version 1.0

Formations à l'informatique

Découvrez la différence ENI

Vers la programmation orientée objet

Support de cours

Cours POO C#

www.eni-ecole.fr

Logistique

- Horaires de la formation
  - 9h00-12h30
  - 14h-17h30
- Pauses

- **Merci d'éteindre vos téléphones portables**

Module 1

Les paradigmes de programmation

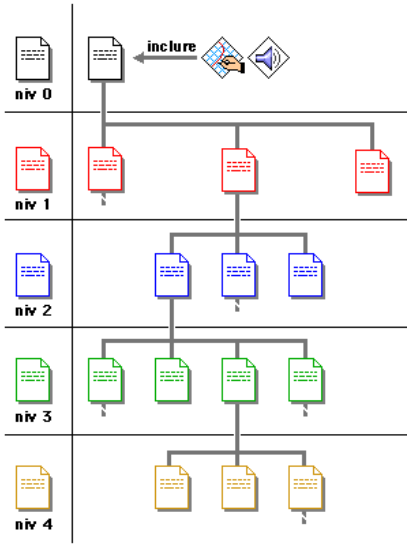
Un paradigme, qu'est ce que c'est ?

« Un paradigme est une vision du monde qui oriente notre manière de penser. »

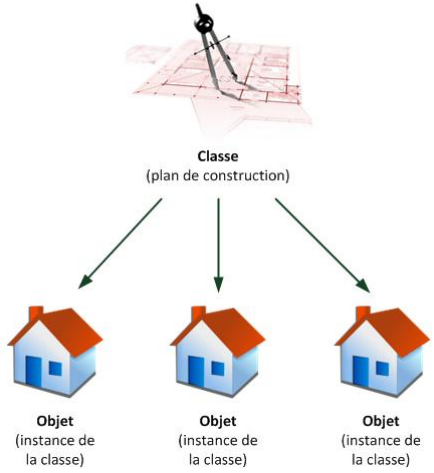
- Un **paradigme de programmation** est un style fondamental de programmation informatique qui traite de la manière dont les solutions aux problèmes doivent être formulées dans un langage de programmation.
- Un paradigme de programmation fournit et détermine la vision qu'a le développeur de l'exécution de son programme.
- Chaque langage de programmation plaide pour un paradigme de programmation particulier.

Deux approches fondamentales

- La programmation procédurale et structurée :
  - Centrée sur les procédures ou opérations.
  - L'application est une suite d'appels de procédures.



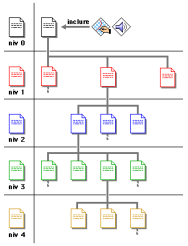
- La programmation orientée objet :
  - Centrée sur les données et les méthodes associées (Objets).
  - L'application est construite à partir des objets. Chaque objet représente une brique de l'application.



La programmation procédurale et structurée

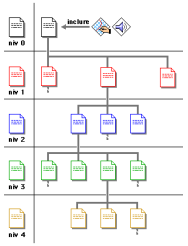
- Centrée sur les procédures (opérations)
  - Décomposition des fonctionnalités d'un programme en procédures qui vont s'exécuter séquentiellement.
- Couplage procédures/données
  - Les données sont indépendantes des procédures.
  - Les données à traiter sont passées en arguments aux procédures.

- Avantages
  - la possibilité de réutiliser le même code à différents emplacements dans le programme sans avoir à le dupliquer (factorisation) :
    - la réduction de la taille du code source,
    - un gain en localisation des modifications, donc une amélioration de la maintenabilité (compréhension plus rapide, réduction du risque de régression) .
  - une façon plus simple de suivre l'exécution du programme .



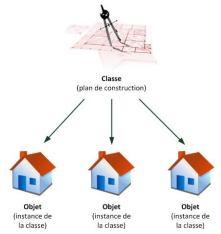
La programmation procédurale et structurée

- Inconvénients
  - Ressemble peu à notre façon de penser
    - Vue algorithmique d'un programme donc très près du langage de la machine.
    - Le programmeur doit faire un effort cognitif pour interpréter ce qu'il veut modéliser pour être "compris" par la machine.
    - Peu intuitif lors de l'analyse d'un code existant.
  - Tend à générer du code "spaghetti"
    - La maintenance et l'ajout de nouvelles fonctionnalités demandent de modifier ou d'insérer des séquences dans ce qui existe déjà.
    - Peu devenir complexe très rapidement.
    - Modularité et abstraction absente (ou presque).
    - Réutilisation ardue => "Couper-coller" = DANGER!
    - Travail d'équipe difficile (peu modulaire), donc la qualité du code en souffre.



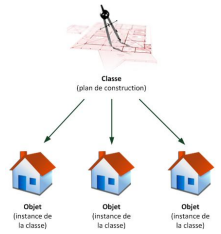
La programmation orientée objet

- Centrée sur les données.
  - Tout tourne autour des "objets" qui sont des petits ensembles de données et de traitements .
    - Un chat a 4 pattes, une queue et fait partie de la famille des félins. Il sait miauler et grimper aux arbres.
    - Une automobile a 4 portes, c'est une familiale, avec un moteur de 6 cylindres, la boite de vitesse est manuelle, etc. Elle connait son autonomie et à quelle vitesse elle roule. Elle sait démarrer, accélérer, freiner et s'arrêter.
- Couplage procédures/données
  - Les données et les traitements qui les manipulent sont encapsulés au sein d'un même composant logiciel.



Où trouver les objets ?

- La modélisation objet consiste à créer un modèle informatique à partir des éléments, des concepts ou des idées issus du monde réel (utilisateur) et propres au métier ou au domaine dont fera partie le système.
- 3 phases :
  - La modélisation Objet consiste à définir, à qualifier dans un premier temps ces éléments sous forme de types, donc indépendamment de la mise en œuvre. C'est ce que l'on appelle l'*analyse orientée objet* ou OOA (Object-Oriented Analysis).
  - Puis, on propose une ou des solutions techniques pour représenter les éléments définis dans le système informatique. C'est ce que l'on appelle la *conception orientée objet* ou OOD (Object-Oriented Design).
  - Il est ensuite possible au développeur de leur donner corps dans un langage de programmation. C'est ce que l'on appelle la *programmation orientée objet* ou OOP (Object-Oriented Programming).
- Pour écrire ces différents modèles, un seul langage : UML (Unified Modeling Language).



Module 2

Théorie de l'objet – Vue d'ensemble

L’objet

- Un objet est une structure de données valuées et cachées qui répond à un ensemble de messages. Cette structure de données définit son *état* tandis que l'ensemble des messages qu'il comprend décrit son *comportement* :
  - Les données qui décrivent sa structure interne sont appelées des *attributs* ;
  - L'ensemble des messages forme ce que l'on appelle *l'interface* de l'objet ; c'est seulement au travers de celle-ci que les objets interagissent entre eux. La réponse à la réception d'un message par un objet est appelée une *méthode*.

  - Le seul mode de communication avec un objet est l’envoi d’un message qui se traduit ensuite par l’exécution d’une méthode.
  - Des informations supplémentaires peuvent être passées dans le message (elles seront par la suite utilisées comme paramètres lors de l’exécution de la méthode).

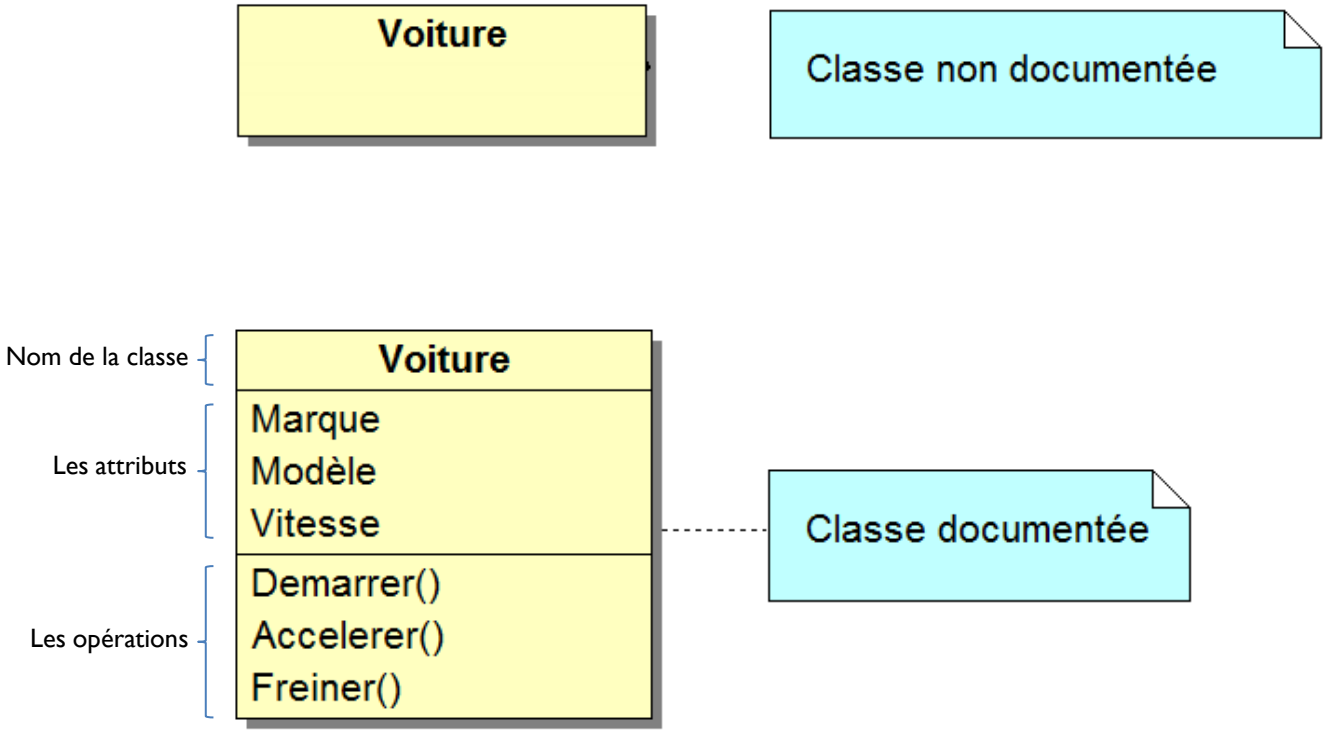
La classe

- Chaque objet est typé.
- Le *type* définit :
  - la syntaxe (« Comment l'appeler ? »).
  - la sémantique (« Qu'est ce qu'il fait ? ») des messages auxquels peut répondre un objet.

- La classe représente le type.
  - Une classe est donc l'abstraction d'une collection d'objets qui possèdent une structure identique (liste des attributs) et un même comportement (liste des opérations).
  - C'est un modèle décrivant le contenu et le comportement des futurs objets de la classe.
  - Un objet est une **instance** d'une et une seule classe.

La classe

- Représentation UML:



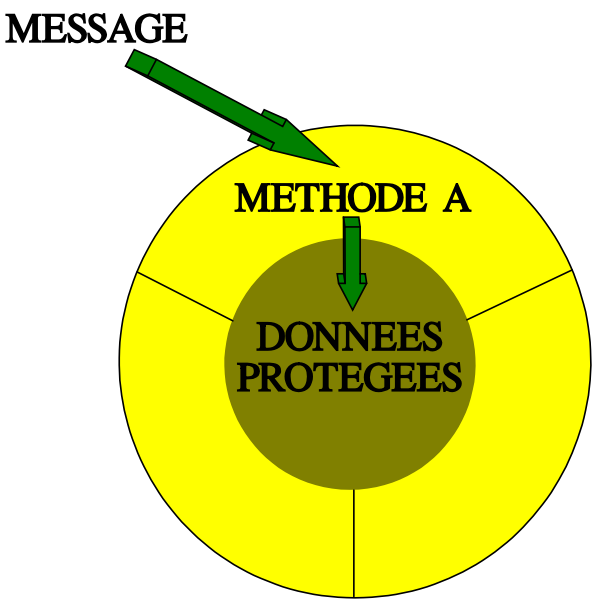
L’encapsulation

- Certains attributs et/ou méthodes peuvent être cachés : c'est le principe d'encapsulation.

- L’encapsulation consiste à masquer les détails d'implémentation d'un objet, en définissant une interface.
- L'interface est la vue externe d'un objet, elle définit les services accessibles (offerts) aux utilisateurs de l'objet.
- L'encapsulation facilite l'évolution d'une application car elle stabilise l'utilisation des objets : on peut modifier l'implémentation d'un objet sans modifier son interface.
- L'encapsulation garantit l'intégrité des données, car elle permet d'interdire l'accès direct aux attributs des objets.

L'encapsulation

- L'encapsulation de données



Exemple :

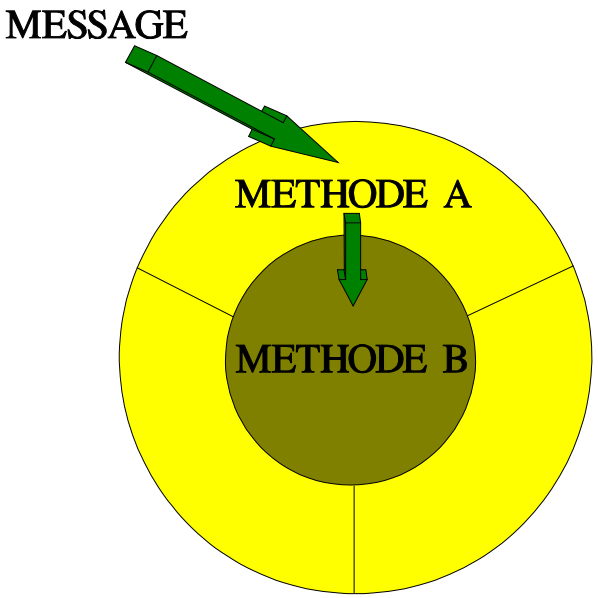
Méthode : Calcul de la clé d'un numéro d'INSEE

Objet : Individu (Assuré Social par exemple)

Donnée protégée : Valeur du modulo pour le calcul de clé

L'encapsulation

- L'encapsulation de méthodes



[Message]
Débiter

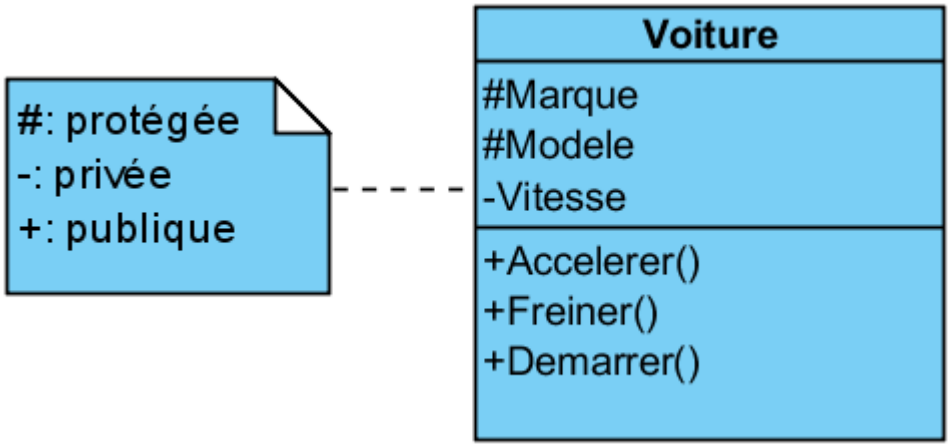
[Objet]
CompteBancaire

[Méthode A - *Visible depuis l'interface*]
Débiter_carte_crédit(somme, code_confidentiel)

[Méthode B - *Non visible par l'utilisateur mais appelée par la méthode A*]
Validation_code_confidentiel(code_confidentiel)

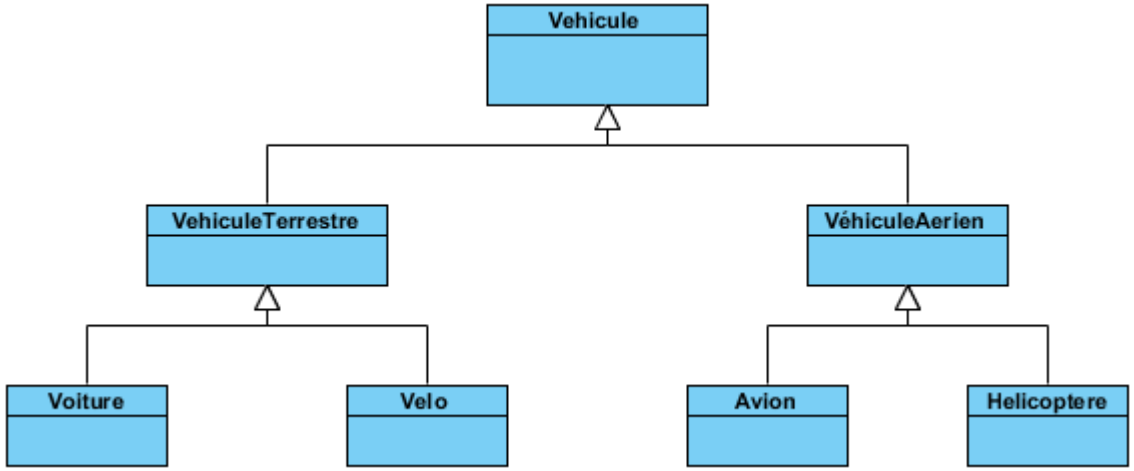
L'encapsulation

- Représentation UML:



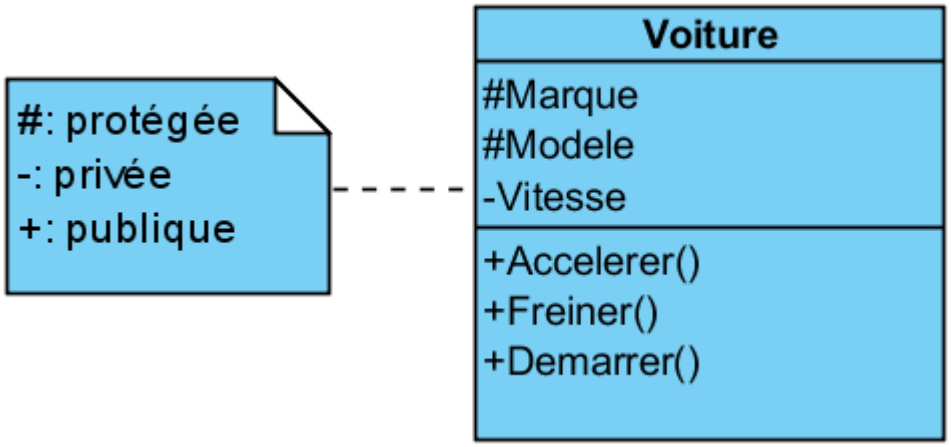
L'héritage

- L'héritage est un mécanisme destiné à exprimer les similitudes entre des classes.
- Il met en œuvre les principes de généralisation et de spécialisation en partageant explicitement des attributs et méthodes communs au moyen d'une hiérarchie de classes.
- La généralisation signifie : « est un ou une sorte de… ».



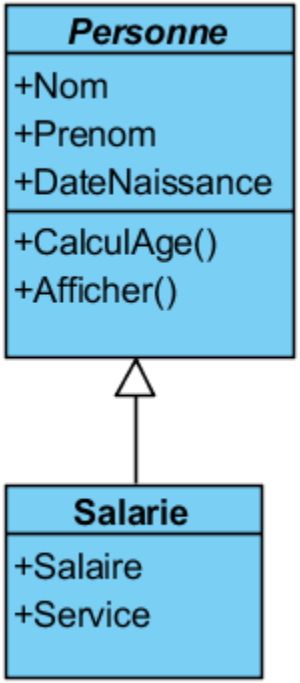
L'héritage

- Les attributs et les opérations sont hérités dans les sous-classes en fonction de la visibilité.
  - Publique (+) : visible par toutes les classes, même les sous-classes.
  - Protégée (#) : visible seulement de la classe mère et de ses classes filles.
  - Privée (-) : invisible pour toutes les classes sauf pour la classe le contenant.



L'héritage

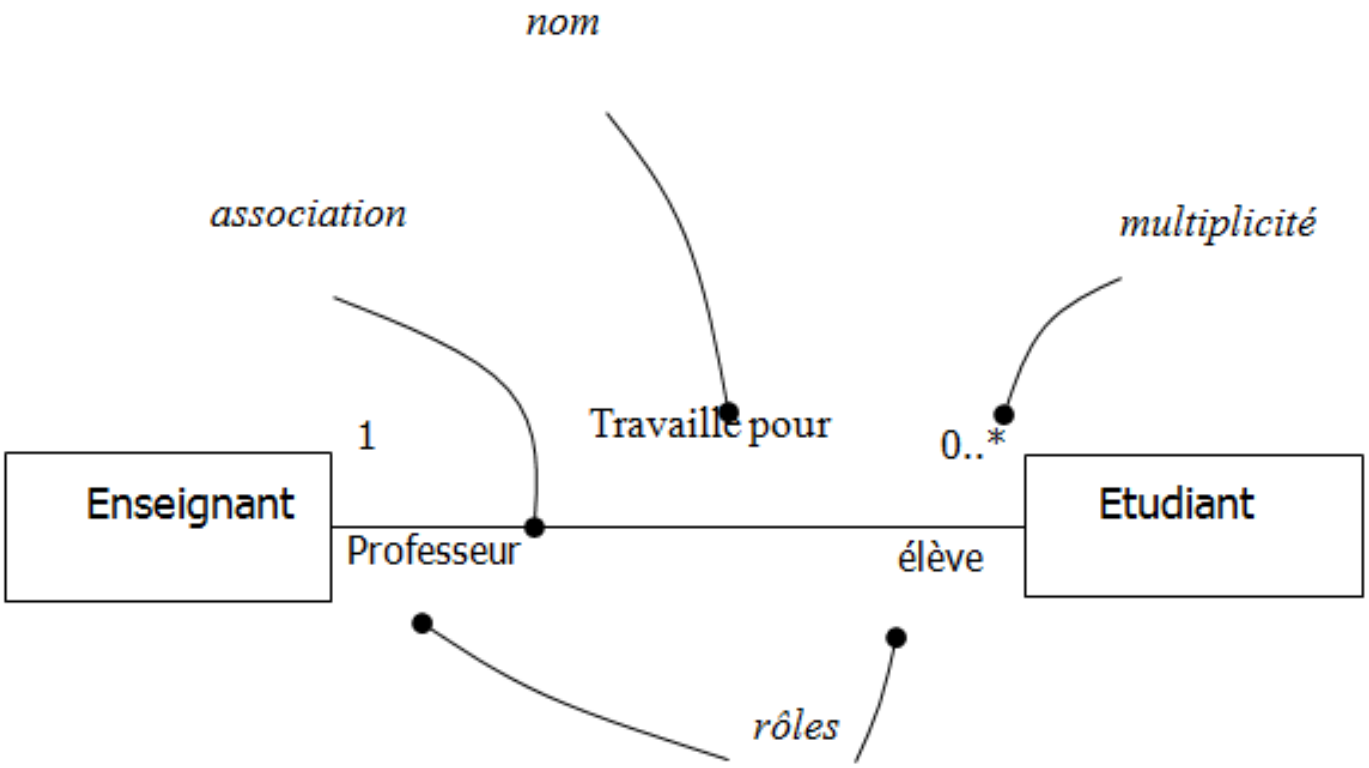
- Les classes abstraites
  - Une classe abstraite ne peut pas être instanciée.
  - Elle sert de base dans une hiérarchie de classe.
  - Elle possède des attributs et des méthodes comme une classe normale



Le polymorphisme

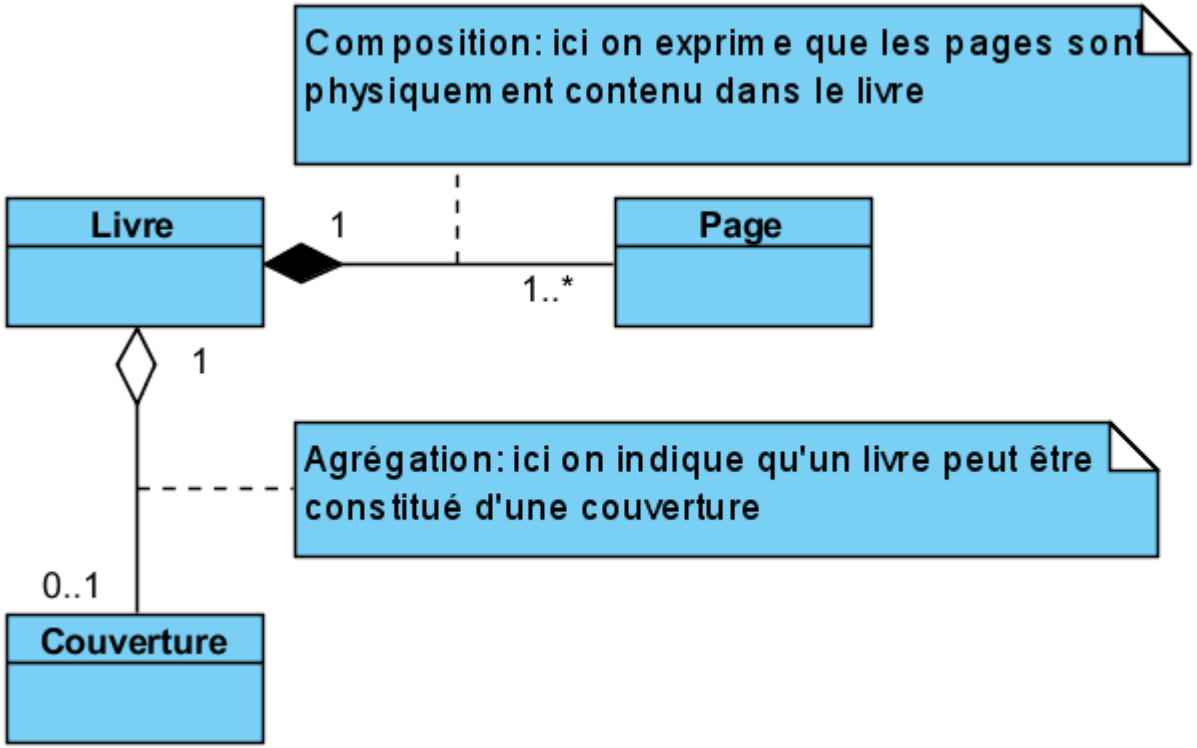
- Il existe 2 types principaux de polymorphisme:
  - Le polymorphisme paramétrique (ou redéfinition)
    - Le polymorphisme paramétrique représente la possibilité de définir plusieurs fonctions de même nom mais possédant des paramètres différents (en nombre et/ou en type) dans la même classe. Le polymorphisme *paramétrique* rend ainsi possible le choix automatique de la méthode à adopter en fonction du type de donnée passé en paramètre.
  - Le polymorphisme d'héritage (ou substitution)
    - La possibilité de redéfinir une méthode dans des classes héritant d'une classe de base s'appelle la **spécialisation**. Il est alors possible d'appeler la méthode d'un objet sans se soucier de son type intrinsèque.

Les associations



Les associations

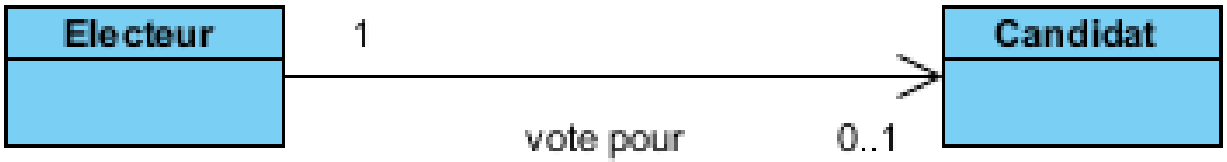
- L'agrégation et la composition



Les associations

- La cardinalité
  - Chaque rôle d'une association porte une indication de multiplicité qui montre combien d'objets de la classe considérée peuvent être liées à un objet de l'autre classe.
    - 1 : un et un seul
    - 0..1 : zéro ou un
    - M..N : de M à N
    - * : de zéro à plusieurs
    - 1..* : de un à plusieurs
    - N : N exactement

- La navigabilité



L'interface

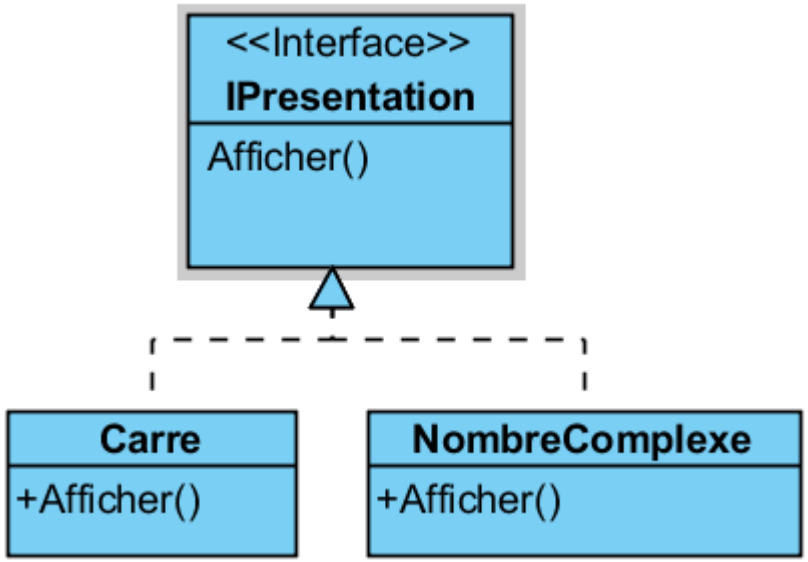
- Des classes qui n'ont rien en commun peuvent vouloir proposer le même service.
  - Exemple : Afficher leur valeur
- Par simplicité d'utilisation, on souhaite appeler ce service toujours de la même manière (même nom de méthode : « Afficher »).

- Ceci est possible avec l'interface :
  - L'interface décrit le service, le formalise.
  - L'interface propose une sorte de contrat que les classes qui l'**implémentent** s'engagent à respecter.
  - Chaque classe se doit de préciser la manière dont elle utilise ce service.

- Une interface est une classe que l'on ne peut pas instancier, qui ne possède pas d'attributs mais seulement des signatures d'opérations.

Concepts – L'interface - 2/2

- Représentation UML:



Le diagramme de classes

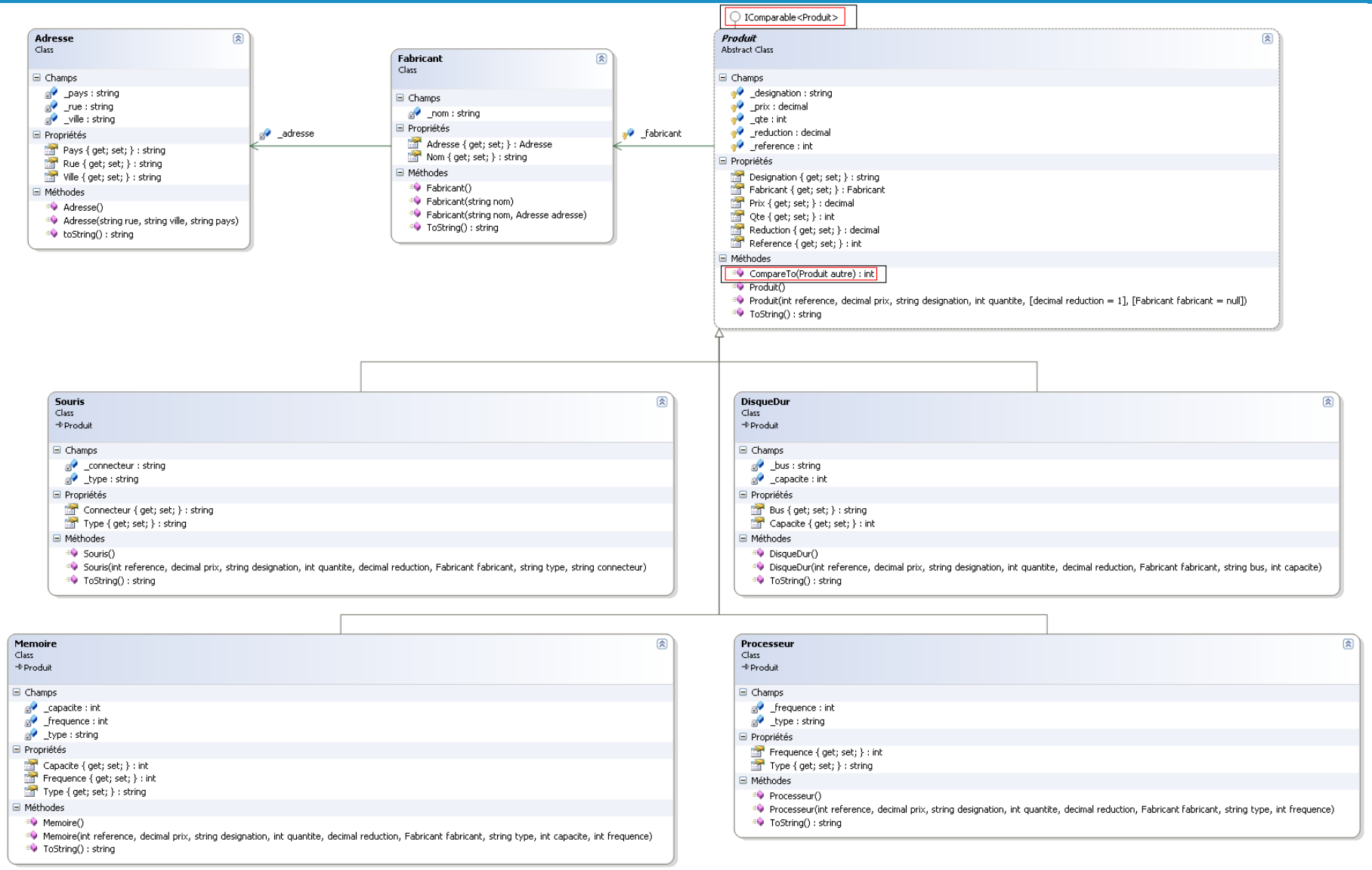
- Un diagramme de classes est une collection d'éléments de modélisation statique qui montre la structure d'un modèle.

- Un diagramme de classes fait abstraction des aspects dynamiques et temporels.

- Pour un modèle complexe, plusieurs diagrammes de classes complémentaires doivent être construits.

- On peut par exemple se focaliser sur :
  - les classes qui participent à un cas d'utilisation ;
  - les classes qui composent un paquetage ;
  - La structure hiérarchique d'un ensemble de classe.

Le diagramme de classes



Fin du module

- Avez-vous des questions ?