

**M1 info**

GINF41B2 (Conception et Programmation Orientée Objet)

# Cours #1

## La conception orientée objet

Pierre Tchounikine

Les concepts de l'Orienté Objet (OO)  
Principes généraux  
Exemple (et UML)

# **Les concepts de base de l'Orienté Objet**

# Les concepts de base de l'orienté objet

- Modélisation et abstraction
- Objet
- Encapsulation
- Classification, héritage et polymorphisme
- Agrégation


# Modélisation et abstraction (1/2)

- Abstraction
  - processus qui consiste à identifier les caractéristiques intéressantes d'une entité, en vue d'une utilisation précise
- Modélisation
  - activité de définition des abstractions pertinentes
- Modèle
  - abstraction de la réalité, vue subjective de la réalité qui reflète des aspects importants de la réalité

# Modélisation et abstraction (2/2)

- Objectif de l'abstraction et de la modélisation :
  - comprendre avant de construire, faciliter la compréhension du système étudié en réduisant la complexité
  - disposer d'un support matériel (le modèle) permettant de visualiser et d'échanger

Important : on abstrait toujours par rapport à un point de vue (ce qui est « important » dépend du but)

- 
- il n'y a pas un modèle, mais des modèles
  - il n'y a pas de bons ou de mauvais modèles, mais des modèles adéquats ou inadéquats, des modèles plus ou moins utiles

# L'objet

- Un objet est une entité définie par
  - une identité (un nom)
  - un ensemble d'attributs qui caractérisent son état
  - un ensemble d'opérations (méthodes) qui définissent son comportement
- Un objet est une instance d'une classe
- Une classe est un type de données abstrait
  - caractérisé par des propriétés (attributs et méthodes) communes
  - permettant de créer des objets possédant ces propriétés

# L'encapsulation

- Encapsulation :
  - principe consistant à masquer les détails d'implémentation d'un objet
- Idée : dissocier
  - une « vue externe » = les services offerts aux utilisateurs de l'objet
  - une « vue interne » = comment les services sont réalisés
- Avantages :
  - évolution : on peut modifier l'implantation sans modifier l'interface  
(on peut modifier le « comment » sans modifier le « quoi »)
  - intégrité des données : on ne peut pas modifier l'objet autrement qu'en passant par l'interface
  - baisse de la complexité

*NB. l'encapsulation n'est pas spécifique aux langages objets (cf. ADA par exemple)*

# Classification, héritage et polymorphisme

- L'héritage est fondé sur des processus « naturels » :
  - la classification
  - la mise en facteur
- Principe (classification) : sens inverse : mise en facteur
  - par un processus d'abstraction on identifie
    - des classes d'objets (objets ayant les mêmes propriétés)
    - des spécialisations de ces classes (caractéristiques plus spécifiques)
  - les propriétés (attributs et méthodes) sont transmises vers les sous-classes
- Avantages :
  - évite la duplication de code (mise en facteur)
  - favorise la réutilisation
  - permet le polymorphisme (une méthode peut être définie dans différentes classes et s'appliquer à des objets différents)
    - ✓ **code plus simple à gérer**
    - ✓ **baisse de la complexité**



# Agrégation

- Agrégation = possibilité de définir une classe à l'aide d'objets d'une autre classe
  - possibilité de construire des objets complexes
  - idée de « délégation »

# Les concepts de la COO

- Un petit nombre de concepts
- Une démarche fondée sur l'abstraction et la différenciation des concepts
  - même s'ils ont une définition formelle identique
  - même s'ils ont une implantation identique (type « abstraits »)
- Des concepts uniformes tout au long du cycle de vie (définition des besoins, analyse, conception, programmation)
  - pas de rupture conceptuelle
- Des représentations graphiques → facilite la « conception »
  - (mais il y a également des formalismes tout à fait formels)

# **Démarche COO générale**

# Démarche très générale de la COO

un processus fondé sur l'élaboration

- **Modélisation**
  - construire un modèle du monde mettant en évidence les propriétés importantes
    - abstraction précise et concise des acteurs, des concepts liés à l'application et de leurs relations (et non de comment sera construite l'application)
    - description fondée sur les objets du domaine et non des objets informatiques (des structures de données), pas de décision d'implantation
- **Conception du système (architecture)**
  - *opérer un découpage du système sur la base de l'analyse*
- **Conception des objets (détails)**
  - implantation des objets (affiner les objets, définir les structures de données et les algorithmes, ajout des détails nécessaires à l'implantation)
- **Implantation**
  - traduction dans un langage cible

*de façon itérative, par cycle,  
en affinant petit à petit*

# Démarche très générale de la COO



C'est fondamentalement **une façon de penser** et pas une façon de programmer

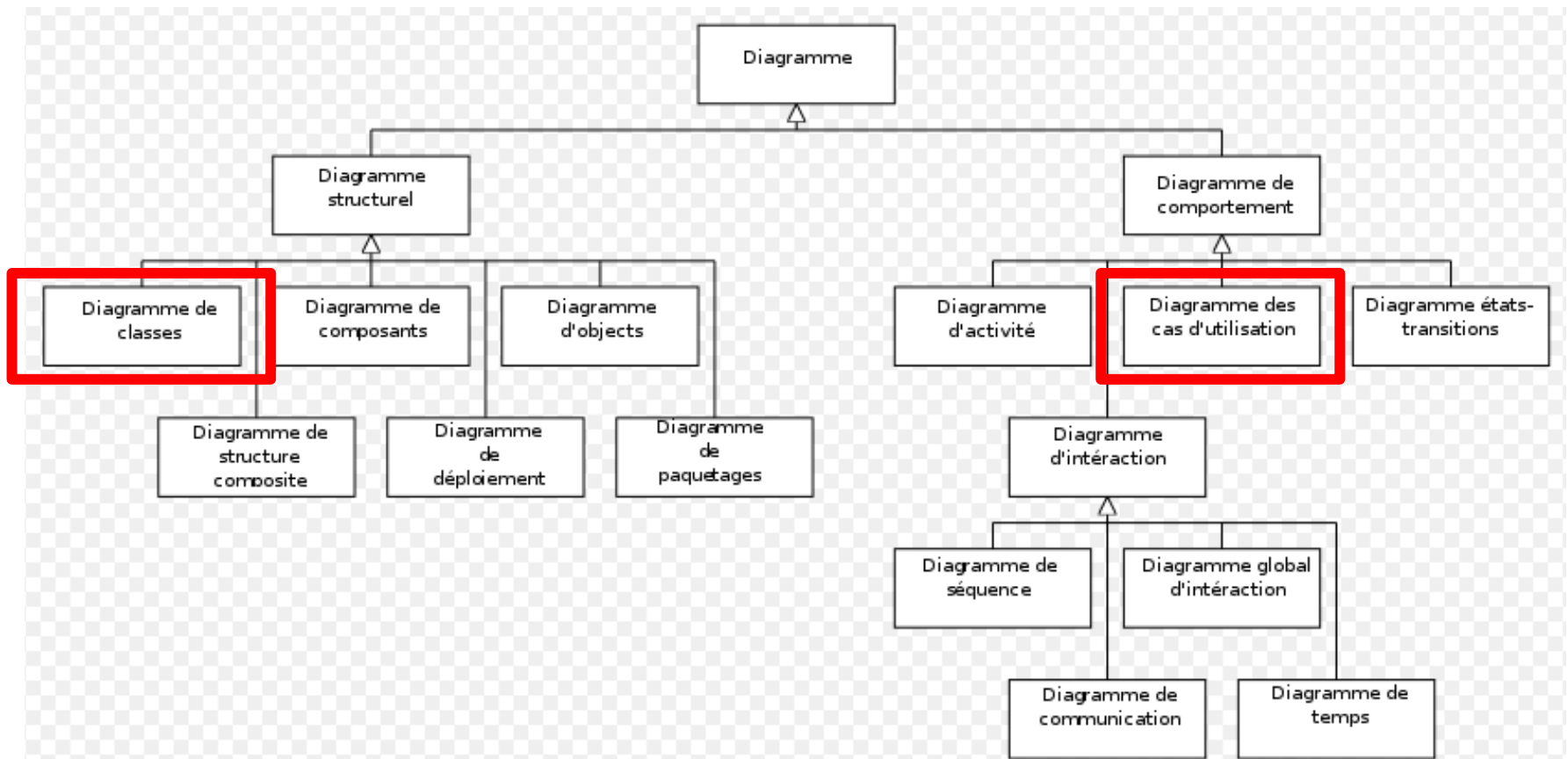
- une erreur fréquente consiste à faire une conception « fonctionnelle » et, parce qu'on programme en objet, croire que l'on fait de la conception orientée objet !
- on peut faire une analyse objet et la programmer par une BD ou un langage « classique » non-objet

*mais bon, COO + POO, c'est pas mal quand même !*

- on peut utiliser un langage orienté objet ... de façon « non-pertinente »

# Un formalisme de base : UML

vous devez maîtriser les bases d'UML



# Un formalisme de base : UML

- un formalisme qui permet de décrire une analyse orientée objet comme un ensemble de vues complémentaires d'un système
- des notations adaptées aux différentes vues et à leurs concepts sous-jacents
  - identifier et décrire les besoins (la modélisation est guidée par les besoins)
  - comprendre et décrire le domaine
  - comprendre et décrire comment on va utiliser cette description du domaine pour répondre aux besoins

**vous devez maîtriser les bases d'UML**

# Pourquoi il faut maîtriser UML

- Principaux intérêts d'UML :
  - universel, indépendant des langages d'implantation
  - semi-formel
  - fondé sur un métamodèle
  - associé à des notations graphiques → visuel
  - fondé sur un large consensus
  - standardisé (notation standard, format d'échange, évolution contrôlée)
  - reconnu et utilisé en entreprises
  - industrialisé (édition des modèles, génération de code, rétro-ingénierie, ...)



UML n'est pas une méthode

(pas de processus imposé)



**Exemple**

# Exemple : le barman virtuel

*projet étudiant*

## Le sujet

Le travail consiste à concevoir et réaliser un logiciel d'aide à la gestion de cocktails.

Un cocktail est une boisson composée, à partir de différents ingrédients, selon un processus précis. Il a différentes propriétés : taux d'alcool, couleur, force, présentation, etc.

La fonctionnalité de base du logiciel est de servir de barman virtuel : proposer au client différents cocktails possibles, l'aider à choisir en lui présentant les cocktails selon différents points de vue, le cas échéant essayer de le satisfaire au mieux en créant un cocktail sur mesure ou en adaptant une recette de cocktail en fonction des ingrédients disponibles.

Le logiciel doit également permettre de créer des points de vue particuliers et d'y rattacher les cocktails déjà créés (par exemple, un établissement peut souhaiter présenter ses cocktails en fonction de thèmes propres à son établissement et/ou à une période).

# Définition des besoins

- Il faut identifier les acteurs en se posant des questions du type :
  - qui utilise le système ? (partir des « métiers »)
  - qui le maintient ?
  - avec quels autres systèmes interagit-il ?
  - etc.
  
- Puis identifier les cas d'utilisation :
  - on prend les acteurs un à un
  - on étudie comment ils utilisent le système (les scénarios : à partir d'un stimulus de l'acteur, il se passe ...)
  - on regroupe, on spécialise, on réorganise

# Exemple : le barman virtuel

projet étudiant

## Les acteurs (définition informelle)

### I. Définition des utilisateurs

On distingue trois utilisateurs : le Client, le Barman, et l'Administrateur.

Le **client** est un utilisateur non enregistré.

Son rôle est de réaliser des commandes ainsi que de créer ses propres cocktails. Il effectue sa commande tout en restant à sa table à l'aide d'un écran tactile qui sera disposé à chaque table du bar.

Ses besoins sont de pouvoir consulter la carte, passer commande, créer des cocktails sur mesure.

Le **barman** est un utilisateur enregistré.

Son rôle est de recevoir automatiquement les commandes sur son écran et de réaliser les cocktails commandés.

Ses besoins sont de pouvoir consulter les commandes passées, pouvoir consulter pour chaque cocktail d'une commande, la recette associée.

L'**administrateur** est un utilisateur enregistré.

Son rôle est de gérer les stocks, les cocktails, les thèmes ainsi que les utilisateurs.

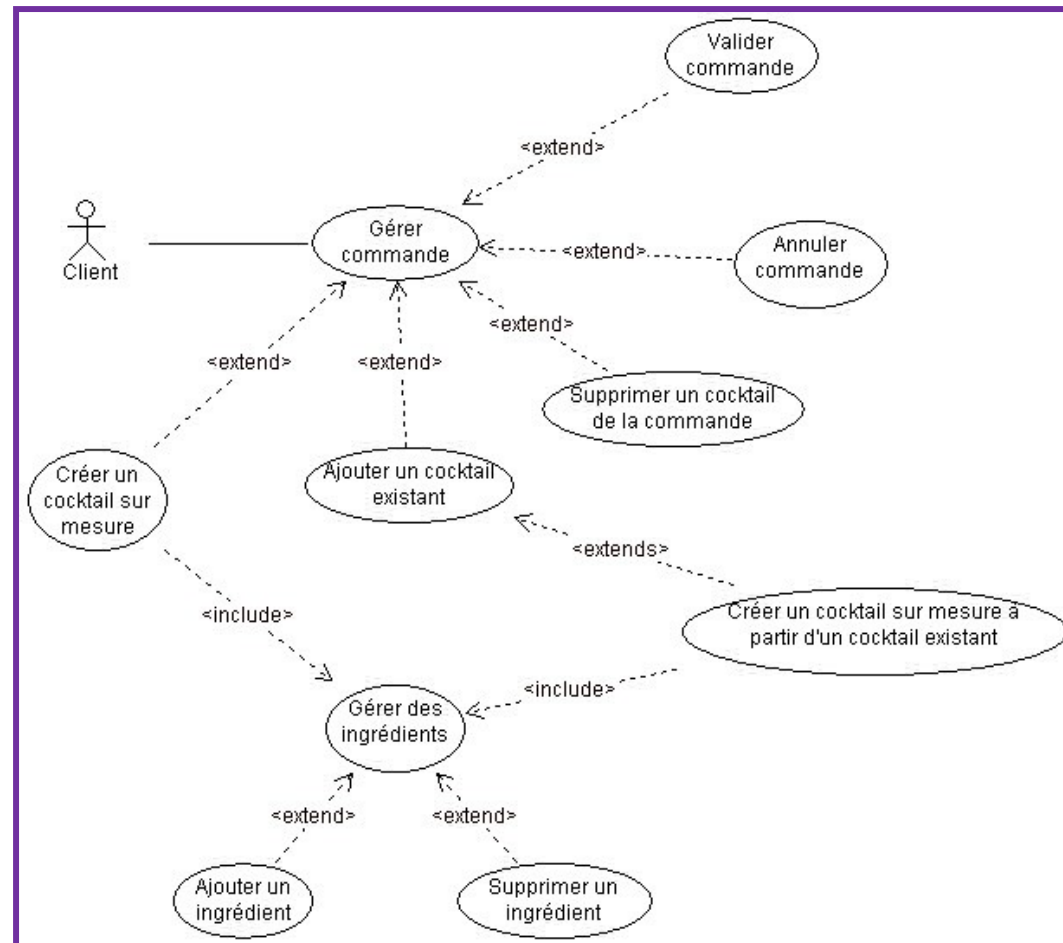
Ses besoins sont de pouvoir gérer les cocktails, les thèmes et les utilisateurs (Barman ou Administrateur). Il doit aussi pouvoir gérer le stock et visualiser l'historique des commandes passées.

# Exemple : le barman virtuel

projet étudiant

## Les acteurs (formalisme des Use Case)

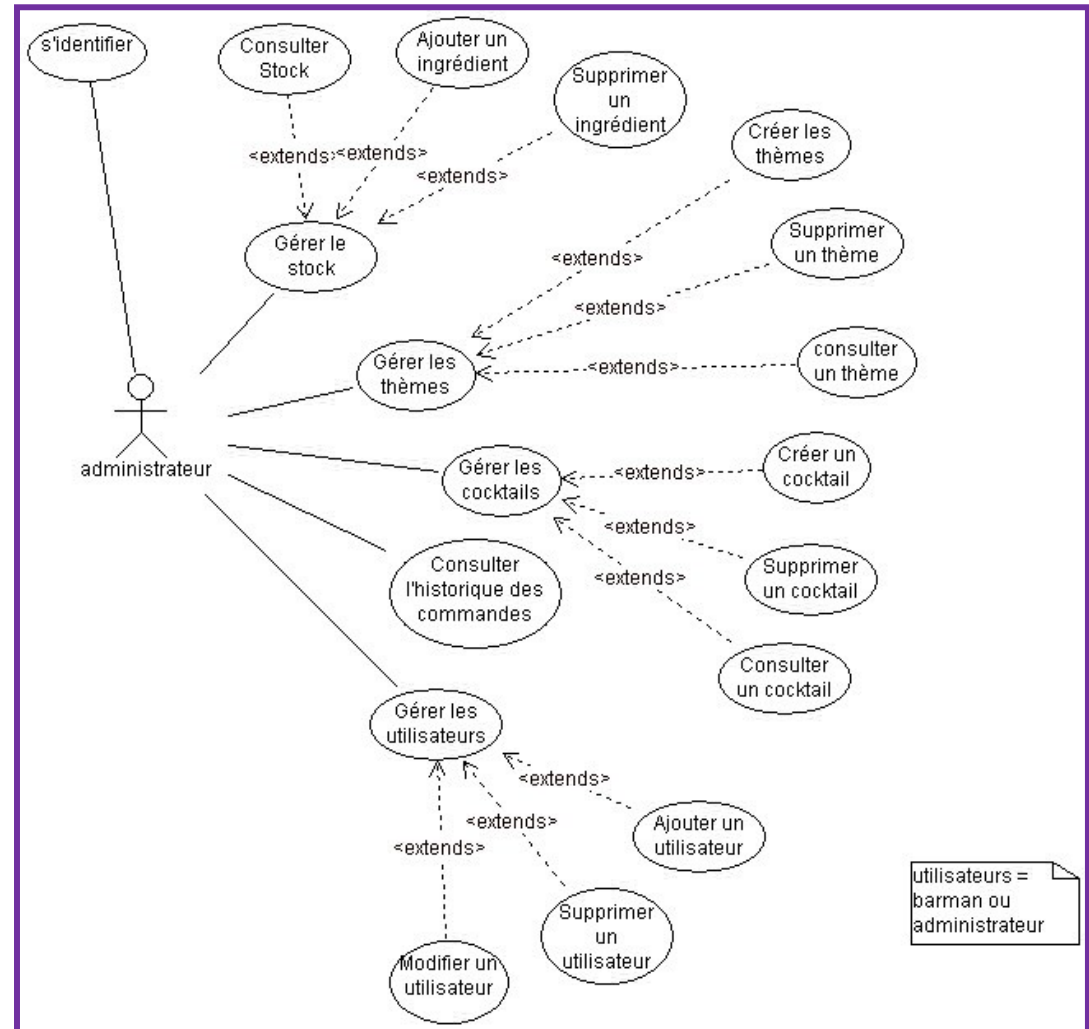
Use Case « client »



# Exemple : le barman virtuel

projet étudiant

## Les acteurs (formalisme des Use Case)



Use Case « administrateur »

# Exemple : le barman virtuel

projet étudiant

## Les scénarios

définition informelle

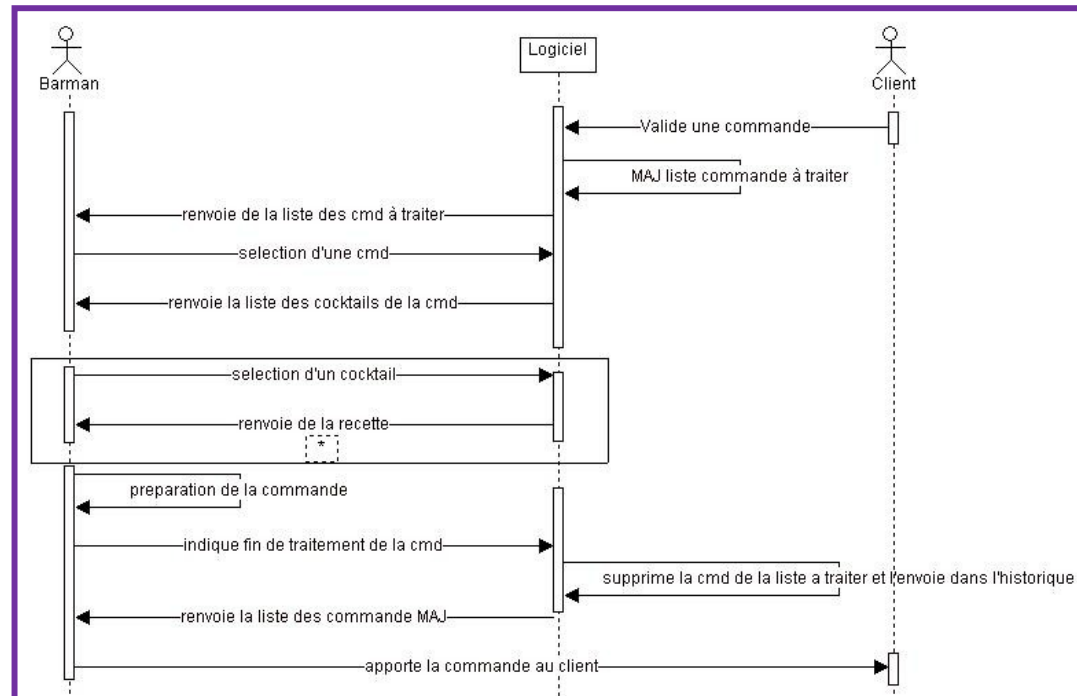
**Scénario :** Traitement d'une nouvelle commande.

**Déroulement :**

- 1) Le barman sélectionne une commande
- 2) Le barman sélectionne un cocktail de cette commande
- 3) Le barman consulte la recette de ce cocktail
- 4) Le barman réalise le cocktail
- 5) Le barman indique qu'il a fini de traiter la commande

*séquences d'interactions typiques  
puis cas particuliers et cas d'erreur*

diagramme de séquence



# Définition des besoins

- Un document qui rassemble (a minima)
  - la liste des cas d'utilisation
  - une description précise des scénarios
  - un glossaire

## I. Définition des objets

### 1) Cocktail

Un cocktail est défini par son nom, sa recette, son prix, et un texte descriptif.

### 2) Recette

Une recette est définie par un texte explicatif concernant la façon de réaliser le cocktail, ainsi que la liste des ingrédients.

### 3) Ingrédient

Un ingrédient est défini par son nom et son prix, son type, sa quantité et son unité.

### 4) Thème

- Un document qui doit être compréhensible par le client
- Un document qui doit pouvoir être utilisé pour une revue



# Un outil utile : la maquette

COMMANDE CLIENT COCKTAILS

Liste de Thèmes

-Alcools forts  
-Alcools fillettes  
-Alcools petits enfants  
. . . . .

Liste de cocktails

TRI: ☐ Tx alcool ☐ Prix ☒ Nom

-cocktail 1, liste ingrédients, 5,00€  
-cocktail 2, liste ingrédients, 5,00€  
-cocktail 3, liste ingrédients, 5,00€  
-cocktail 4, liste ingrédients, 5,00€  
-cocktail 5, liste ingrédients, 5,00€  
. . . . .

Détails de la commande

- cocktail 1, 3 5,00€  
- cocktail 1, 2 5,00€  
- cocktail 3, 2 5,00€

Quantité : + | -

TOTAL: 15,00€

Créer un cocktail de A à Z

commentaires

IMAGE

INGRÉDIENTS,

ajouter à la commande

Modifier le cocktail

Enregistrer cocktail

VALIDER

ANNULER

bouton qui apparait apres la creation d'un cocktail

ONGLETS

CK

HISTORIQUE

UTILISATEURS

THEMES / COCKTAILS

TE DES INGREDIENTS

QUANTITES

CATÉGORIE 1

LISTENIG

Qt 1

CATÉGORIE 2

LISTENING 2

Qt 2

AJOUTER AU STOCK

SUPPRIMER DU STOCK

ETES VOUS SURE ?

OUI NON

AJOUTER AU STOCK

<NOUVEAU>

BIERE WHISKY

...

Qt

VALIDER

ANNULER

MENU DEROULANT

ZONE DE TEXTE

# Puis, petit à petit, analyse/conception

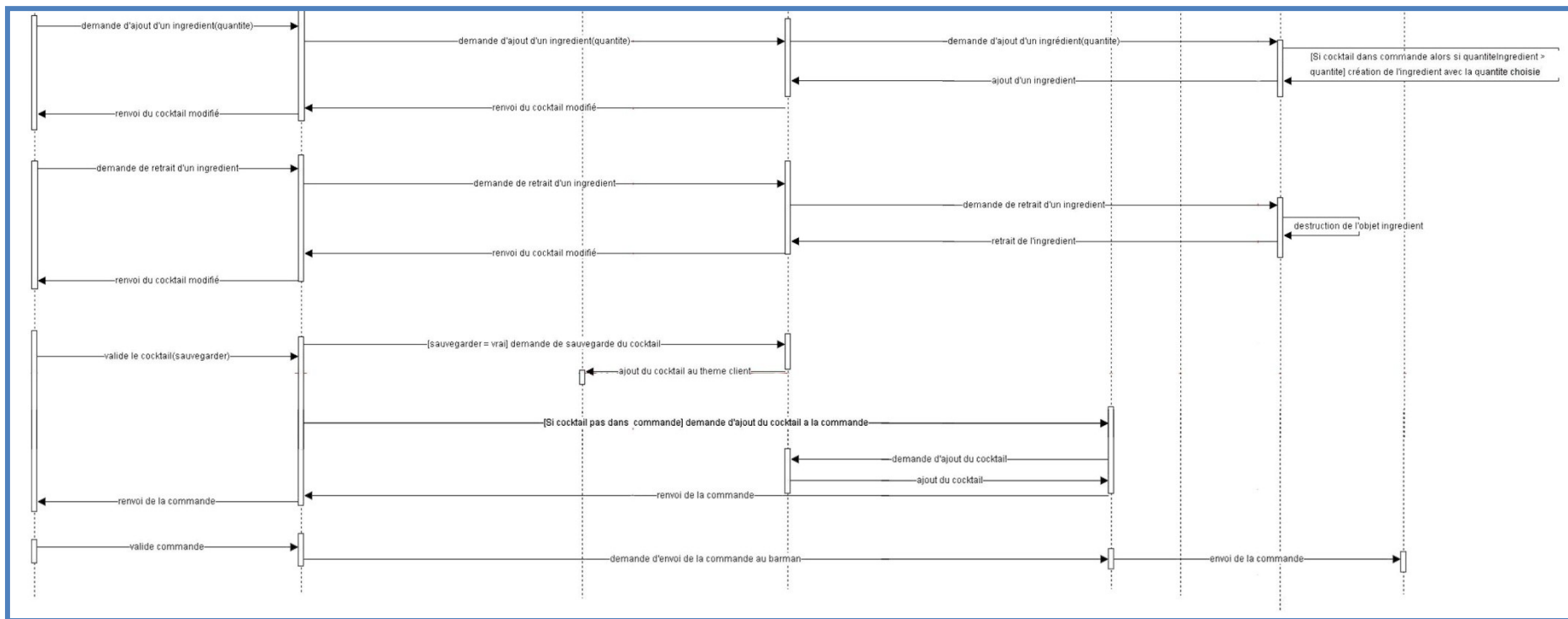
- *Gestion de certains aspects de « conception architecturale »*
  - *identification de l'architecture générale (client/serveur, 3/3...)*
  - *identification des sous-systèmes et des dépendances*
- Identification des classes principales et de leurs relations
- Élaboration des diagrammes (de plus en plus) détaillés
  - les classes (les objets) et leurs relations
  - les modèles dynamiques (diagrammes de séquences, de collaboration, d'états et d'activités)

diagramme de séquence → il y a des interactions (une collaboration), des « liens »  
analyse des interactions → définition des classes et des associations mises en jeu

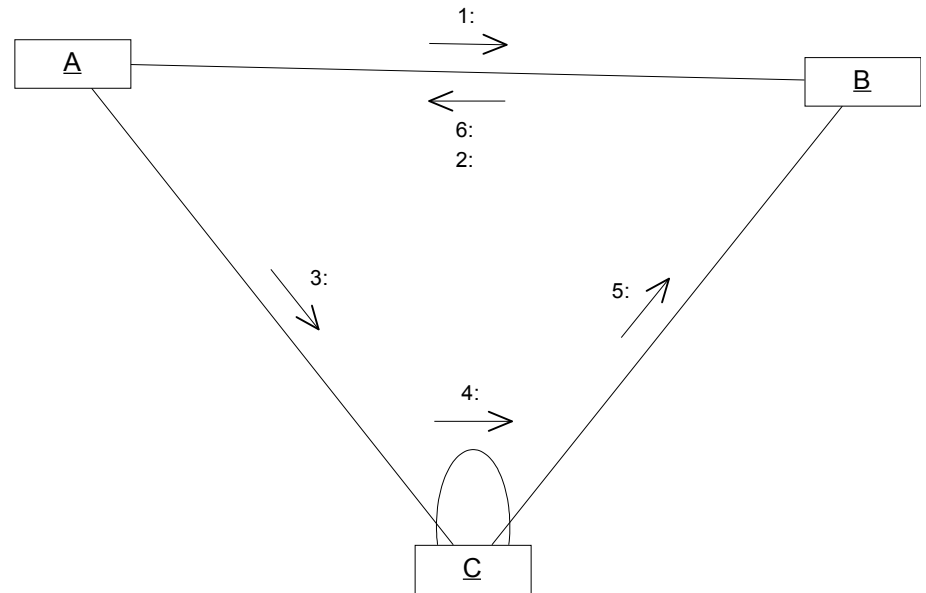
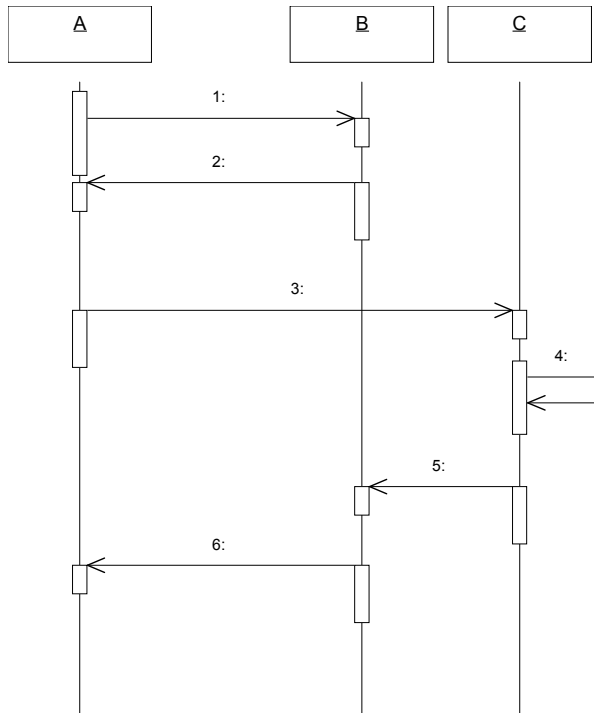
# Exemple : le barman virtuel

projet étudiant

## Les diagrammes de séquence détaillés



# Diagramme séquence / collaboration

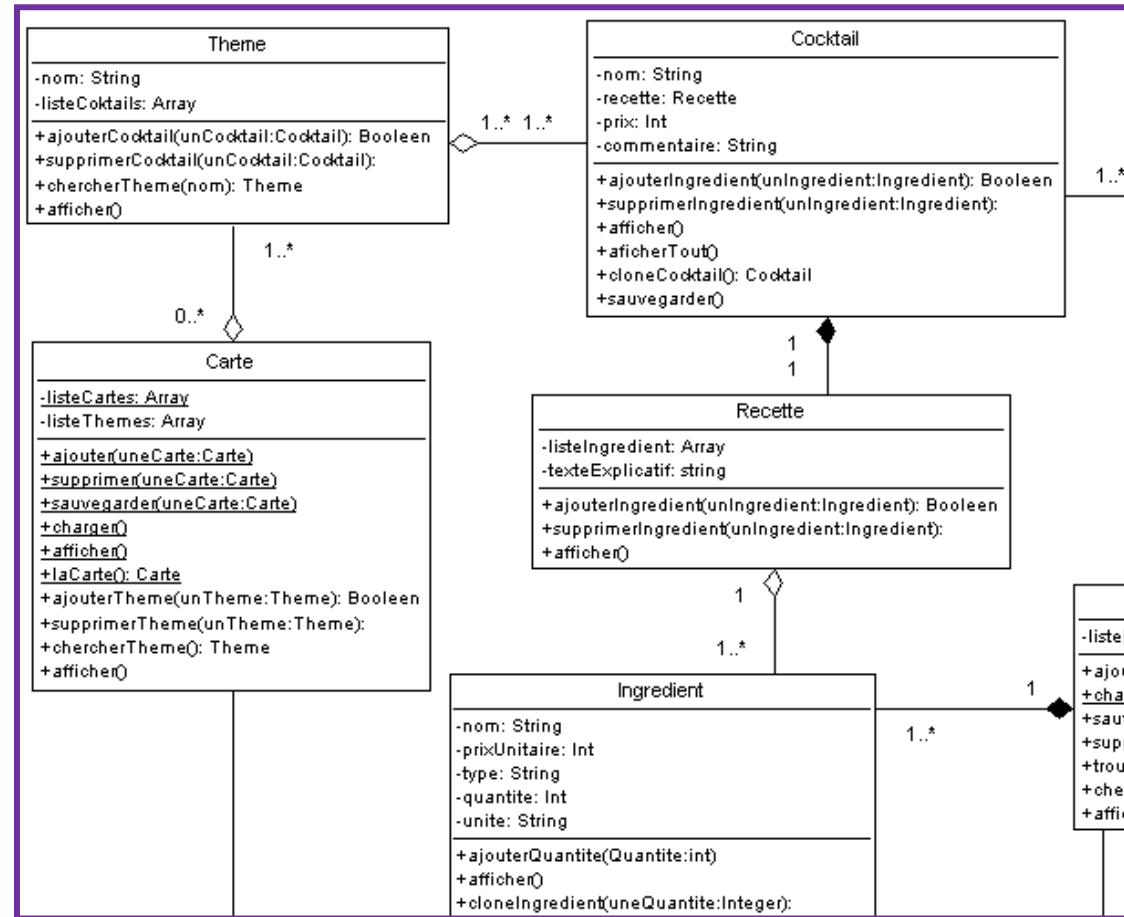
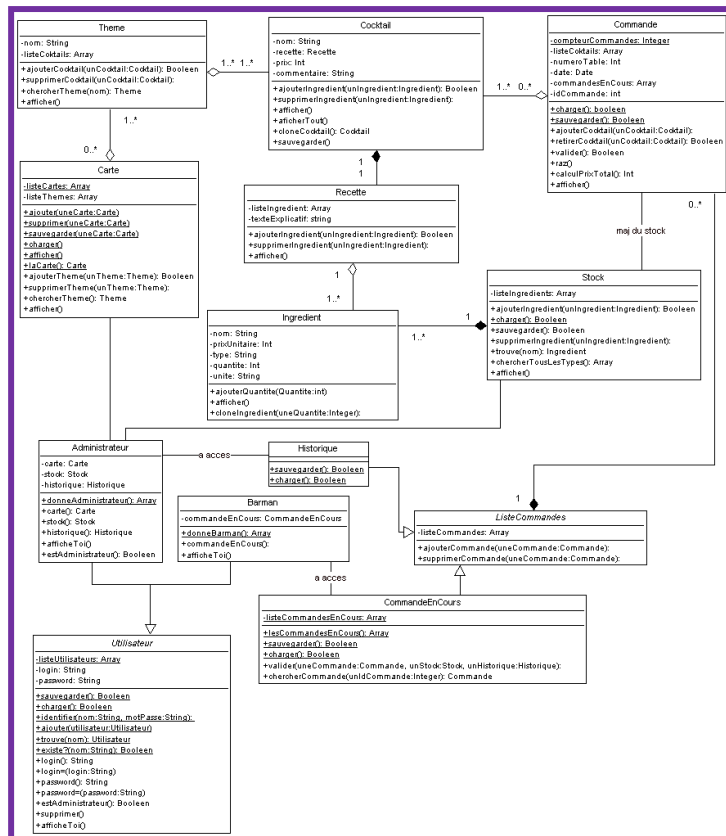


- pour documenter les cas d'utilisation  
analyser les événements qui peuvent se produire
- pour décrire de façon précise les interactions  
envois de messages (synchrone, asynchrone, réflexif – retour implicite ou explicite – contraintes – boucles, etc.) → syntaxe graphique précise

# Exemple : la barman virtuel

projet étudiant

## Les diagrammes de classe



# Puis, petit à petit ...

... on va gentiment vers le code

- définition des classes
- implantation des classes
- réalisation des scénarios par **collaboration** des objets

## Rappel :

- vision COO/POO de base (simpliste)
- les architectures modernes posent des problèmes spécifiques, et il y a des solutions spécifiques !

### 1) Cocktail

#### Description :

Un cocktail est défini par son nom, sa recette, son prix, et un texte descriptif.

#### Variable(s) de classe :

Aucune

#### Méthode(s) de classe :

- `Cocktail.creer(unNom,uneRecette,unCommentaire) : void`  
Lors de la création, le prix du cocktail est calculé directement en fonction des ingrédients que contient sa recette.

#### Variable(s) d'instance

Nom de la variable	Type	Description	Getter	Setter
<code>nom</code>	String	Nom du cocktail	oui	oui
<code>recette</code>	Recette	Lien vers la recette du cocktail	oui	oui
<code>prix</code>	Integer	Prix du cocktail	oui	non
<code>commentaire</code>	String	Texte de commentaire	oui	oui

#### Méthode(s) d'instance

- `afficher() : void`  
Méthode qui demande au cocktail d'afficher son nom.
- `afficherTout() : void`  
Méthode qui demande au cocktail de tout afficher (son nom, sa recette, son prix et son commentaire).
- `ajouterIngredient(unIngredient) : booléen`  
Méthode qui ajoute « unIngredient » à la recette du cocktail, ainsi qu'une mise à jour du prix du cocktail. Cette méthode renvoie True si l'ingrédient a été correctement ajouté, False sinon.
- `supprimerIngredient(unIngredient) : booléen`  
Méthode qui supprime « unIngredient » de la recette du cocktail, ainsi qu'une remise à jour du prix du cocktail. Cette méthode renvoie True si l'ingrédient a été correctement supprimé, False sinon.
- `cloneCocktail() : Cocktail`  
Méthode qui crée une copie du cocktail Cocktail. Ceci permet au client de modifier un cocktail existant  
Cette méthode renvoie le nouveau cocktail.

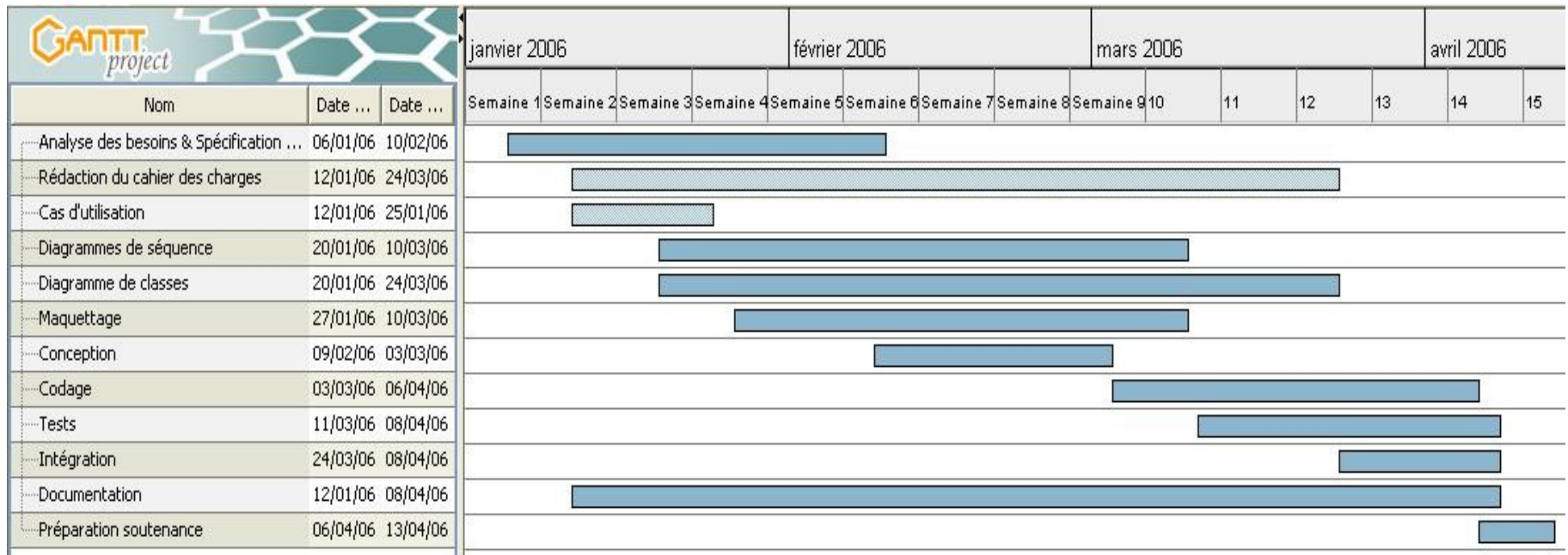
# COO / POO

## Continuum ...

- définition des besoins
  - analyse
  - conception
- ... autour de la notion de cas d'utilisation
- ... sur la base des mêmes notions
- ... avec des notations qui permettent de préciser petit à petit
- ... jusqu'à l'implantation

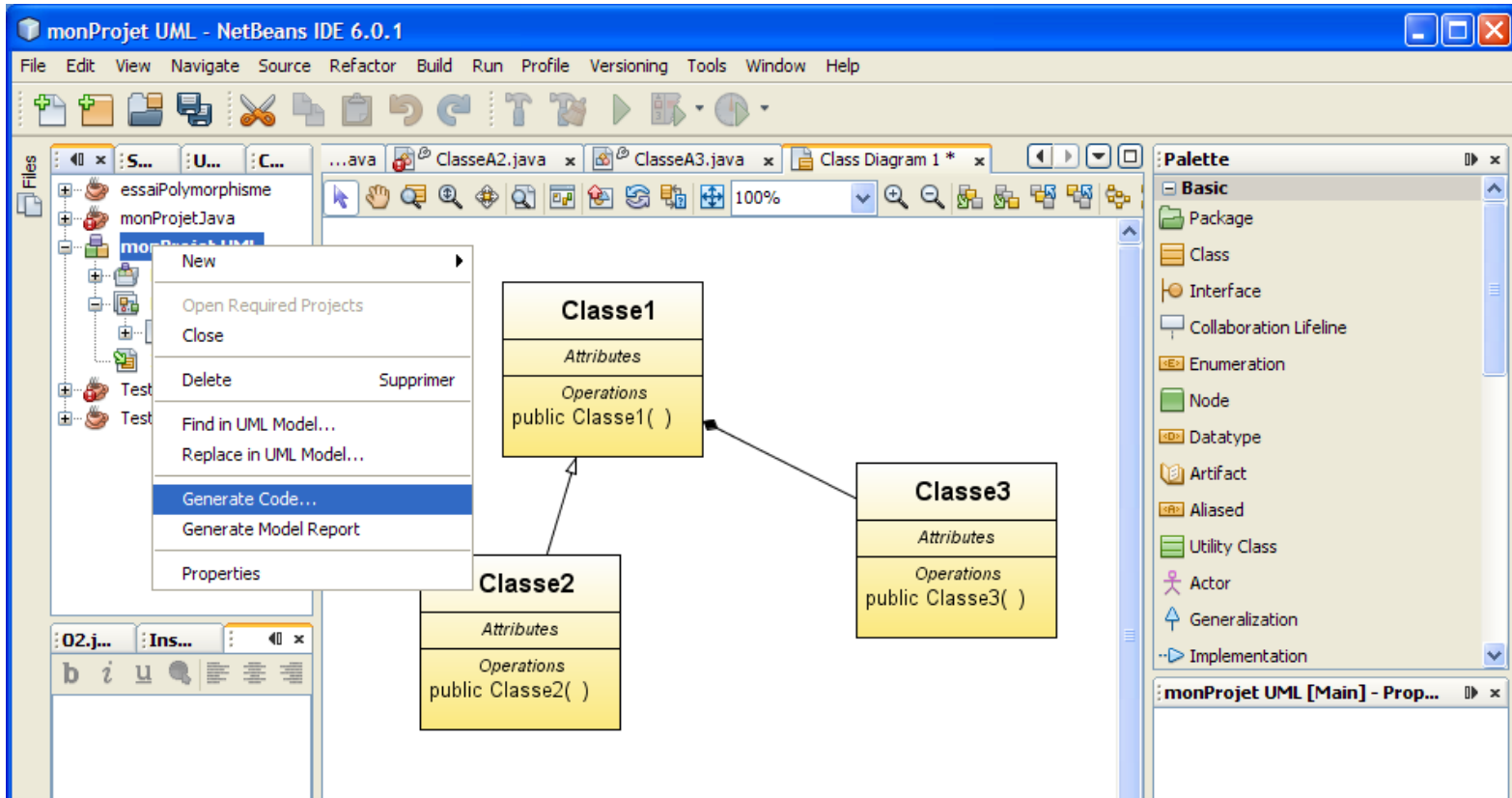
**pas de rupture conceptuelle**

# Gestion du temps





# Les IDE



# Un travail de CONCEPTION

- vers une activité de type « dessin industriel » ?



vous devez savoir faire un diagramme de classe UML

# Et pour réaliser tout cela ...

- Compétences en conception
    - architecture
    - classes
  - Compétences en utilisation des composants et des outils
    - frameworks
    - IDE
    - bibliothèques
  - Compétences sur les notions de base et leur exploitation
    - pour comprendre
    - pour faire
- focus du cours*