Modélisation Objet

ENSIBS Informatique 3ème année 2023-2024

Mazen EL-SAYED

mazen.el-sayed@univ-ubs.fr

Organisation du cours

- 24h (CM/TD/TP)
- Evaluation
  - Devoir surveillé
  - Un projet

Compétences visées

- Connaître les concepts fondamentaux de la modélisation orientée objet
- Maîtriser la notation et les techniques de UML (Unified Modeling Language) pour la modélisation statique en termes de diagrammes de classes et d'objets
  - Un deuxième module est prévu en S7 pour la suite de ce module
- Maîtriser le langage de contraintes OCL (Object Constraint Language) de UML et le langage d'actions OCL-Impératif pour la modélisation de logiciels
- Savoir utiliser un outil de modélisation (USE) , type Modeleur UML avec OCL et OCL-Impératif, pour réaliser une conception du logiciel

Plan

1. Introduction à la Modélisation Orientée Objet des logiciels
2. Le langage de modélisation unifié UML (Unified Modeling Language) pour la modélisation statique
   - Diagrammes de classes et d'objets
3. Le langage de contraintes OCL (Object Constraint Language) pour UML
4. Conception des applications avec UML, OCL et OCL – impératif (en utilisant l'outil USE)

Références

- UML 2.0
  - Benoît Charoux, Aomar Osmani, Yann Thierry-Mieg
  - Editions Pearson, Education France (2008)
- Modélisation et Conception Orientée Objet avec UML 2
  - Michael Blaha, James Rumbaugh
  - Editions Pearson, Education France (2005)
- Cours de Jean-Marie Favre, IMAG http://megaplanet.org/jean-marie-favre
- Cours de Pierre Gérard https://lipn.univ-paris13.fr/~gerard/
- CoursUML-ESI4-ITIS-EVRY https://sites.google.com/site/umlesi4itisevry/home/Cours/
- Cours de Prof. Flavio Oquendo

Chapitre 1 : Introduction à la Modélisation Orientée Objet des logiciels

Cycle de vie d’un logiciel

- Pour obtenir un logiciel de qualité, il faut en maîtriser le processus d'élaboration.
  - La vie d'un logiciel est composée de différentes étapes de développement.
  - La succession de ces étapes forme le cycle de vie du logiciel.

Etapes du développement

- Expression des besoins
  - Discuter avec le client afin de comprendre de quoi il a besoin.
  - Déterminer les fonctionnalités du logiciel.
  - Sans parler de comment les réaliser.
- Analyse : Comprendre les besoins et les décrire dans le système
- Conception : s'accorder sur la manière dont le système doit être construit
  - Déterminer la façon dont le logiciel fournit les différentes fonctionnalités recherchées. (comment)
  - On propose une solution particulière
- Implémentation : Codage du résultat de la conception
- Test : Le système est il conforme au cahier des charges?

Principes de la modélisation objet

- Pour construire un système ou une application complexe, il est nécessaire d'en faire des représentations simples, axée sur un point de vue particulier.
  - On parle alors de modèles et de modélisation.
- Un modèle est une représentation partielle et simplifiée de la réalité
  - Les objets du modèle sont des abstractions d'objets du monde réel.
    - Abstraction : Retenir les propriétés pertinentes et ignorer les détails non intéressants .
  - Vue subjective et simplifiée d'un système

Principes de la modélisation objet

- Le modèle doit être connecté au monde réel
- Un modèle peut être exprimé avec différents niveaux de précision
  - Plus de précision ⇒ modèle plus détaillé
- Un simple modèle n'est pas toujours suffisant, il y a plusieurs façons de voir un système.
  - Besoin de plusieurs modèles
- Pour partager un modèle avec d'autres personnes, il faut s'assurer au préalable :
  - que les personnes comprennent les conventions de représentation et les notations utilisées,
  - qu'il ne pourra pas y avoir d'ambiguïté sur l'interprétation du modèle.

Principes de la modélisation objet

De quoi a-t-on besoin pour réaliser des modèles orientés objet ?

- Un langage de modélisation
  - Notation claire
  - Sémantique précise
- Le langage UML est le langage de modélisation orientée objet le plus connu.
  - Il propose 13 diagrammes
- Dans la suite du cours, nous abordons deux des diagrammes proposés dans ce langage
  - Les autres diagrammes seront vus dans un autre module en S7.

Introduction au Langage UML

Introduction

- UML : Unified Modeling Language
- Langage unifié pour la modélisation objet
- Langage visuel utilisant une notation graphique

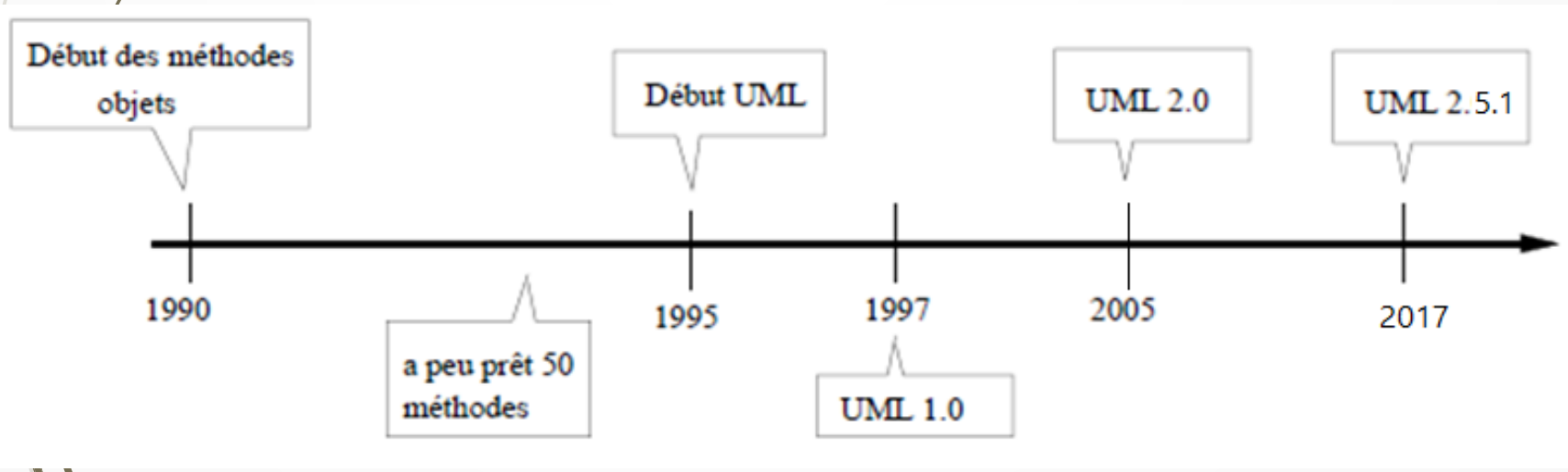
- Notation standard pour la modélisation d'applications à base d'objets
  - Mais utilisable dans de nombreux autres contextes de conception ou spécification
    - Exemple : schéma de BDD

Introduction

- UML a été « boosté » par l'essor de la programmation orientée objet.
- Historique de l'approche orientée objet :
  - Simula (1967),
  - Smalltalk (1976),
  - C++ (1985),
  - Java (1995),
  - …
- UML permet de modéliser une application selon une vision objet, indépendamment du langage de programmation.

Historique

- UML hérite principalement des méthodes objets OOD (Booch), OMT (Rumbaugh) et OOSE (Jacobson)
- Normalisé par l'OMG (Object Management Group) en 1997



Les diagrammes UML

- Difficulté d'appréhender complètement un système complexe à partir d'une seule perspective,
  - UML propose un ensemble de 13 diagrammes :
  - UML 1.x proposait 9 diagrammes
  - UML 2 a enrichi les concepts des diagrammes existants et a ajouté 4 nouveaux diagrammes

- Diagrammes structurels ou statiques(6)
  - De classes (class diagram)
  - D'objets (object diagram)
  - De composants (component diagram)
  - De structure composite (composite structure diagram)
  - De déploiement (deployment diagram)
  - De paquetages (package diagram)

Les diagrammes UML

- Diagrammes de comportement (7)
  - De cas d'utilisation (use case diagram)
  - D'activité (activity diagram)
  - D'états-transition (state diagram)
  - Diagrammes d'interaction
    - De séquence (sequence diagram)
    - Vue générale d'interaction (interaction overview diagram)
    - De communication (communication diagram)
    - De temps (timing diagram)

Les diagrammes UML

- Ces diagrammes permettent de définir une application selon plusieurs points de vue
  - Fonctionnel (cas d'utilisation)
  - Statique (classes, objets, structure composite)
  - Dynamique (séquence, états-transition, activité, interaction, communication, temps)
  - Implémentation (composants, déploiement, paquetage)

Les diagrammes UML

- UML est une grande boîte à outils, et comme toute boîte à outils..
  - On n'utilise pas tout …
  - Certains outils servent plus souvent que d'autres …
  - Dans chaque situation il faut choisir le bon outil !

Les diagrammes UML

- Les diagrammes seuls ne permettent pas de définir toutes les contraintes de spécification requises
  - Utilisation du langage textuel de contraintes OCL en complément
  - S'applique sur les éléments de la plupart des diagrammes

Le langage UML

- Il existe de nombreux outils (modeleur) pour faire de la modélisation UML :
  - Rational Rose,
  - Objecteering,
  - Together,
  - StarUML,
  - ArgoUML,
  - Eclipse Papyrus, ….

Les concepts de l’approche par objets

Concepts fondateurs de l'orientée objet

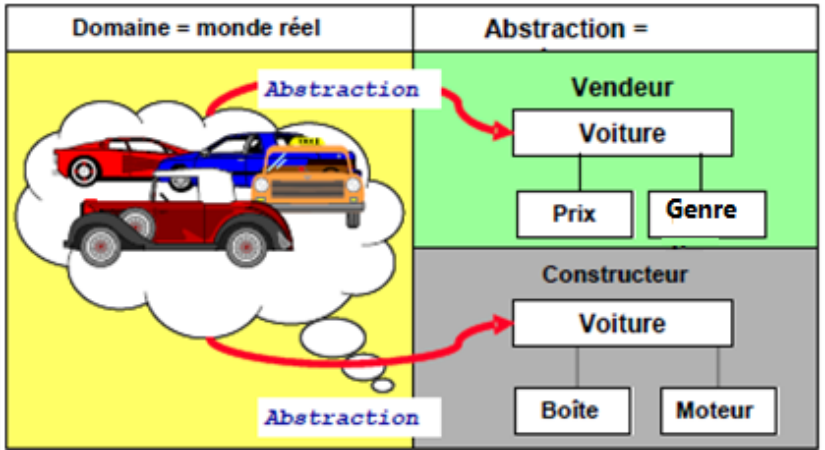
- Objet
- Classe (type)
- Héritage
- Polymorphisme
- Encapsulation

Les Objets

- Un objet (en méthodologie orientée objet) est une representation d'un objet du monde réel qui
  - peut une réalité vivante comme un être humain, un animal,...
  - peut une réalité matérielle comme une voiture, une bouteille,...
  - peut une réalité immatérielle ou abstraite comme une idée, une dette, un compte bancaire, …
- Les objets logiciels définissent une représentation simplifiée des objets du monde réel.
  - On parle d'abstraction d'objets du monde réel.

Abstraction

- Une abstraction est un résumé, une mise à l'écart des détails non pertinents dans un contexte donné (point de vue particulier).
  - Retenir les propriétés pertinentes et
  - ignorer les détails non intéressants.
- Une abstraction peut être faite à plusieurs niveaux
  - Propriétés à retenir et celles à ignorer
- Exemples :



Les concepts fondateurs : Objet

- Objet = État (attributs)+ Comportement (méthodes) + Identité
- L'état d'un objet :
  - regroupe les valeurs instantanées de tous les attributs d'un objet
  - évolue au cours du temps
- Le comportement d'un objet :
  - décrit les actions et les réactions d'un objet
  - se représente sous la forme d'opérations (méthodes).
    - Le comportement d'un objet est dicté par le code de ses méthodes.

Les concepts fondateurs : Objet

- Exemple : La **Renault Mégane bleue immatriculée 123 456 B** du chef de département
- Objet Renault_Megane_du_chef

  Genre: Renault

  Catégorie: Mégane

  Propriétaire: Chef de Département

  Immatriculation: 123 456 B

  Couleur: Bleue

  Avancer ()

  Reculer()

  Démarrer()
- Fin objet.

Etat

Attributs = données

Ce sont des variables stockant des informations sur l'état de l'objet

Méthodes

Caractérisent le comportement de l'objet, c.à.d. l'ensemble des actions (opérations) que l'objet est à même de réaliser

Les concepts fondateurs : Objet

- Exemple : Le compte d'épargne no 12345 de Paul dont le solde est 5200 euros
- Objet compte d'épargne no 12345

  Numéro : 12345

  Type: épargne

  Titulaire: Paul

  Solde: 5200

  Devise : Euros

  Retirer ()

  Deposer()

  Virement()

- Fin objet.

Etat de l'objet compte
Attributs = données

Opérations sur les objets compte

Les concepts fondateurs : Objet

Objet = État (attributs)+ Comportement (méthodes) + Identité

- Tout objet possède une identité qui lui est propre et qui le caractérise.
- L'identité permet de distinguer tout objet de façon non ambiguë, indépendamment de son état.
  - On peut avoir deux objets différents mais ayant le même état.
- Les langages objets utilisent généralement les adresses *(références, pointeurs)* pour réaliser les identifiants

Les concepts fondateurs : Objet

- Un objet peut faire appel aux compétences d'un autre objet.

- L'état et le comportement d'un objet sont liés
  - Le comportement dépend souvent de l'état
  - L'état est souvent modifié par le comportement

Les concepts fondateurs : Classe

- Moyen pour définir des objets ayant une structure commune (liste des attributs) et un comportement commun (liste des opérations).

- C'est un modèle d'objets ayant les mêmes attributs et mêmes comportements.
  - Un objet est une instance d'une classe (une variable ayant comme type la classe).
  - Chaque instance d'une classe possède ses propres valeurs pour chaque attribut.

Les concepts fondateurs : Classe

- Exemple : La classe des Voitures peut être décrit par:

Classe Voiture

Genre : String
Catégorie : String
Propriétaire : String
Immatriculation : String
Couleur : String
Avancer ()
Reculer()
Démarrer()

Fin classe.

Objet Voiture_de_Paul

Genre : Renault
Catégorie : Mégane
Propriétaire : Paul
Immatriculation : 123456 B
Couleur : Bleue
Avancer ()
Reculer()
Démarrer()

Fin classe.

- **Voiture_de_Paul : Voiture**

Les concepts fondateurs : Classe

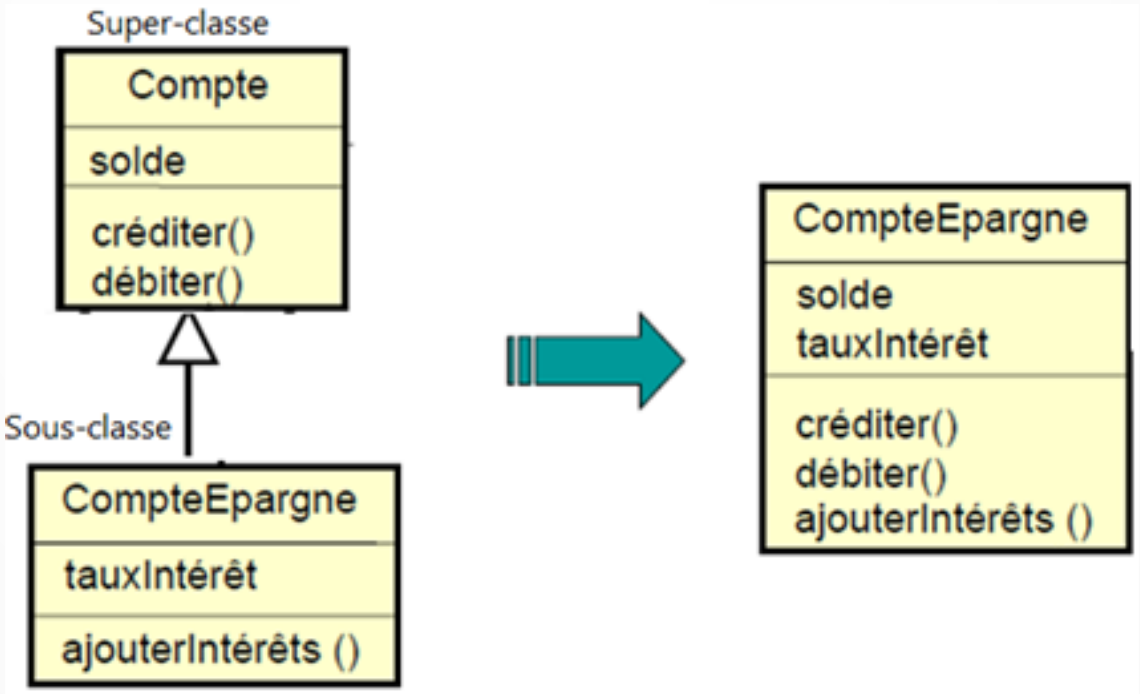
- Les concepts de classe et d'objet sont interdépendants.
- Une classe : Description d'une famille d'objets ayant
  - Même structure
  - Même comportement
- Seuls les objets (ou instance de classe) auront une existence dans les programmes
- **Remarque** : Il y a des classes qui sont abstraites et qu'on ne peut pas les instancier

Les concepts fondateurs : Héritage

- Une classe peut être définie séparément des autres classes, ou bien définie à partir d'une autre classe si elle est un cas particulier de celle-ci.
  - La classe définie est alors appelée sous-classe et l'autre classe est appelée super-classe
  - Entre les deux classes, on a la relation « est une sorte de »
  - La sous classe est une spécialisation de la super-classe
  - La super-classe est une généralisation de la sous-classe
- La sous-classe hérite ou reprend intégralement tout ce qui a déjà été fait dans la super-classe et peut :
  - Ajouter des nouveaux attributs et des méthodes.
  - Redéfinir certaines méthodes.

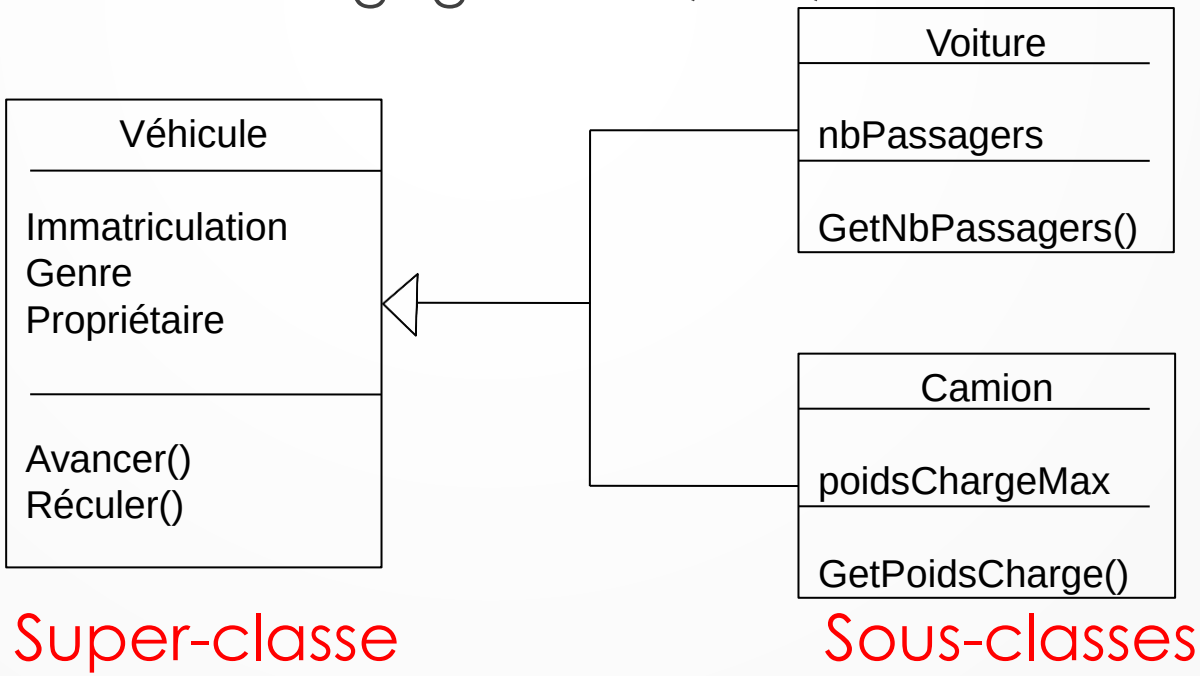
Les concepts fondateurs : Héritage

- **CompteEpargne** « est une sorte de » de **Compte**
- Il hérite de ses propriétés (attributs et méthodes) auxquels s'ajoutent un attribut et une nouvelle méthode
- On se sert de ce qui existe dans la classe Compte pour définir la classe CompteEpargne



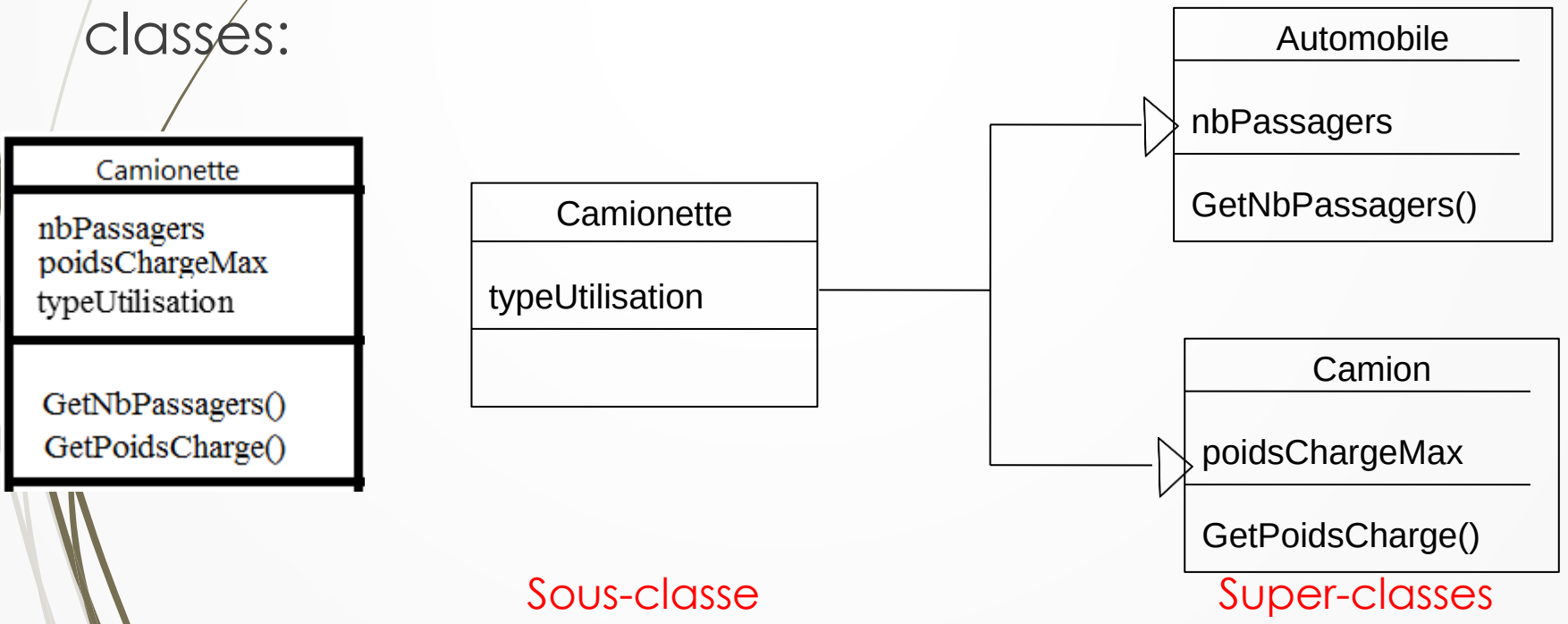
Les concepts fondateurs : Héritage

- Deux types d'héritage :
  - Héritage simple
  - Héritage multiple
- **Héritage simple** : une classe hérite d'une seule classe
  - Utilisée dans les langages Java, C#, …



Les concepts fondateurs : Héritage

- **Héritage multiple:** une classe hérite de plusieurs super-classes
  - Utilisée dans le langage C++, …
- L'héritage multiple est la possibilité de regrouper en une seule classe ce qui est commun à plusieurs classes:



Les concepts fondateurs : Encapsulation

- Concept fondamental : protection de l'information contenue dans un objet
  - Garantit l'intégrité de l'objet (exemple : évite les affectations par erreur, = au lieu de ==!)
  - Garantit la cohérence éventuelle entre plusieurs attributs inter-dépendants
  - Masque en partie le fonctionnement interne d'une classe

Les concepts fondateurs : Encapsulation

Notion de visibilité

- L'accès aux membres de la classe est diffèrent selon que l'on se trouve à l'intérieur ou à l'extérieur de la classe
- A l'intérieur de la classe, tout est permis
- A l'extérieur de la classe, le programmeur ne "voit" que ce que la classe "veut bien lui montrer"
- 3 niveaux de "visibilité" pour les membres (attributs et méthodes) :
  - public : accessibles à tous
  - protected : accessibles seulement aux classes dérivées (voir héritage)
  - private : accessibles seulement au sein de la classe elle-même

Les concepts fondateurs : Polymorphisme

- **Polymorphisme** = une même méthode (même nom), peut avoir plusieurs formes (plusieurs implémentations)

- Polymorphisme : deux types
  - Surcharge,
  - redéfinition

- **Surcharge** = plusieurs implémentations dans la même classe
  - Même nom mais signatures différentes
  - Exemple la méthode **F** a deux signatures différentes
    - **F**(x entier, y réel) : String

      F(x entier) : String

Les concepts fondateurs : Polymorphisme

- **La redéfinition** : Redéfinition d'une opération d'une super-classe dans les sous-classes
  - Une même opération peut se comporter différemment dans plusieurs classes (super-classe et sous-classes)
  - Les différentes méthodes qui implémentent cette même opération dans les différentes sous-classes doivent porter la même signature (nom, paramètres et valeur de retour)