Génie logiciel

Conception

Louis-Edouard LAFONTANT

geodes

Université de Montréal

Conception

Après l'analyse, nous savons de quoi nous avons besoins pour répondre aux exigences.

⇒ *Question: Comment y arriver?*

- **Structurer la solution, indépendamment de l'implémentation finale.**
- Comprendre les forces et faiblesses de sa solution, sans voir le code.

Des exigences à la conception

- Une fois que les exigences sont comprises et claires, la **transformation** de l'analyse à la conception peut commencer
- Le *workflow* de conception détermine la **structure interne** du logiciel: **comment** il va atteindre ses buts
- Bonne conception contribue à la **qualité** du logiciel (fiabilité, flexibilité, modularité)

Types de conception

Conception **architecturale** (haut niveau)

- Définit la **structure et l'organisation générale** du logiciel
- Décrit les modules clés du domaine, les relations entre eux et contraintes

Conception **détaillée** (bas niveau)

- Réalise chaque **cas d'utilisation**
- Respecte le plan de la conception architecturale
- Décrit le **fonctionnement interne** de chaque module
- Prépare l'implémentation

Objectifs d'une bonne conception

- **Forte cohésion**
  Éléments ne sont pas réunis dans un même module par hasard, ils forment un tout pour réaliser une tâche
- **Faible couplage**
  Modules sont relativement indépendants : dépendent le moins possible des éléments d'autres modules
- **Abstraction**
  Décomposition intuitive exprimée en termes du problème, qui permet de se concentrer sur un module à la fois
- **Encapsulation** et **dissimulation d'information**
  Détails d'implémentation propices à changer sont cachés derrière une interface stable

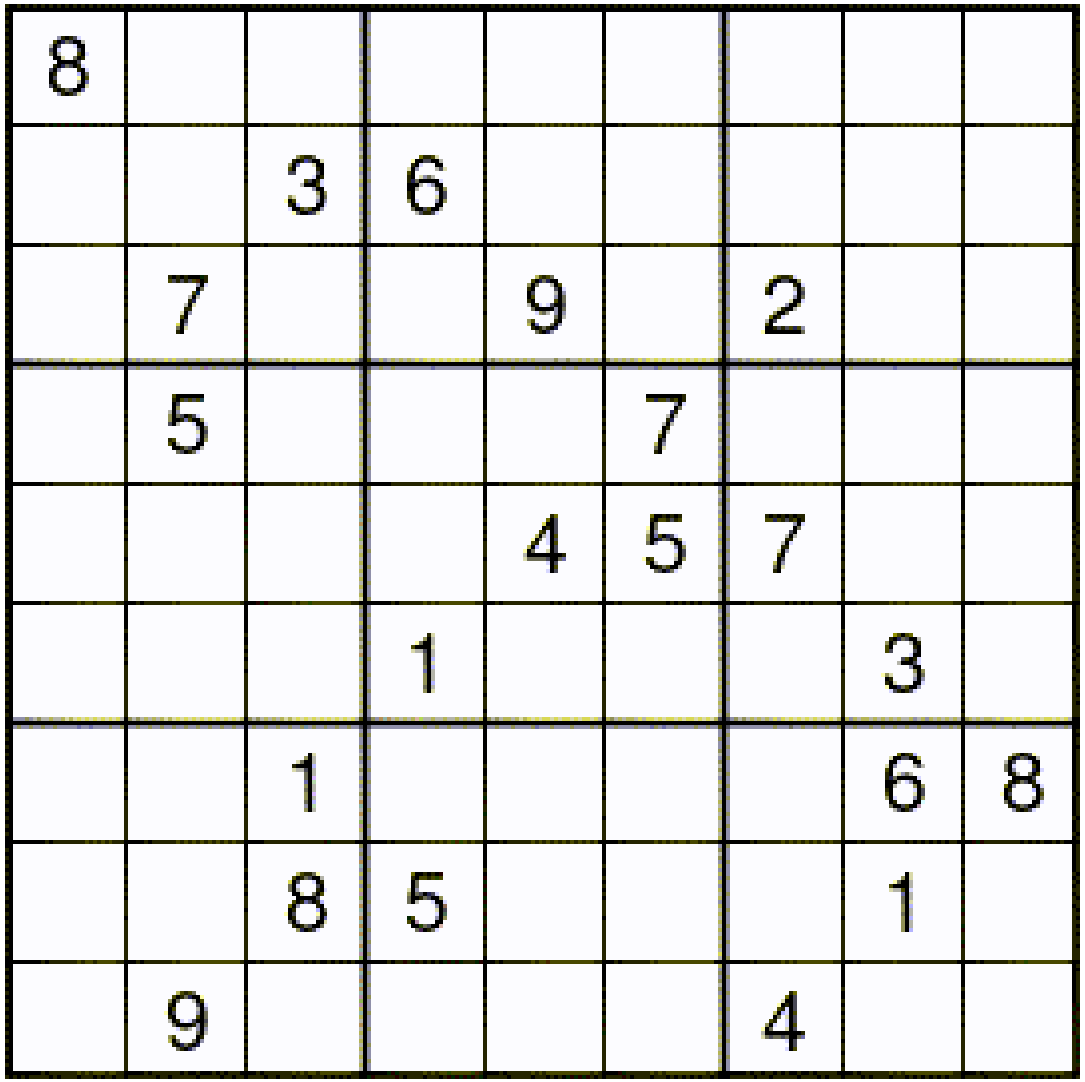
Conception évolutive

- Conception doit tenir compte
  - Des besoins **existants**
  - Des besoins **à venir**
- Obtenir une conception qui facilite l'adaptation du logiciel aux changements
  - Capacité d'**évolution**
  - **Anticipation** du changement
- Types de changements
  - Fonctionnalité, algorithme
  - Représentation des données
  - Environnement
  - Processus de développement

Décomposition du système

- **Décomposer** le système en **petits modules** qui sont individuellement plus faciles à concevoir et implémenter
- Si les sous-systèmes sont indépendants, ils peuvent être implémentés par des équipes travaillant en parallèle

⇒ Livraison plus rapide



Décomposition fonctionnelle

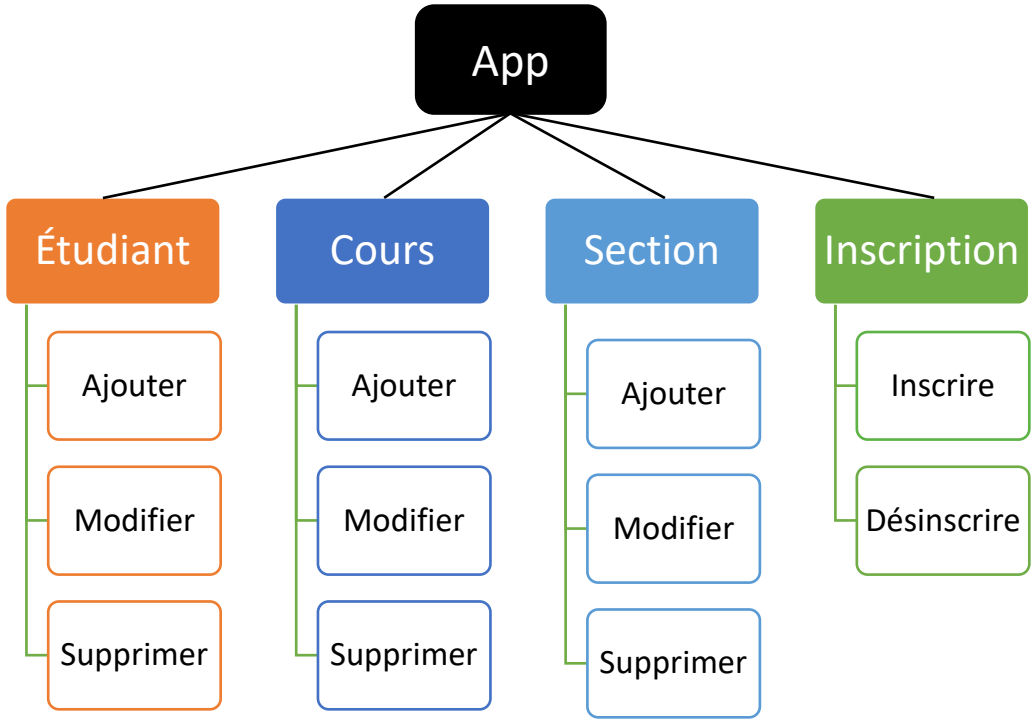
Décomposer le système en modules par fonctionnalité

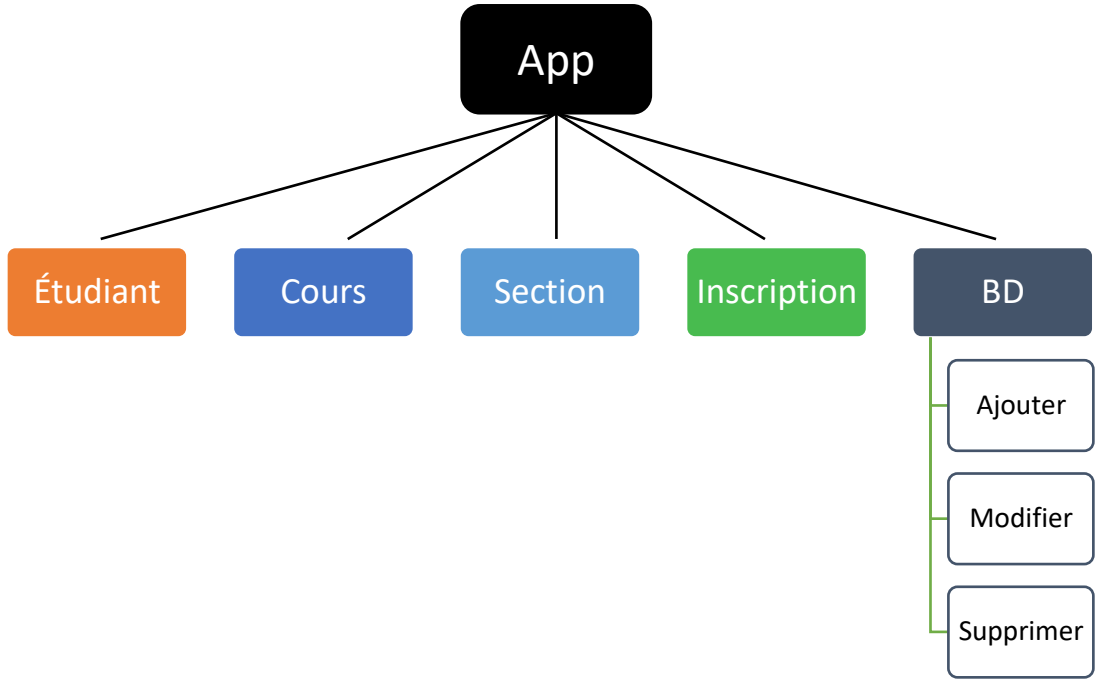
Exemple: Concevoir un système qui gère l'inscription à un cours. Les exigences sont:

- Modifier et supprimer des étudiants de la base de données
- Modifier et supprimer des cours de la base de données
- Ajouter, modifier et supprimer des sections d'un cours
- Inscrire et désinscrire des étudiants d'une section

Quels sont les modules nécessaires et comment décomposer le système ?

Décomposition en modules



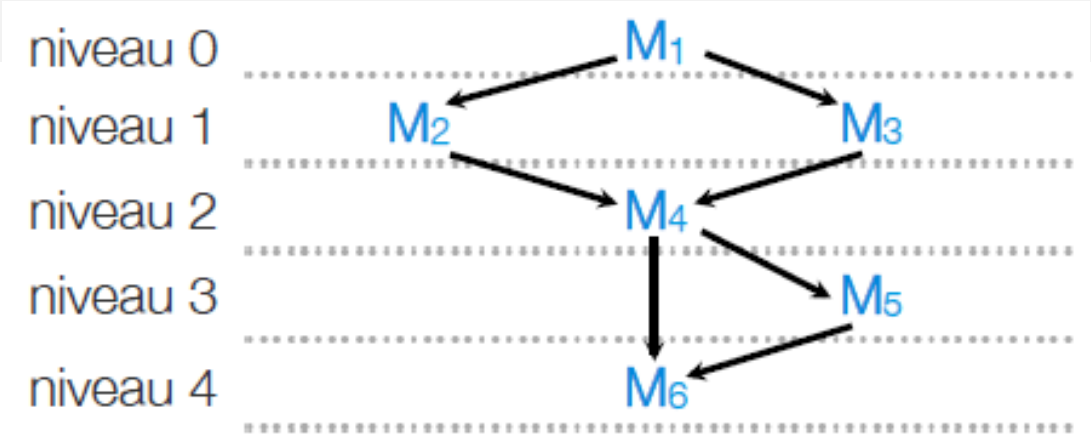


Qu’est-ce qu’un module?

- Unité fournissant des ressources et/ou des services
  - Morceau de code indépendant muni d’une interface bien définie avec le reste du système
  - Exemple: méthode, classe, objet, paquet, {…}
- **Système** est **partitionné** en **modules** distincts mis en relation
  - **Relation « spécialise »** : raffine les détails du module plus général
  - **Relation « contient »** : modules contenus dans un module parent
  - **Relation « utilise »** : fait référence/dépend d’un autre module

Établir une hiéarchie

- Facilite la compréhension de la structure par niveau d'**abstraction**
- Facilite les **tests** unitaires
- Permet de bâtir un système partiel mais fonctionnel: de manière **incrémentale**



Interface et implémentation de modules

Interface: **partie publique**

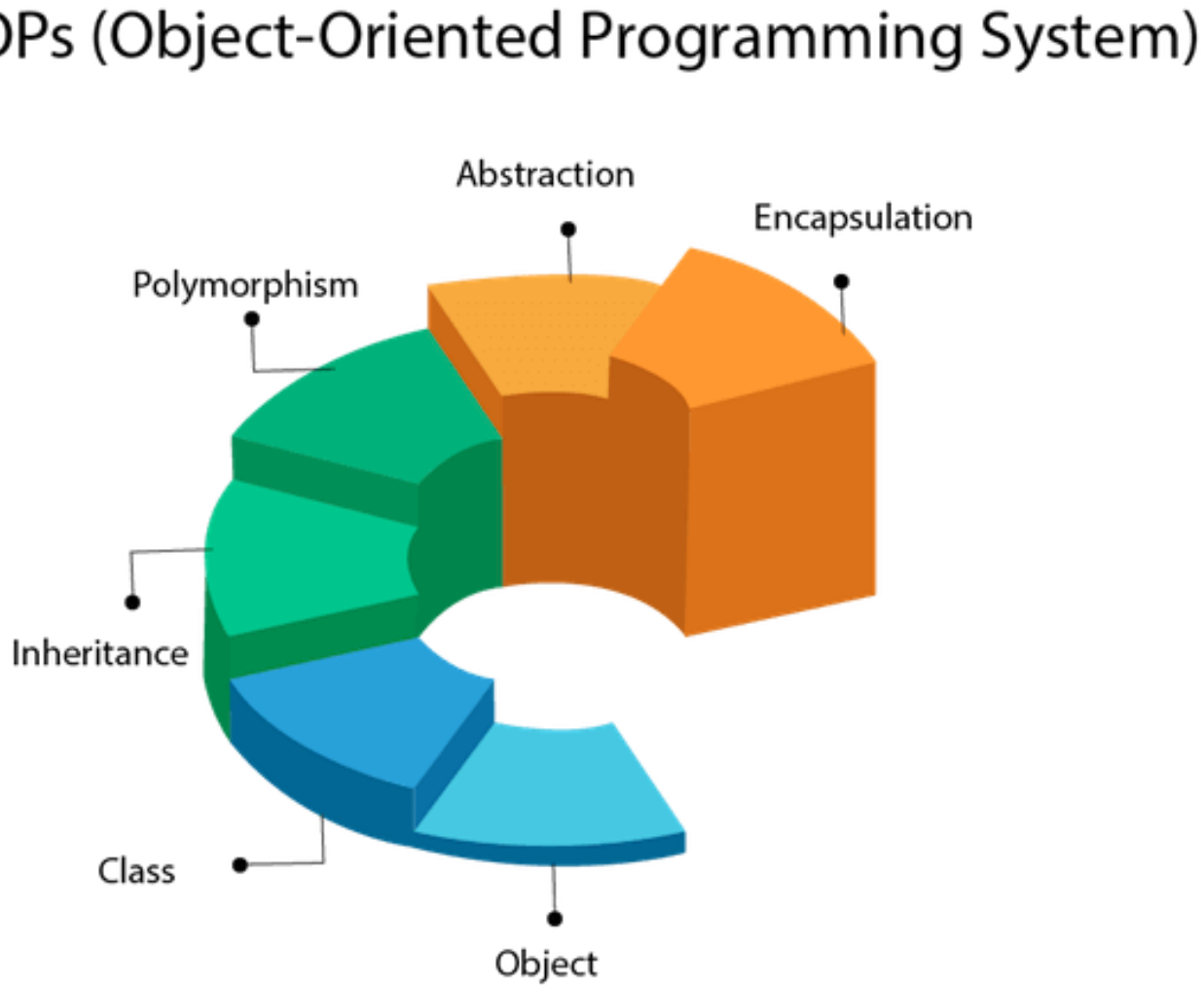
- ***Quoi offrir ?*** Analyse et conception
- Ensemble des ressources (opérations, attributs…) rendues accessibles aux autres modules (« clients »)
- Conception d'un module ne nécessite que les interfaces des modules qu'il pourra utiliser

Implémentation : **partie privée**

- ***Comment le réaliser ?*** Conception et implémentation
- Façon dont les ressources sont concrètement représentées et réalisées dans le module

Java, C#, Python, Kotlin, Swift, PHP…

Conception orientée-objet (OO)



Conception orientée-objet

But: Concevoir le logiciel en termes d'objets

Instances des classes extraites du modèle d'analyse

ALTERNATIVES

❑ Langage non OO (ex: C, Ada)

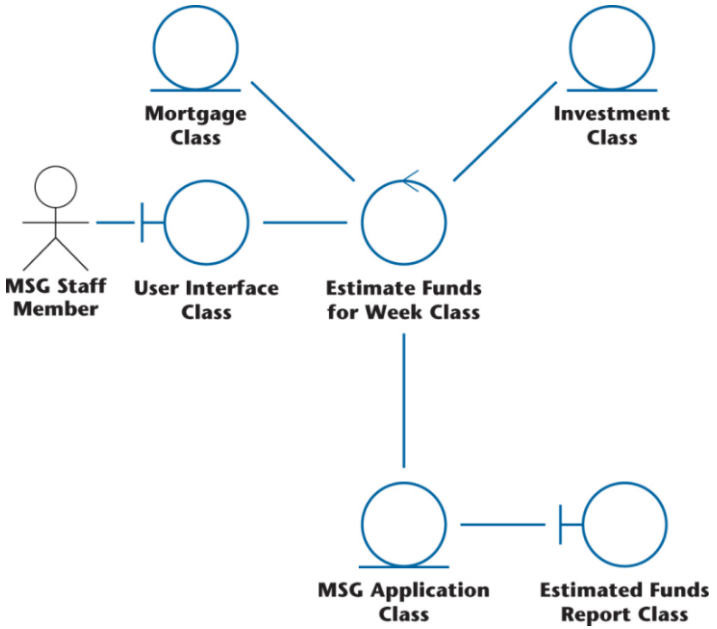
⇒ Conception de type de donnée abstrait

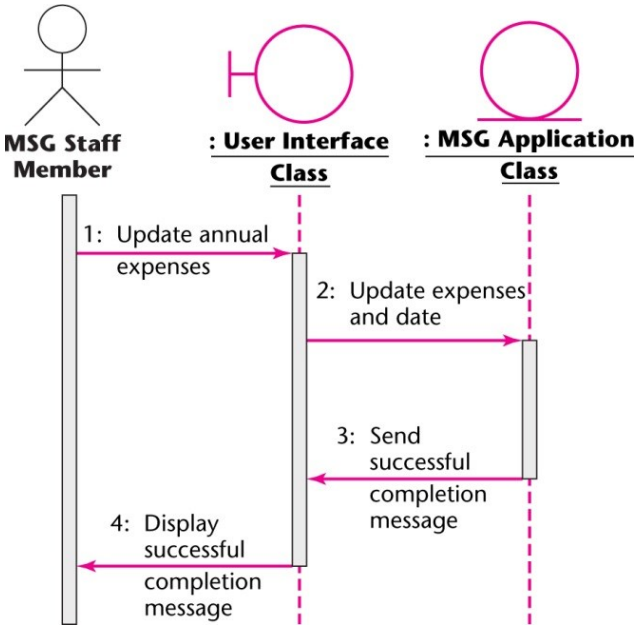
❑ Langage non typé (ex: FORTRAN, Javascript)

⇒ Conception par encapsulation des données

Communication entre les classes

- Déterminer les messages envoyés entre les objets
- Si objet A envoie message `M` à B, alors B doit avoir une méthode `m()` qui peut recevoir `M`



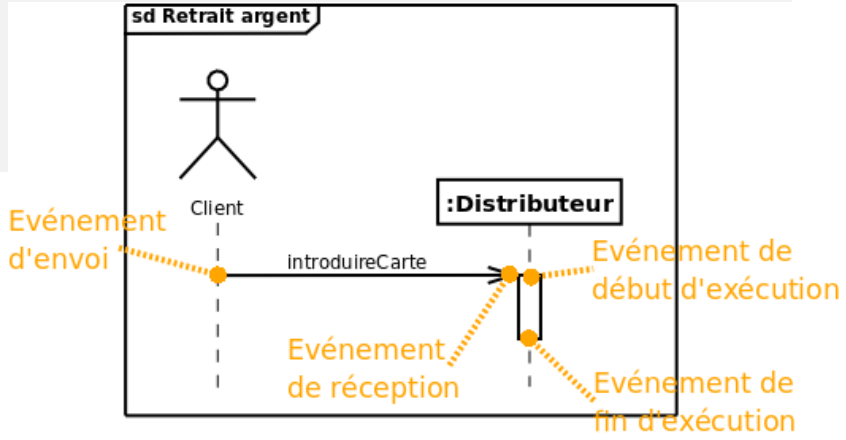


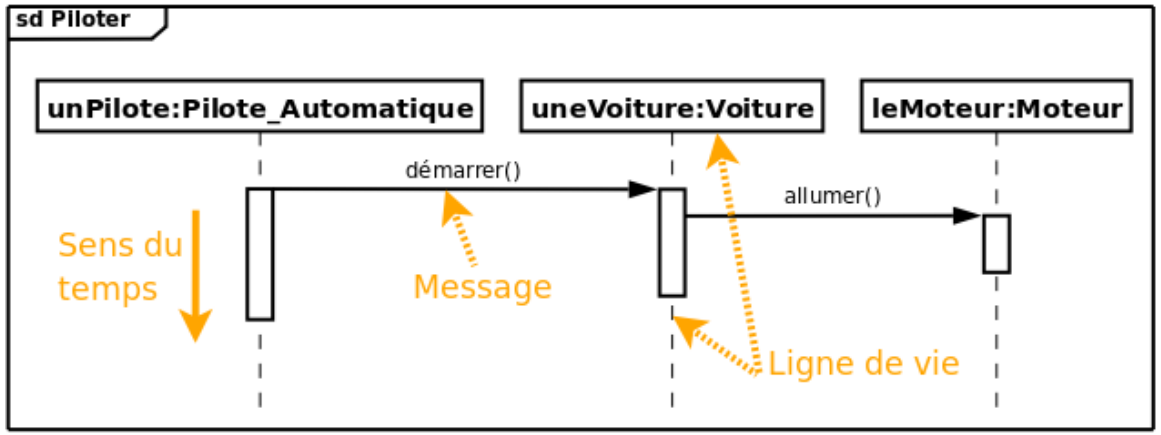
Collaboration entre classes

Construire un **diagramme de séquence UML**

1. Placer les objets interfaces et entités sur le diagramme
2. Pour chaque tâche représentée par un objet contrôle, identifier les **messages** qui doivent être échangés entre les objets pour réaliser la tâche
3. Faire correspondre la **signature d'une opération** à chaque message
4. Ajouter les flèches d'envois de messages correspondantes dans le diagramme de collaboration ou de séquence
5. Mettre à jour le diagramme de classes : ajouter les opérations identifiées

Diagramme de séquence





- Décrit les **interactions entre les objets**, instances du diagramme de classe
  - Montre la **trace** des messages échangés
- **Ordre** dans lequel les méthodes sont invoquées
- **Un diagramme de séquence par CU**
  - Des fois plus si on veut séparer les scénarios alternatifs
- Doit être **cohérent avec le diagramme de classe**

Diagramme de classe

- Vue détaillée du design
- Plus de **classes** que dans le modèle d'analyse
  - N'ajouter que si nécessaire
- Déterminer les **attributs** de chaque classe
  - État des objets
  - Propriétés caractéristiques
  - Type et format des attributs
- Identifier les **méthodes** à assigner à chaque classe
  - Opérations publiques nécessaires pour réaliser les CU
  - Méthodes privées/protégées pour les aider

Identifier les classes

- Classes du modèle d'analyse et du domaine
- Quels **objets** envoient ou reçoivent des **messages** ?
  - Invocateur et receveur
- Quelle **information** est nécessaire pour chaque opération ?
  - Paramètres, types de donnée abstraite
- Quelle information est **retournée** par chaque opération ?

Identifier les attributs

- De quelle information un objet a-t-il besoin pour effectuer une action ?
  - Données pour créer l'objet
  - Données pour calculer ou traiter un message reçu
  - Données à passer en paramètre à une méthode d'un autre objet
  - Information caractéristique de l'état de l'objet
- Types d'attributs
  - Types **primitifs**: booléen, nombre, texte
  - Types **abstraits**: DateTime, classes du diagramme de classes (typiquement modélisé comme association)

Identifier les méthodes

- Quelles **actions** le système doit-il effectuer ?
  - Au moins **une** méthode publique par CU
- Dans chaque CU, identifier le **sujet** de chaque action
  - Sujet est un **acteur: humain ou autre système**
  - Sujet est un **objet agissant sur un acteur**
  - Sujet est un **objet agissant sur un autre objet**