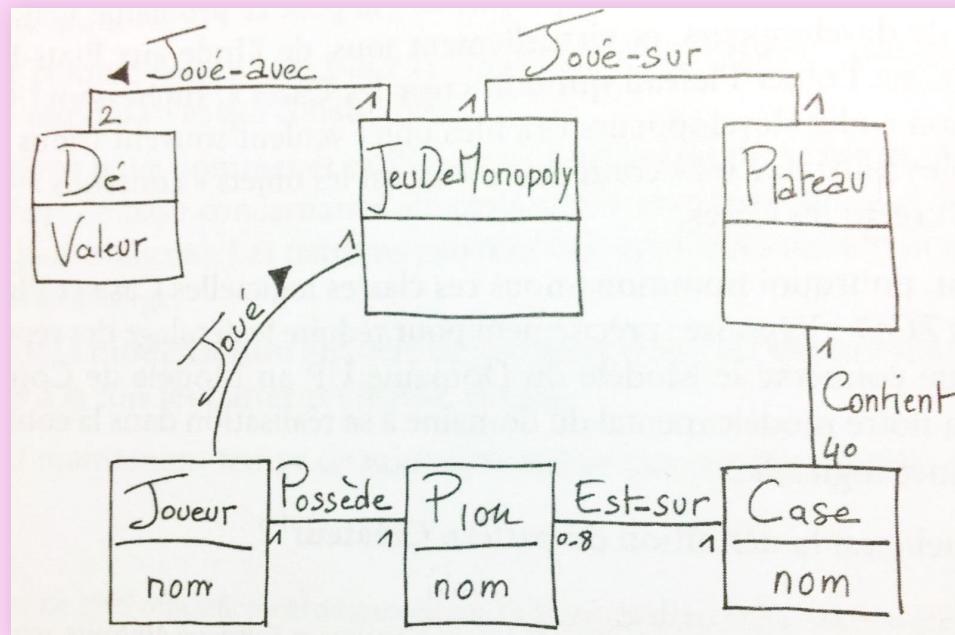
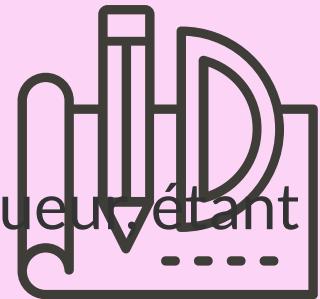


- Selon la figure A17.3/F16.3, qui connaît un objet Joueur, étant donné le nom du joueur?



# VALIDATION DE LA COMPRÉHENSION

- Selon la figure A17.3/F16.3, qui connaît un objet Joueur, étant donné le nom du joueur?



Quel est ce patron GRASP?

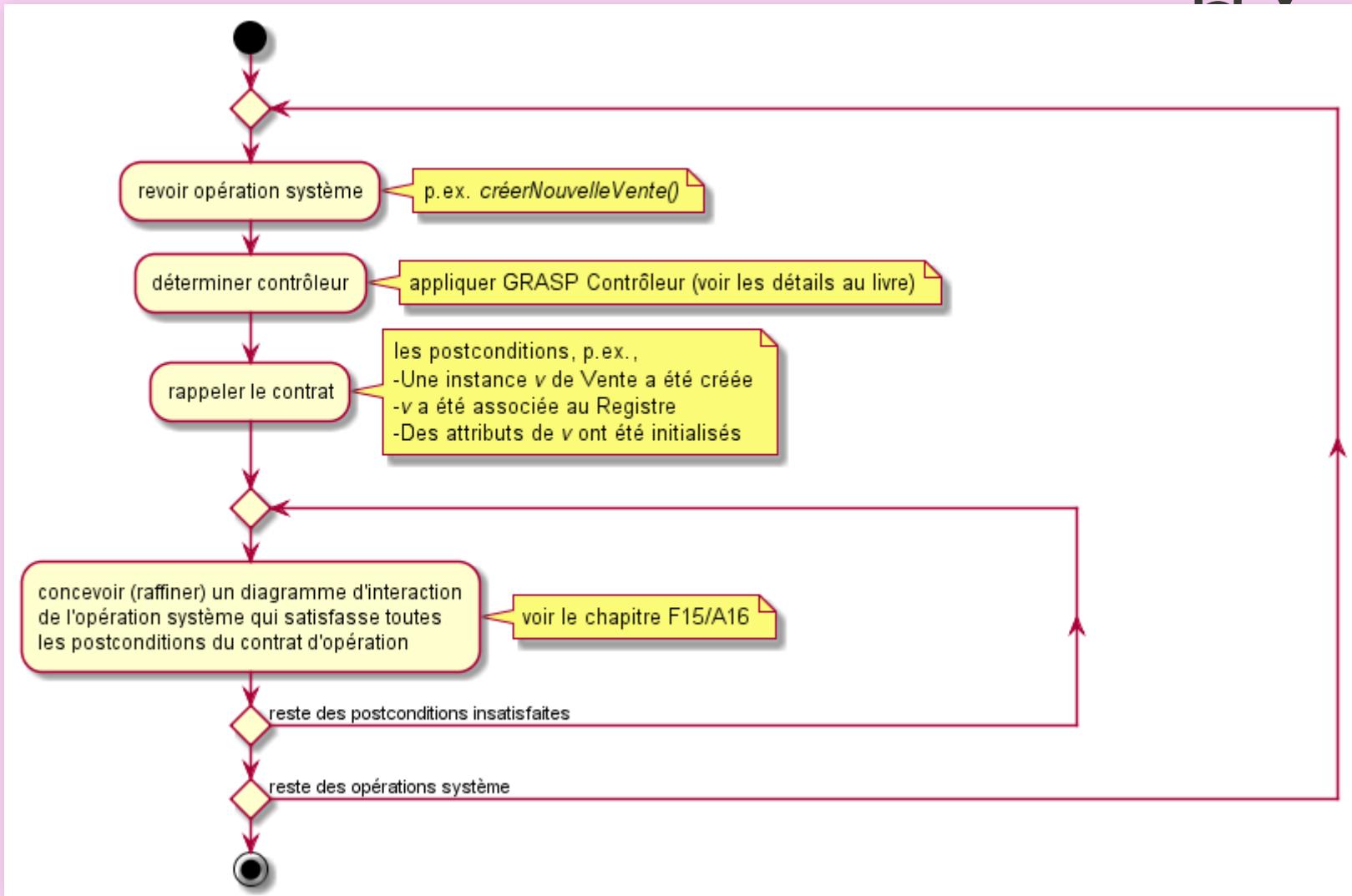
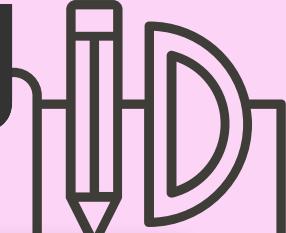
# RÉALISATION DE CAS D'UTILISATION NEXTGEN



- La zone non-magie
  - Création de diagrammes d'interaction bien conçus n'est pas magique!
  - Tout s'appuie sur des principes de bonne conception justifiables.
    - GRASP (General Responsibility Assignment Software Pattern)
- L'examen final est aussi une zone non-magie!
  - Il faut y faire preuve d'utilisation des principes justifiables (GRASP)



# PROCESSUS DE HAUT NIVEAU



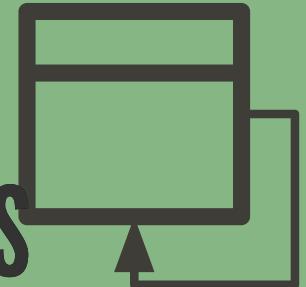
# RÉALISATION DE CAS D'UTILISATION



- Cela « décrit la façon dont les cas d'utilisation sont réalisés dans le Modèle de Conception, en termes d'objets qui collaborent » [RUP]
  - Réalisation de scénario
- Cela fait le lien entre exigences et conception
- C'est un diagramme d'interaction
- C'est une documentation des décisions de COO
  - annotations GRASP

# LOG210 SÉANCE #05

## ANALYSE ET CONCEPTION DE LOGICIELS

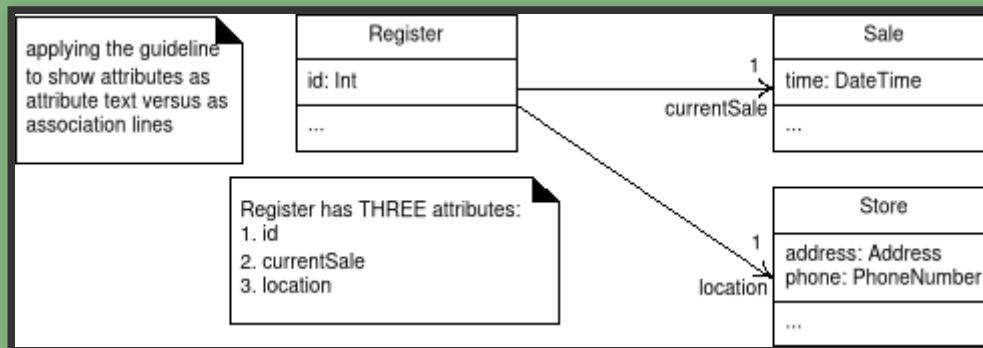


Created by Swen-Peter Ekkebus  
from the Noun Project

1. **Equipe** - Culture d'équipe
2. **MDD** - Modèle du domaine - Révision
3. **TypeScript** - Exemples TypeScript (SGB)
4. **Complexité** - Inhérente et accidentelle
5. **DSS** - Diagramme de séquence système
6. **Contrats** - Contrats pour réaliser les RDCU
7. **RDCU** - Exercice avancé
8. **DCL** - Diagramme de classes logiciel  **S20203**
9. **Exercice** - Exercice



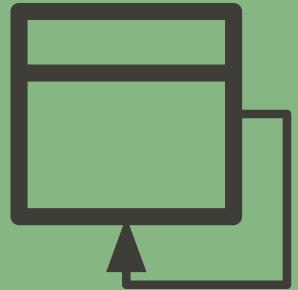
# DEUX FAÇONS DE NOTER LES ATTRIBUTS



Created by Swen-Peter Ekkelbus  
from the Noun Project

```
public class Register {  
    private int id;  
    private Sale currentSale;  
    private Store location;  
    // ...  
}
```

# CLASSES COLLECTION



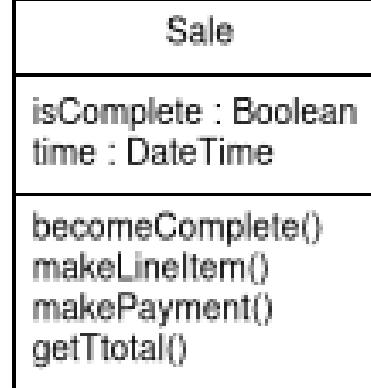
Created by Swen-Peter Ekkebus  
from the Noun Project

- Multiplicités des associations
- Besoin d'un groupe d'objets similaires
- Choix du type de collection
  - influencé par le besoin

```
public class Sale{  
    private List<SalesLineItem> lineItems =  
        new ArrayList<SalesLineItems>();  
    // ...  
}
```

# AJOUTER UNE CLASSE DE COLLECTION

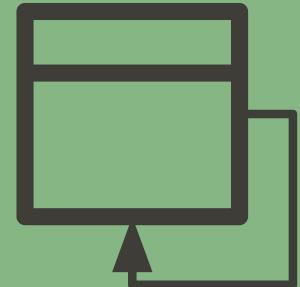
```
public class Sale  
{  
    ...  
  
    private List<LineItem> lineItems = new ArrayList();
```



A collection class is necessary to maintain attribute visibility to all the SalesLineItems.

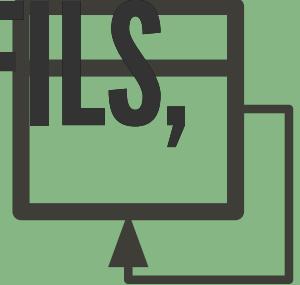
# OPÉRATION VS MÉTHODE

- UML DCC : signature des opérations
  - Déclaration (décision de conception)
  - Opérations sont quasiment « vides » en UML
- Java : implémentation des méthodes
  - Implémentation (décision de codage)
- Déclaration d'**opération** (UML) est différente de l'implémentation de **méthode** (langage OO)
  - P.ex. un contrat d'opération définit les contraintes sur une opération et non une méthode
  - La différence est subtile mais importante



Created by Swen-Peter Ekkebus  
from the Noun Project

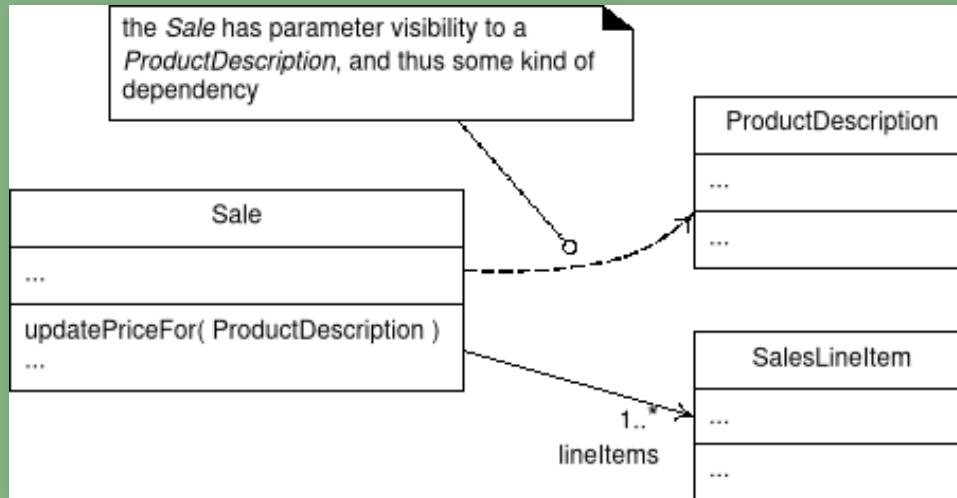
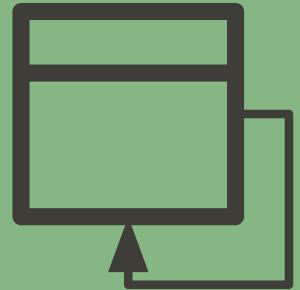
# MOTS CLÉS, STÉRÉOTYPES, PROFILS, ÉTIQUETTES



Created by Swen-Peter Ekkebus  
from the Noun Project

- Voir les sections A16.7/F15.7, A16.8/F15.8
  - Stéréotype <create> , <destroy>, <metaclass>, <Actor>, <Interface>, ...
  - Le stéréotype déclare un ensemble d'étiquettes
- Etiquette: @login\_required
  - Fonctionnalités et attributs que l'on veux donner à une classe ou une méthode.

# NOTATION DE DÉPENDANCE



Created by Swen-Peter Ekkebus  
from the Noun Project

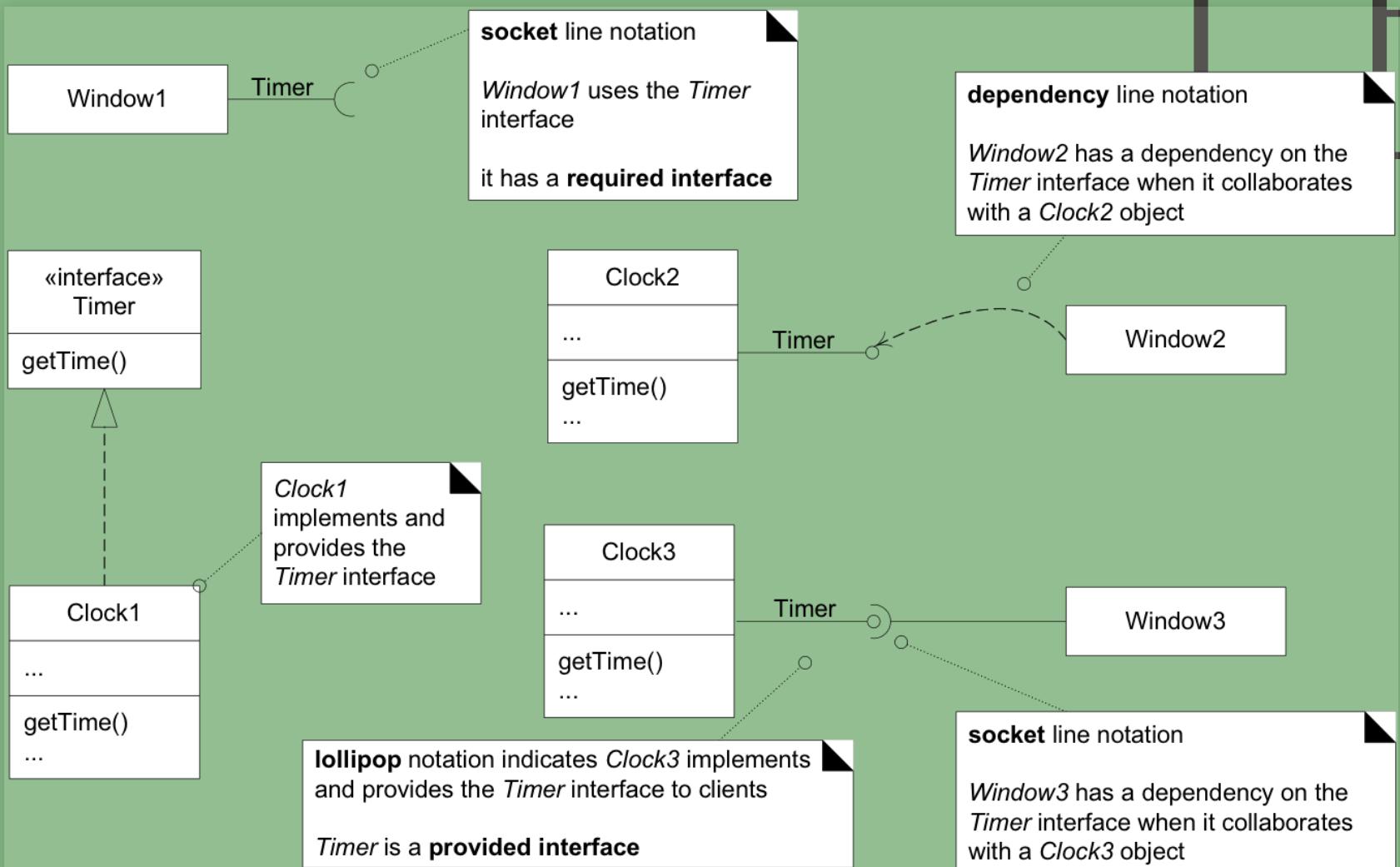
fig F15.9

```

public class Sale{
    public void updatePriceFor(
        ProductDescription
        description) {
            Money basePrice = description.getPrice();
            // ...
        }
        // ...
    }
  
```



# Interfaces et réalisation d'interface



# LOG210 SÉANCE #05

## ANALYSE ET CONCEPTION DE LOGICIELS

1. **Equipe** - Culture d'équipe
2. **MDD** - Modèle du domaine - Révision
3. **TypeScript** - Exemples TypeScript (SGB)
4. **Complexité** - Inhérente et accidentelle
5. **DSS** - Diagramme de séquence système
6. **Contrats** - Contrats pour réaliser les RDCU
7. **RDCU** - Exercice avancé
8. **DCL** - Diagramme de classes logiciel
9. **Exercice** - Exercice  S20203



# RÉVISION EXERCICE

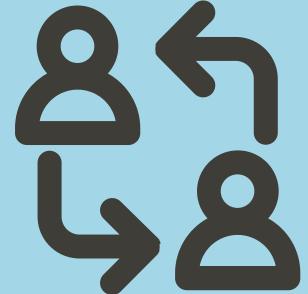
## SYSTÈME D'ÉCHANGE DE LIVRE

<https://github.com/yvanross/LOG210-exercices>



# SÉANCE #05

## 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>

