M1 info
GINF41B2 (Conception et Programmation Orientée Objet)

Cours #2
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 à des objets
  - 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 (1/2)

- L'héritage est fondé sur un processus naturel : la classification et la mise en facteur
- Idée :
  - 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) des classes sont transmises vers les sous-classes

vu dans l'autre sens : on cherche à regrouper les caractéristiques communes d'un ensemble de classes en les généralisant en une classe qui les « factorise » → mise en facteur

Classification, héritage et polymorphisme (2/2)

- Avantages :
  - évite la duplication de code (mise en facteur)
  - favorise la réutilisation
  - permet une forme de 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
- on peut utiliser un langage orienté objet … de façon « non-pertinente »

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

Exemple (et UML)

Intro UML (1/3)

- UML = Unified Modelling Language
  - UML est un formalisme qui permet de décrire une analyse orientée objet comme **un ensemble de vues complémentaires** d'un système

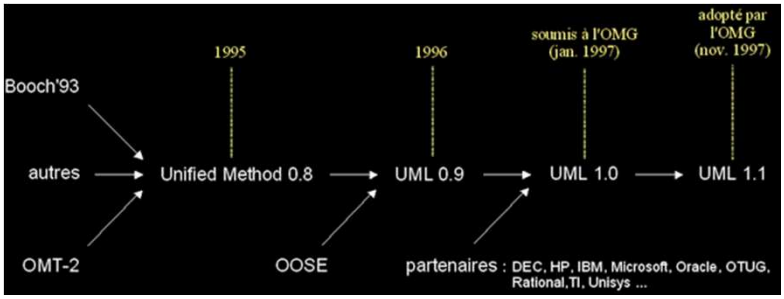
des notations adaptées aux différentes vues
et à leurs concepts sous-jacents

- 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

Intro UML (2/3)

- UML = fusion et synthèse des idées et notations de différentes approches orientées objet dont notamment :
  - OMT (James Rumbaugh)
  - OOD (Grady Booch)
  - OOSE (Ivar Jacobson)

http://uml.free.fr/



et cela continue (UML 2)

Intro UML (3/3)

- 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 (outils supports à l'utilisation d'UML : descriptions des modèles, génération de code)

UML n'est pas une méthode

c'est un formalisme qui permet de décrire des modèles, mais qui n'impose pas de processus pour les construire

UML -1 : 9 types de diagrammes

- Modéliser les vues statiques d'un système
  - **diagramme des cas d'utilisation** : représenter les fonctions du système du point de vue de l'utilisateur
  - **diagramme de classes** : structures statiques en termes de classes et de relations
  - diagramme d'objets : les objets et leurs relations
  - diagramme de composants : composants physiques d'une application
  - diagramme de déploiement : déploiement des composants sur les dispositifs matériels
- Modéliser les vues dynamiques d'un système
  - **diagramme de séquence** : représentation temporelle des objets et de leurs relations
  - diagramme de collaboration : représentation spatiale des objets, des liens et des interactions
  - diagramme d'états-transitions : comportement d'une classe en termes d'états
  - diagramme d'activités : comportement d'une opération en terme d'actions

vous devez savoir faire un diagramme de classe UML

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

- Point de départ : définir les acteurs et les cas d'utilisation pour
  - délimiter le système par rapport à son environnement
  - déterminer qui interagit avec le système et quelles fonctions il en attend
  - définir un glossaire commun
- Il est souvent intéressant de partir du modèle de l'entreprise, de ses « métiers » :
  - (type de) personnes intervenant de l'entreprise
    → un acteur potentiel dont on détermine les rôles

Définition des besoins

- Il faut identifier les acteurs en se posant des questions du type :
  - qui utilise le système ?
  - 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.

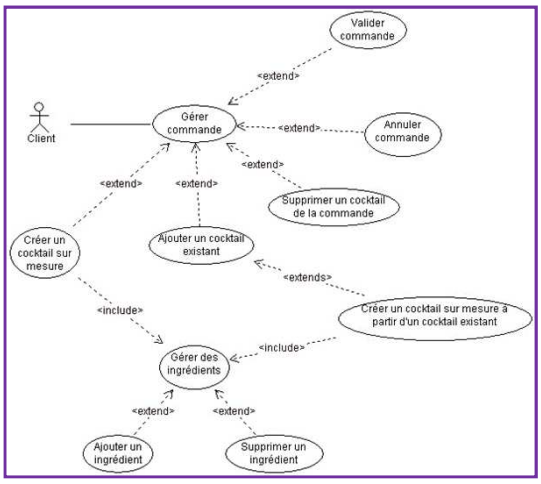
L'**administrateur** est un utilisateur enregistré.
Son rôle est de gérer les stocks, les cocktails, les thèmes ainsi que les utilisateurs.
Ses besoins sont de pouvoir gérer les cocktails, les thèmes et les utilisateurs (Barman ou Administrateur). Il doit aussi pouvoir gérer le stock et visualiser l'historique des commandes passées.

Exemple : le barman virtuel

projet étudiant

Les acteurs (formalisme des Use Case)

Use Case « client »

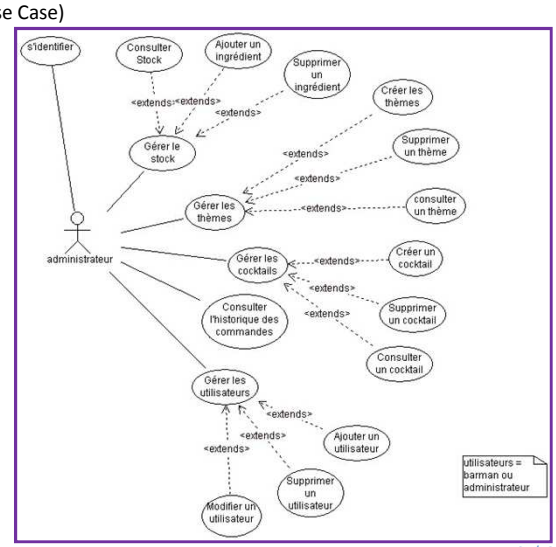


Exemple : le barman virtuel

projet étudiant

Les acteurs (formalisme des Use Case)

Use Case « administrateur »



Exemple : le barman virtuel

projet étudiant

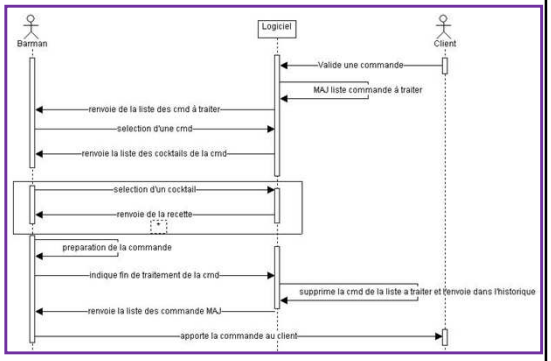
Les scénarios

définition informelle

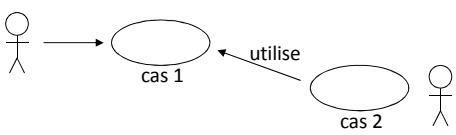
Scénario : Traitement d'une nouvelle commande.
Déroulement :

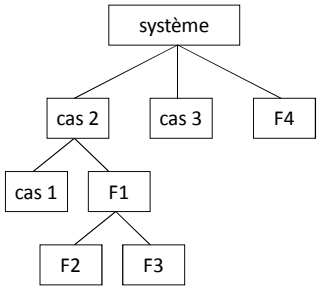
1) Le barman sélectionne une commande
2) Le barman sélectionne un cocktail de cette commande
3) Le barman consulte la recette de ce cocktail
4) Le barman réalise le cocktail
5) Le barman indique qu'il a fini de traiter la commande

diagramme de séquence

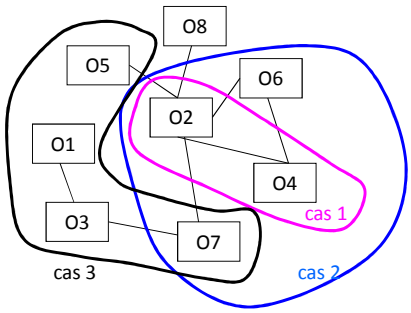


Rappel : Conception fonctionnelle / objet



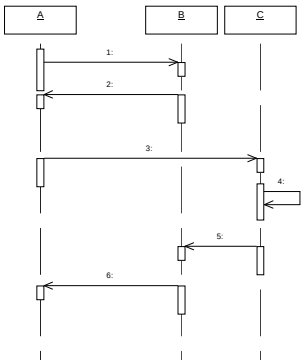


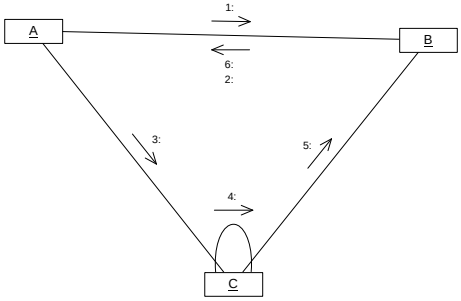
Conception fonctionnelle



Réalisation de Use Cases par collaboration d'objets

Diagramme séquence / collaboration





différentes utilisations :

<u>pour documenter les cas d'utilisation</u>
analyser les événements qui peuvent se produire

<u>pour décrire de façon précise les interactions</u>
envois de messages (synchrone, asynchrone, réflexif –
retour implicite ou explicite – contraintes – boucles, etc.)
→ syntaxe graphique précise

Définition des besoins

- Tout ceci se fait par cycles
  - on commence par les séquences d'interactions typiques, puis on prend en compte les diverses situations possibles et les cas d'erreur
  - on stabilise petit à petit une première description du système qui évoluera

- Objectif : document rassemblant
  - la liste des cas d'utilisation (tous les besoins fonctionnels, pas de redondance)
  - description précise des scénarios
  - documentation textuelle additionnelle

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

Définition des besoins

- Quand on est en phase d'identification des besoins
  - le formalisme de base est celui des diagrammes de cas d'utilisation
  - on va utiliser les diagrammes de séquences en « boîtes noires » pour comprendre puis documenter les scénarios

- Au fur et à mesure : construction d'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.

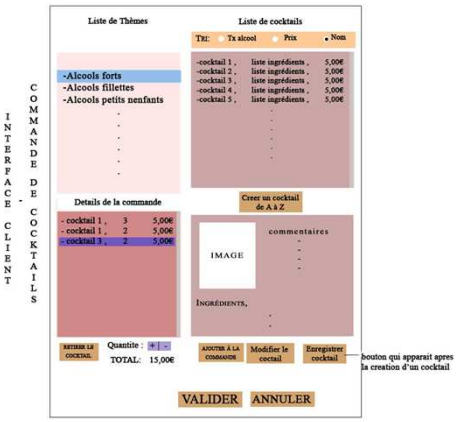
4) Thème

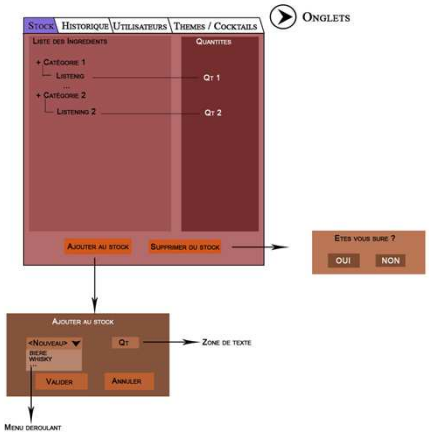
remarque : certains objets commencent à apparaître

- Le cas échéant, construire une maquette (aspects interface)

Prémices d'une maquette

ici, avec PowerPoint !





Puis, petit à petit …

- On précise les choses, on passe en « 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 entre sous-systèmes

(structuration en paquetages)

➡ **Identification des classes principales et de leurs relations**
  - on reste à un premier niveau très général, puis on précise

Puis, petit à petit …

- On commence à voir le système en « boîte blanche »

Les diagrammes sont détaillés :

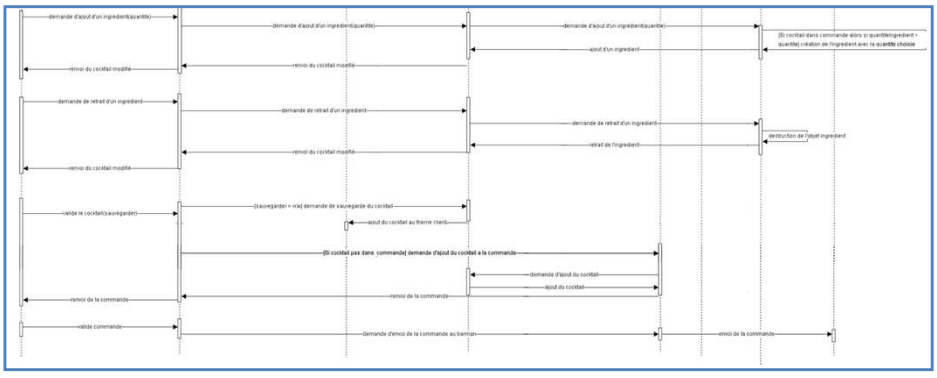
- les **classes** (les **objets**) principaux sont **identifiés, décrits, élaborés, raffinés**
- les modèles dynamiques (diagrammes de séquences détaillés, de collaboration, d'états et d'activités) amènent à définir, préciser, raffiner les **interactions entre objets et** les **besoins et responsabilités de chaque classe**

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

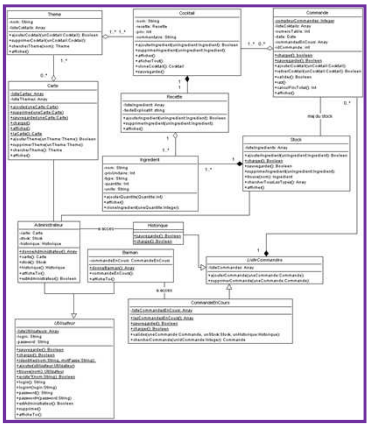
Les diagrammes de séquence détaillés

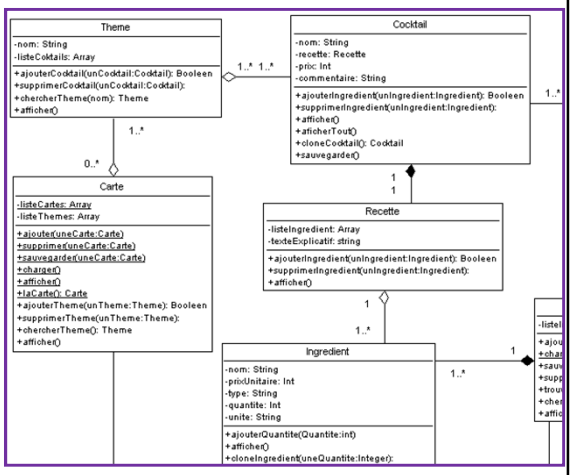


Exemple : la barman virtuel

projet étudiant

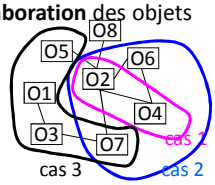
Les diagrammes de classe





Puis, petit à petit …

- … on va gentiment vers le code
  - définition des classes
  - implantation des classes
  - réalisation des scénarios par **collaboration** des objets



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
nom	String	Nom du cocktail	oui	oui
recette	Recette	Lien vers la recette du cocktail	oui	oui
prix	Integer	Prix du cocktail	oui	non
commentaire	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.

Rappel :

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

COO / POO

- **Continuum** ...
  - définition des besoins
  - analyse
  - conception

    ... autour de la notion de cas d'utilisation

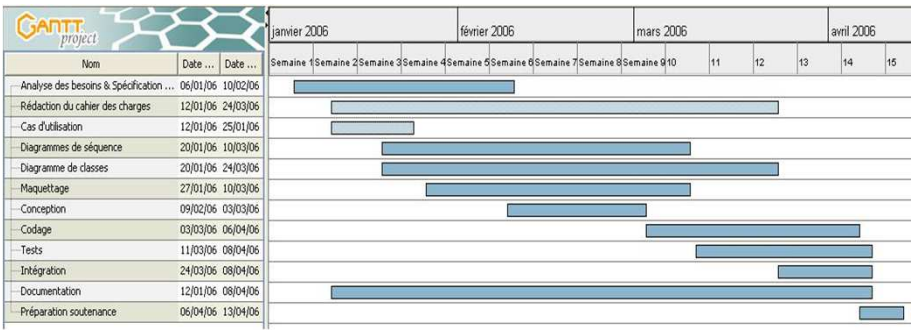
    ... sur la base des mêmes notions

    ... avec des notations qui permettent de préciser petit à petit

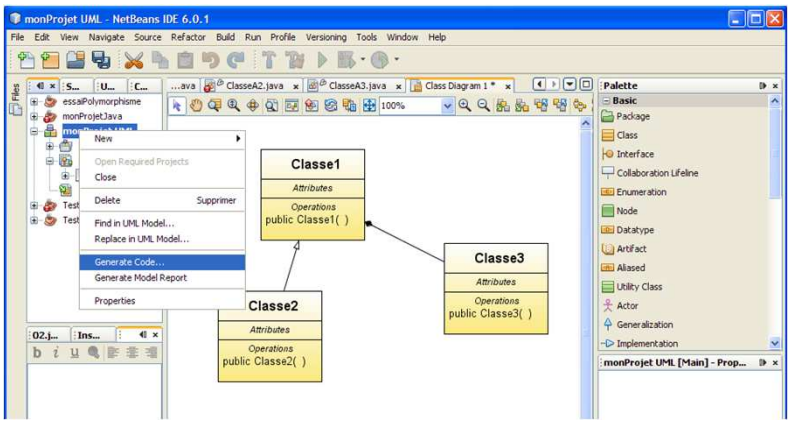
    ... jusqu'à l'implantation

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 les analystes / programmeurs dans tout ça ?

- 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