Présentations

Responsable du module 	**Mohamed NEMICHE** • **Nom, courte biographie et expérience**
Intervenants 	**Mohamed NEMICHE**
Bibliographie 	• **Livre obligatoire (à prendre à la bibliothèque)** • **Livres recommandés pour consultation** • **Notes du professeur** • **Bibliothèque virtuelle, moodel,…**
Syllabus 	• **Distribution et lecture détaillée du syllabus**

Règles disciplinaires

Horaires 	• **Les retards ne sont pas tolérés** • **La porte de la salle se ferme après 5 min du démarrage du cours**
Absence 	• **3 abs. de 2h / Matière → Avertissement + 0 en assiduité** • **6 abs. de 2h / Matière → Interdiction d'examen** • **9 abs. de 2h / Matière → Interdiction de rattrapage**
Pause 	• **La durée de pause de ne dois pas dépasser 15 minutes** • **Les étudiants ne peuvent pas sortir pendant le cours**
Téléphone 	• **Les téléphones doivent être en arrêt pendant le cours** • **Le professeur a le droit de confisquer tout appariel qui gène le déroulement du cours**
Autres 	• **Tenue correcte obligatoire en cours** • **Interdit de manger ou de boire pendant le cours** • **Tout manquement de respect envers un intervenant, un membre la direction ou un étudiant est sanctionné**

INTRODUCTION AU GENIE LOGICIEL ORIENTE OBJET

Mohamed Nemiche
nemiche@uv.es

Plan de la Séance

- **Principes et problématique du génie logiciel**
- **Décomposition et modularité**
- **Cohésion et couplage**
- **Pourquoi l'approche orienté objet ?**
- **Concepts clés de l'orientation à objets**
  - Abstraction
  - Objets & Classes
  - Encapsulation,
  - Héritage,
  - Surcharge, Redéfinition,
  - Polymorphisme.

Principes et problématique du génie logiciel

Qu'est-ce qu'un logiciel ?

Définition (Logiciel)

Un logiciel est un ensemble d'entités nécessaires au fonctionnement d'un processus de traitement automatique de l'information.

Parmi ces entités, on trouve par exemple :

- des programmes (en format *code source ou exécutables) ;*
- des documentations d'utilisation ;
- des informations de configuration.

Qu'est-ce qu'un logiciel ?

- **Caractéristiques d'un produit logiciel**
  - Un logiciel est un produit avec des caractéristiques particulières
    - Le processus de développement se poursuivi après la livraison du logiciel, pour la maintenance
      - Après son développement un produit logiciel est mis en exploitation et entre dans une phase de maintenance qui peut remettre en cause les fonctions du système et impliquer des modifications et/ou un re-développement
        - La correction des défauts : maintenance corrective
        - Adaptation du logiciel à un nouvel environnement : maintenance adaptative
        - Amélioration des caractéristiques et suivi de l'évolution des besoins : maintenance perfective

La crise du logiciel

- **Le problème est que à un moment donné de l'histoire de l'informatique (développement des logiciels) les développeurs n'ont pas sais prendre compte de tous des particularité et ils n'ont pas réfléchit sérieusement au processus de développement.**
- **Cet état de fait a mené à ce qu'on appelle la crise du logiciel pendant la fin des années soixante.**

La crise du logiciel

- **La crise du logiciel**
  - Quelques constats de l'ampleur de l'impact des défaillances :
    - la sonde Mariner vers Vénus s'est perdue dans l'espace à cause d'une erreur de programme FORTRAN (années 60) ;
    - en 1972, lors d'une expérience météorologique en France 72 ballons contenant des instruments de mesure furent détruits à cause d'un défaut dans le logiciel ;
    - en 1981, le premier lancement de la navette spatiale a été retardé de deux jours à cause d'un problème logiciel. La navette a d'ailleurs été lancé sans que l'on ait localisé exactement le problème (mais les symptômes étaient bien délimités) ;
    - le développement du compilateur PL1 de Control Data n'a jamais abouti ;

La crise du logiciel

- **La crise du logiciel**
  - Quelques constats de l'ampleur de l'impact des défaillances :
    - la SNCF a rencontré des difficultés importantes pour la mise en service du système Socrate
    - l'explosion d'Ariane 5, le 4 juin 1996, qui a coûté un demi milliard de dollars (non assuré !), est due à une faute logicielle d'une composante dont le fonctionnement n'était pas indispensable durant le vol
  - Pourquoi : manque de méthodologie de développement des produits logiciels

La crise du logiciel

En 1979, le gouvernement américain estimait que la plupart des grands projets avaient échoué :

• Payés mais jamais livrés	$3.2M	47%
• Livrés mais jamais utilisés	$2.0M	29%
• Abandonnés ou refaits	$1.3M	19%
• Utilisés après modification	$0.2M	3%
• Utilisés tel quel	$0.1M	2%

La crise du logiciel

Selon « The Standish Group Report», en 1999

Project Duration, Team Size Affect Project Success

Project Size	People	Time (mos.)	Success Rate
Less than $750K	6	6	55%
$750K to $1.5M	12	9	33%
$1.5M to $3M	25	12	25%
$3M to $6M	40	18	15%
$6M to $10M	+250	+24	8%
Over $10M	+500	+36	0%

La crise du logiciel

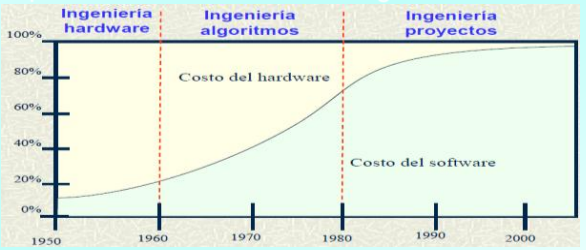
- **Chaos Repport by Standish Group 2009**
  - Seulement 32% des projets réussissent (temps, budget et *features)*
  - 44% ne respectent pas les délais, les coûts et/ou les besoins énoncés
  - 24% des projets n'aboutissent pas

La crise du logiciel

- **La crise du logiciel**
  - Prise de conscience :
    - La première reconnaissance publique de la crise du logiciel a été faite lors d'une conférence à Garmisch
    - Les symptômes les plus caractéristiques de cette crise sont :
      - Les logiciels réalisés ne correspondent souvent pas aux besoins des utilisateurs
      - Les logiciels contiennent trop d'erreurs (qualité du logiciel insuffisante)
      - Les coûts de développement sont rarement prévisibles et sont généralement exagérés
      - La maintenance des logiciels est une tâche complexe et coûteuse
      - Les délais de réalisation sont généralement dépassés
      - Les logiciels sont rarement portables

La crise du logiciel

- **La crise du logiciel**
  - Prise de conscience :
    - Comparer l'évolution du coût du logiciel et du matériel :



      - Le coût du matériel baisse régulièrement;
      - Le matériel devient de plus en plus puissant, conduisant à la réalisation de logiciels de plus en plus complexes et donc coûteux
      - Le coût des logiciels représenterait actuellement 85 % des dépenses totales des systèmes informatiques

La crise du logiciel

- **La crise du logiciel**
  - Quelques sources de la crise
    - Une idée grossière du logiciel à réaliser est suffisante pour commencer d'écrire un programme (il est assez tôt de se préoccuper des détails plus tard).
      - Faux : une idée imprécise du logiciel à réaliser est la cause principale d'échecs
    - Une fois que le programme est écrit et fonctionne, le travail est terminé
      - Faux : la maintenance du logiciel représente un travail important : le coût de la maintenance représente plus du 50 % du coût total d'un logiciel

La crise du logiciel

- **La crise du logiciel**
  - Quelques sources de la crise
    - Si les spécifications du logiciel à réaliser changent continuellement, cela ne pose pas de problème, puisque le logiciel est un produit souple
      - Faux : des changements de spécifications peuvent se révéler coûteux et prolonger les délais
    - Si la réalisation du logiciel prend du retard par rapport aux délais prévus, il suffit d'ajouter plus de programmeurs afin de finir dans les délais.
      - Faux : si l'on ajoute de gens à un projet, il faut compter une période de familiarisation. Le temps passé à communiquer à l'intérieur du groupe augmente également, lorsque la taille du groupe augmente

Génie logiciel

- **De tout cet état de lieu, ces idées fausses, ces échecs qui ont couté d'un temps et de l'argent, et les constats concernant les symptômes de la maladie du logiciel est né une nouvelle discipline : le génie logiciel.**
- **Le génie logiciel est donc l'art :**
  - De spécifier, de concevoir, de réaliser, et de faire évoluer des programmes, des documentations et des procédures de qualité avec un coût et dans des délais raisonnables.

Quel est l'enjeu du génie logiciel?

- **Essayer de maitriser la complexité et le coût d'un développement de logiciel.**
- **Augmenter la probabilité de réussite d'un projet de développement de logiciel.**
- **Bien développer le bon logiciel.**

- **Difficile équilibre : Qualité – Coût – Délai**
  - Rapide et pas cher ! Mauvaise qualité
  - Rapide et de bonne qualité ! Cher
  - Bonne qualité et pas cher ! Lent

Principes appliqués en génie logiciel

1. **Rigueur et formalisation**
   - précision des spécifications
   - utilisation de formalismes appropriés à chaque étape du développement d'un logiciel
2. **Décomposition d'un problème en sous-problèmes**
   - Il s'agit de :
     - *Décorréler les problèmes pour n'en traiter qu'un seul à la fois.*
     - *Simplifier les problèmes (temporairement) pour aborder leur* complexité progressivement.

Principes appliqués en génie logiciel

3. **Modularité**
   - Décomposition du logiciel en éléments physiquement distincts (modules)
4. **Abstraction**
   - Indépendance des spécifications par rapport aux choix de représentations internes du logiciel (implémentation et implantation)
5. **Prévision de changements**
   - à tous niveaux, imposés ou délibérés

Principes appliqués en génie logiciel

6. **Généricité**
   - Un logiciel réutilisable a beaucoup plus de valeur qu'un composant dédié.
   - Un composant est générique lorsqu'il est adaptable
7. **la construction incrémentale**
   - Un développement logiciel a plus de chances d'aboutir si il suit une cheminement incrémental (*baby-steps).*
   - Réalisation progressive du logiciel, avec développement et mise au point séquentielle ou parallèle des composants

Modularité: Couplage et cohésion

Couplage

- **Le couplage est une mesure du degré auquel un élément est lié à un autre. S'il y a couplage ou dépendance, l'objet dépendant peut être affecté par les modifications de celui dont il dépend.**
- **Le faible couplage a pour objectif de faciliter la maintenance en minimisant les dépendances entre éléments.**
- **Bien entendu, il ne faut pas tomber dans le piège de décider de concevoir des éléments tous indépendants et faiblement couplés, car ceci irai à l'encontre du principe OO définissant un système objet comme un ensemble d'objets connectés les uns aux autres et communiquant entre eux**

Cohésion

- **La cohésion mesure le degré de spécialisation des responsabilités d'un module.**
- **La cohésion faible altère la compréhension, la réutilisation, la maintenabilité et subit toute sorte de changements.**
  - Par exemple on ne peut exiger d'un réfrigérateur qu'il fasse radiateur et range-disques en même temps. La multiplication des disciplines à responsabilité accroît exponentiellement le risque d'erreurs intrinsèques à cette classe.
- **Mettre tous ce qui semblable dans le même module/classe.**

Modularité: forte cohésion & faible couplage

- **Modularité :**
  - forte cohésion dans la classe, faible couplage entre les classes

Pourquoi l'approche orienté objet ?

Approche fonctionnelle vs Approche orientée objet

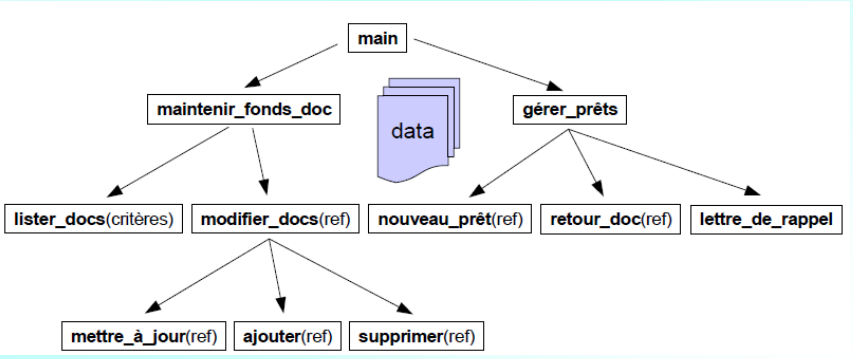
- **Approche fonctionnelle : la modélisation est réalisée à partir de fonctions que doit réaliser le système.**
- **Approche orientée objet : on identifie les objets manipulés par le système, avec leurs états et leurs comportements.**

Approche fonctionnelle

- **C'est l'approche traditionnelle de la programmation : la modélisation du logiciel est réalisée a partir des fonctions que doit implémenter (réaliser) le système.**
  - ➔ cf. : module de langage C

Approche fonctionnelle

- **La découpe fonctionnelle d'un problème informatique est une approche intuitive.**
  - Exemple : un logiciel de gestion d'une bibliothèque

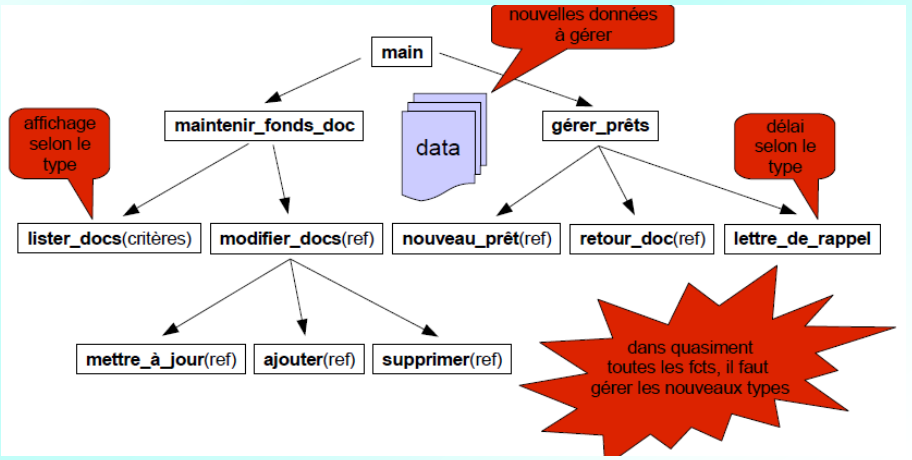


Approche fonctionnelle

- Maintenance complexe en cas d'évolution En cas d'évolution majeure du logiciel (passage de la gestion d'une bibliothèque a celle d'une médiathèque, par exemple), de gros problèmes se posent.
- Même si la structure générale du logiciel reste valide, la multiplication des points de maintenance, engendrée par le chainage des fonctions, rend l'adaptation très laborieuse. Le logiciel doit être retoucher dans sa globalité

Approche fonctionnelle

- Evolution bibliothèque → médiathèque



Approche fonctionnelle

- **Evolution bibliothèque → médiathèque**
  - L'évolution nécessite (entre autres) :
    - de faire évoluer les structures de données qui sont manipulées par les fonctions
    - d'adapter les traitements qui ne manipulaient a l'origine qu'un seul type de document (des livres)
  - Il faudra donc modifier toutes les portions de code qui utilisent la base documentaire pour gérer les données et les actions propres aux différents types de documents.
  - En fait, c'est la quasi-totalité de l'application qui devra être adaptée pour gérer les nouvelles données et réaliser les traitements correspondants. Et cela, a chaque fois qu'on décidera de gérer un nouveau type de document.

Approche fonctionnelle

- **Une amélioration possible : rassembler les traitements associes a un type, auprès du type.**
  - Avantage : cela permet de retrouver immédiatement ou faire la modification et ne la faire qu'a cet endroit précis.
  - Ecrit en ces termes, le logiciel est plus facile a maintenir et bien plus lisible.

Approche fonctionnelle

- Les modifications apportées au logiciel de gestion de médiathèque nous ont amène a transformer ce qui était a l'origine une structure de données, manipulée par des fonctions, en une entité autonome, qui regroupe un ensemble de propriétés cohérentes et leur traitements associes.
- Une telle entité s'appelle un objet et constitue le concept fondateur de l'approche du même nom.

Approche orientée objet

Définition

- La programmation objet tourne autour d'une unique entité : l'objet ceci est a mettre en opposition de la programmation procédurale par exemple qui est constituée de procédures et fonctions agissant sur des données dissociées.
- Comme beaucoup de nouveautés dans le monde du développement, la programmation objet avait pour but de simplifier la programmation en diminuant le niveau de complexité sur les gros programmes.

Pourquoi une structuration orientée objet ?

- **Unicité et universalité du paradigme**
  - Réduire le décalage entre monde réel et logiciel ;

    Objets réels ➔ Objets conceptuels ➔ Objets logiciels
- **Réutilisabilité et évolutivité facilitées par différents mécanismes**
  - Encapsulation, Modularité, Abstraction, Polymorphisme, Héritage
  - Faible couplage inter-objets / Forte cohésion intra-objet

Pourquoi une structuration orientée objet ?

- **Paradigme qui arrive à maturité**
  - Bibliothèques de classes, Design patterns, UML, Méthodologies de développement (USDP, Agile, XP, ...),
  - Environnements de développement intégrés (IDE)

Concepts clés de l'orientation à objets

Concepts clés de l'orientation à objets

- Abstraction
- Objets & Classes
- Encapsulation,
- Héritage,
- Surcharge, Redéfinition,
- Polymorphisme

Abstraction

- Il s'agit d'un processus qui consiste à identifier les caractéristiques intéressantes d'une entité, en vue d'une utilisation précise.
- L'abstraction désigne aussi le résultat de ce processus, c'est-à-dire l'ensemble des caractéristiques essentielles d'une entité, retenues par un observateur.

Qu’est-ce qu’un objet ?

- Dans la vie courante on manipule des objets.
- Un objet possède un état et réagit selon un comportement.
- L’état évolue au cours du temps, en fonction du comportement
- Les objets échanges des messages.

Qu'est-ce qu'un objet ?

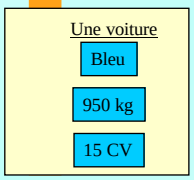
- **Objet informatique :**
  - Est une unité atomique formée de l'union d'un état et d'un comportement
  - Définit une représentation abstraite d'une entité du monde réel ou virtuel, dans le but de la piloter ou de la simuler
    - Voiture, étudiant, compte en banque, …
- **Les objets du monde réel et du monde informatique naissent, vivent et meurent**

Objet

- **Caractéristique fondamentales d'un objet (informatique)**

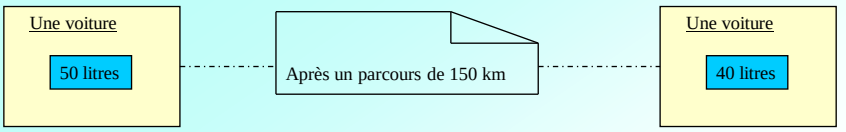
  Objet = État + Comportement + Identité

  - État
    - Regroupe les valeurs instantanées de tous les attributs d'un objet :
      - attribut est une information qui qualifie l'objet qui le contient
      - Chaque attribut peut prendre une valeur dans un domaine de définition donné
    - Exemple : Un objet voiture regroupe les valeurs des attributs couleur, masse et puissance fiscale



Objet

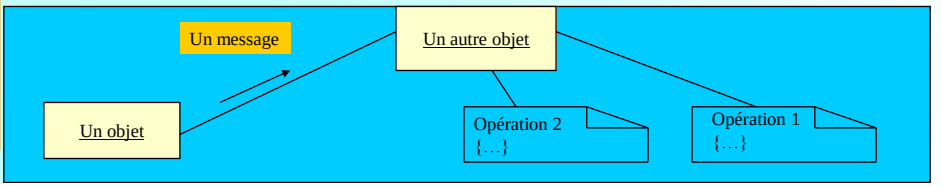
- **Caractéristique fondamentales d'un objet (informatique)**
  - État
    - L'état d'un objet à un instant donné, correspond à une sélection de valeurs, parmi toutes les valeurs possibles des différent attributs
    - L'état évolue au cours du temps, il est la conséquence de ses comportement passés
      - Une voiture roule, la quantité de carburant diminue, les pneus s'usent, etc.



    - Certaines composantes de l'état peuvent être constantes
      - La marque de la voiture, pays de la construction de la voiture

Objet

- **Caractéristique fondamentales d'un objet (informatique)**
  - Le comportement
    - Regroupe toutes les compétences d'un objet et décrit les actions et les réactions de cet objet
    - Chaque atome (partie) de comportement est appelé opération
      - Les opérations d'un objet sont déclenchées suite à une stimulation externe, représentée sous la forme d'un message envoyé par un autre objet
      - L'état et le comportement sont liés



Objet

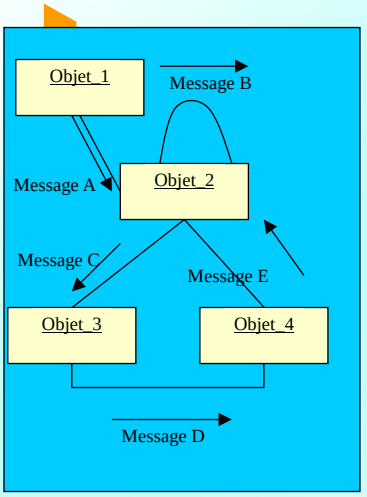
- **Caractéristique fondamentales d'un objet**
  - L'identité
    - Chaque objet possède une identité qui caractérise son existence propre
    - L'identité permet de distinguer tout objet de façon non ambiguë, indépendamment de son état
    - Permet de distinguer deux objets dont toutes les valeurs d'attributs sont identiques
      - deux pommes de la même couleur, du même poids et de la même taille sont deux objets distincts.
      - Deux véhicules de la même marque, de la même série et ayant exactement les mêmes options sont aussi deux objets distincts.

Objet

- **Caractéristique fondamentales d'un objet**
  - L'identité
    - L'identité est un concept, elle ne se représente pas de manière spécifique en modélisation. : chaque objet possède une identité de manière implicite
    - En phase de réalisation, l'identité est souvent construite à partir d'un identifiant issu naturellement du domaine du problème.
      - Nos voitures possèdent toutes un numéro d'immatriculation, nos téléphones un numéro d'appel.

Objet

- Communication entre objets : le concept de message
  - Les systèmes informatiques à objets peuvent être vus comme des sociétés d'objets qui travaillent en synergie afin de réaliser les fonctions de l'application
  - Le comportement global d'une application repose sur la communication entre les objets qui la composent
  - L'unité de communication entre objets s'appelle message



Classe

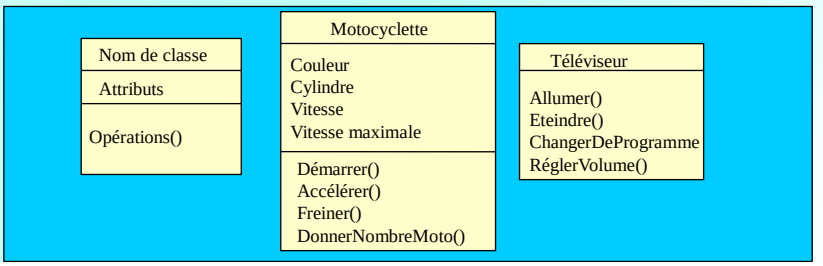
- **Définition**
  - Une classe décrit une abstraction d'objets ayant
    - Des propriétés similaires
    - Un comportement commun
    - Des relations identiques avec les autres objets
    - Une sémantique commune

Classe

- **Caractéristiques d'une classe**
  - Les généralités sont contenues dans la classe et les particularité sont contenues dans les objets
  - Les objets sont construits à partir de la classe, par un processus appelé instanciation : tout objet est une instance de classe
  - Un objet créé par (on dit également appartenant à) une classe sera appelé une *instance de cette classe* ce qui justifie le terme ``*variables d'instances* ''
  - Nous distinguons deux types de classes
    - Classe concrète : peut être instanciée
    - Classe abstraite : est une classe qui ne donne pas directement des objets.

Classe

- **Représentation graphique d'une classe en UML**
  - Chaque classe est représentée sous la forme d'un rectangle divisé en trois compartiments
  - Les compartiments peuvent être supprimés pour alléger les diagrammes
  - Représentation des classes abstraites : le nom d'une classe abstraite est en italique



Encapsulation

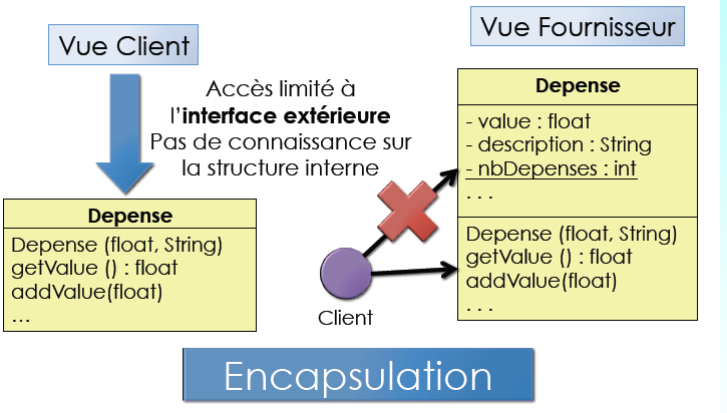
Encapsulation

- **L'encapsulation est un mécanisme consistant à rassembler les données et les méthodes au sein d'une structure en cachant l'implémentation de l'objet, c'est-à-dire en empêchant l'accès aux données par un autre moyen que les services proposés**

- **L'encapsulation permet donc de masquer un certain nombre de champs et méthodes tout en laissant visibles d'autres champs et méthodes.**
  - *Pensez à votre voiture : elle vous fourni des méthodes (tourner le volant, appuyer sur le frein. etc.) afin de pouvoir interagir avec elle. Vous savez donc qu'en tournant le volant, que la voiture changera de direction. Vous n'avez toutefois pas à savoir comment la mécanique interne de ce changement de direction s'opère. Tout ce qui important de savoir c'est que de tourner le volant change la direction.*
  - *L'encapsulation permet de garder une cohérence dans la gestion de l'objet, tout en assurant l'intégrité des données qui ne pourront être accédées qu'au travers des méthodes visibles.*

Encapsulation

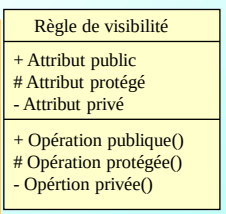
- **Points de vue** : fournisseur X client

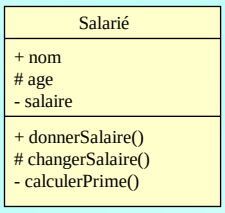


Encapsulation

- **Encapsulation**
  - **Protéger les objets** d'interventions extérieures intempestives
  - Chaque objet est **responsable de son état**
- **Avantages**
  - Permettre au **fournisseur** de **modifier l'implémentation** interne sans affecter les clients
  - Pouvoir garantir le **respect aux règles** (« métier »)
  - Avoir un debug / **maintenance + facile**

Encapsulation

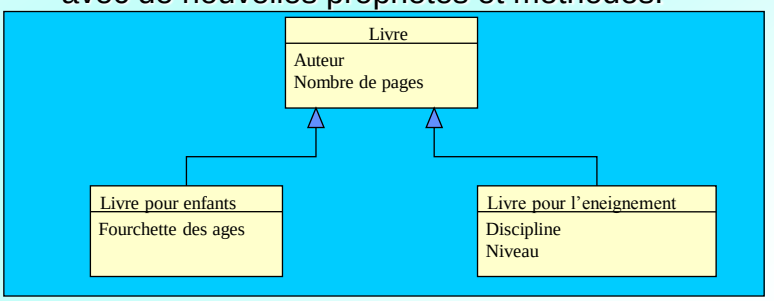




- Il est possible d'assouplir le degré d'encapsulation au profit de certaines classes utilisatrices bien particulières
  - En définissant des niveaux de visibilité
- Les trois niveaux distincts d'encapsulation couramment retenus sont:
  - Niveau privé : c'est le niveau le plus fort; la partie privée de la classe est totalement opaque
  - Niveau protégé : c'est le niveau intermédiaire ; les attributs placés dans la partie protégée sont visibles par les classes dérivées de la classe fournisseur. Pour toutes les autres classes, les attributs restent invisibles
  - Niveau publique : ceci revient à se passer de la notion d'encapsulation et de rendre visibles les attributs pour toutes les classes

Héritage

- **Les objets sont définis à partir de classes. En POO, les classes sont définis à partir d'autres classes.**
  - Par exemple : un VTT est une sous-classe de Vélo.
  - Une sous-classe n'est pas limitée aux caractéristiques de sa super-classe. Elle peut étendre cette dernière avec de nouvelles propriétés et méthodes.



Héritage

Définition

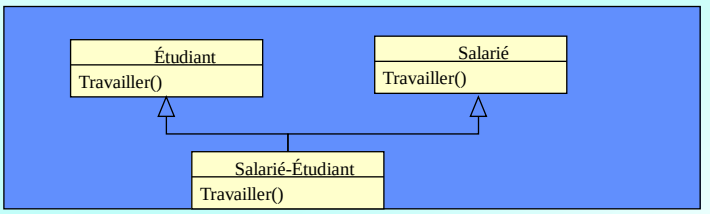
une classe qui hérite (ou dérive) d'une autre a les mêmes attributs et les méthodes qu'elle, et peut en définir des supplémentaires. Ses instances peuvent être utilisées comme des instances de la sous-classe ou de la sur-classe.

L'héritage traduit le principe de généralisation/spécialisation. Une classe dérivée (fille) est une forme spécialisée de sa classe de base :

- Le modèle ne doit pas être alourdi par des classes trop nombreuses
- Certains langages proposent l'héritage sélectif : le programmeur doit spécifier les attributs et méthodes à hériter
- Héritage multiple (pas Java) : une classe concrète hérite de deux classes abstraites (certains langages préfixent les noms de méthodes et propriétés par leur classe d'origine)

Héritage multiple

- **Les hiérarchies de classes**
  - Généralisation multiple : elle existe entre arbres de classes disjoints
    - Une classe ne peut posséder qu'une fois une propriété donné



  - Problème de classification
    - L'établissement d'une hiérarchie (classification) dépend du point de vue
      - Pas une seule hiérarchie de classe mais des hiérarchies, chacune adaptée à un usage donné
      - Exemple :les animaux
        - Critères de classification : type de nourriture, la protection

Redéfinition et Surcharge

Redéfinition

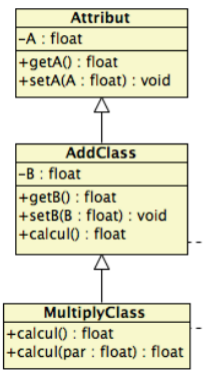
- **Redéfinition** d'une méthode précédemment **définie**
- **Nouvelle méthode** avec la **même signature**
  - **Signature = Implémentation ≠**
- **Redéfinition** d'un **comportement hérité**
  - Le comportement de la **super-classe** est **masqué**, mais il reste **disponible**
    - **super**

Surcharge

- Définition de **plusieurs méthodes** aux **noms identiques**, mais aux **signatures distinctes**
- Différentes **implémentations** pour une **même opération**
- **Signatures ≠**
  - **Paramètres ≠**

Redéfinition et Surcharge

- **Exemple : Surcharge & Redéfinition**



- **Redéfinition** : méthode **calcul()**
  - Changement de comportement

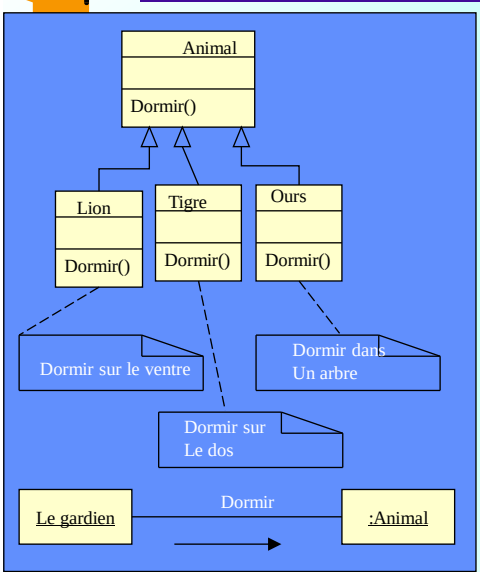
```
public float calcul() { return this.getA() + this.getB(); }
public float calcul() { return this.getA() * this.getB(); }
```

- **Surcharge** : méthode **calcul(par)**
  - Nouvelle méthode

```
public float calcul(float par) {
    return par * super.calcul() ; }
```

super
référence à la super-classe
comportement de la super-classe
reste disponible

Polymorphisme



- **Les hiérarchies de classes**
  - Polymorphisme :
    - Le polymorphisme signifie qu'un même opération peut se comporter différemment sur différents classes
      - L'opération Dormir() agira de manières différentes sur un Lion, un Tigre ou un Ours
    - Le polymorphisme signifie que les différentes méthodes d'une opération ont la même signature
      - Lorsque une opération est invoquée sur un objet, celui-ci « connaît » sa classe et par conséquent est capable d'invoquer automatiquement la méthode correspondante.

Polymorphisme

- **Le polymorphisme signifie qu'une même opération peut se traduire différemment selon l'objet sur laquelle elle s'applique : C'est *la capacité d'un objet à prendre plusieurs formes.***
- **Le terme polymorphisme décrit la caractéristique d'un élément qui peut prendre plusieurs formes, comme l'eau qui se trouve à l'état solide, liquide ou gazeux**
- **En informatique : le polymorphisme désigne le fait qu'un nom d'objet peut désigner des instances de classes différentes issues d'une même arborescence**