

Introduction aux concepts objet



Objectifs

- Positionner les langages objet vis-à-vis des approches procédurales
- Avoir une bonne approche des concepts objet
- Comprendre les conséquences et les atouts d'un développement objet sur un projet
- Aborder la notion de modélisation objet avec UML



Sommaire

1. Evolution des langages vers l'objet

2. Caractéristiques de l'approche objet

Annexe: Introduction à UML



Sommaire

1. Evolution des langages vers l'objet

1. Recule sur un projet Informatique
2. Apparition, historique, évolution et situation actuelle
3. Comparaison avec les approches traditionnelles linéaire, procédurales, modulaire

2. Caractéristiques de l'approche objet

Annexe: Introduction à UML

Informatique = Simuler

- Toute activité informatique tend à modéliser des caractéristiques du monde réel pour les simuler
 - simulation nucléaire
 - automatisation de calculs fastidieux
 - utilisation de fiches à des fins statistiques
 - gestion de la comptabilité
 - applications militaires
 - publication assistée par ordinateur
 - ...

Informatique = Programmer

Qu'est-ce qu'un programme informatique ?

- Suite d'instructions codées dans une machine
- Les instructions manipulent des données
 - les données changent d'état
- Très grand nombre de langages de programmation
 - apparus à diverses époques
 - Fortran, Lisp, Cobol, Simula, C, Basic, Pascal, Smalltalk, PROLOG, ADA, C++, Java...
 - Chacun avec ses propres caractéristiques, en réponse à divers besoins
 - unifier la programmation des systèmes de gestion US → COBOL
 - écrire des programmes d'intelligence artificielle → LISP
 - faciliter l'écriture d'UNIX → C
 - unifier la programmation des systèmes militaires « temps réel » US → ADA

Informatique = Industrialiser

- Résoudre la crise du logiciel...
- Objectif 1 : améliorer la productivité des développeurs
- Objectif 2 : maîtriser la complexité des développements de logiciels de + en + gros
- Objectif 3 : améliorer la maintenance, l'évolution, bref la pérennité des logiciels
- Comment ? → Critères importants
 - Lisibilité
 - Réutilisabilité
 - Modularité

Réutilisabilité

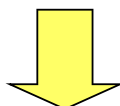
- Ne pas réécrire sans arrêt les mêmes choses
- Mais pouvoir aussi s'adapter si le nouveau contexte le nécessite
- La première forme de réutilisation consiste à recopier un squelette de programme et à le compléter...
- ... Mais une correction dans l'original n'est pas recopiée dans les clones
- L'industrialisation passe par des standards permettant l'interconnexion d'éléments réutilisables

Réutilisabilité

- Deuxième forme de réutilisation : Au lieu de recopier, on réutilise un module déjà construit, éventuellement par d'autres
- Le module améliore la maintenance en séparant clairement les fonctionnalités de leur implémentation
- Principe de boîte noire et d'interface
 - une modification dans la boîte noire ne remet pas en cause les utilisations de la boîte noire
 - n'importe quelle autre boîte noire offrant la même interface est utilisable à la place

Conclusion provisoire...

Faciliter la modélisation du monde réel



La programmation
« orientée objet »



Faciliter & Industrialiser la programmation

... et objectifs du cours

- Comprendre les concepts Objet, c'est connaître les mécanismes de base qui facilitent
 - la modélisation des Objets (mécanismes d'héritage, d'association....)
 - la programmation des Objets (mécanismes de création de classes & instances, accès à l'état d'un objet, définition des comportements d'un objet, ...).

Sommaire

1. Evolution des langages vers l'objet

1. Recule sur un projet Informatique
2. Apparition, historique, évolution et situation actuelle
3. Comparaison avec les approches traditionnelles linéaire, procédurales, modulaire

2. Caractéristiques de l'approche objet

Annexe: Introduction à UML



L'évolution des langages

Lisibilité : ☹️

Réutilisabilité : ☹️

Modularité : ☹️

Langage machine

Codage binaire
01100101100101



L'évolution des langages

Lisibilité : ☹️

Réutilisabilité : ☹️

Modularité : ☹️

Assembleur

Langage machine

Un premier niveau
d'abstraction

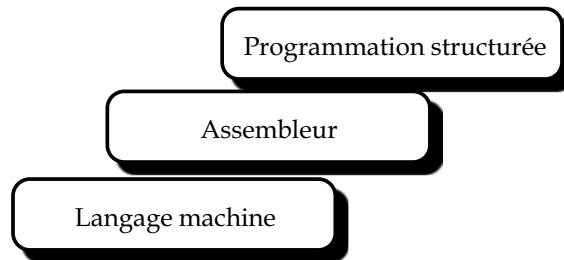


L'évolution des langages

Lisibilité : ☹

Réutilisabilité : ☹

Modularité : ☹



Lisibilité nettement améliorée !

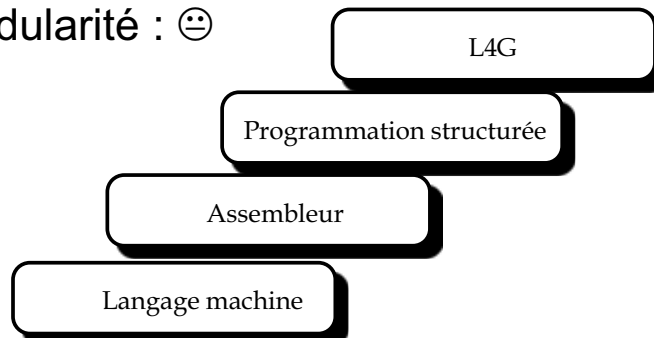


L'évolution des langages

Lisibilité : ☹

Réutilisabilité : ☹

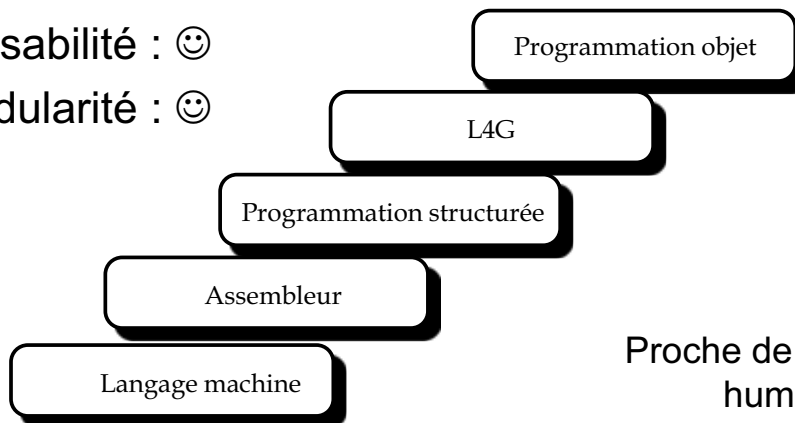
Modularité : ☹



Complexité masquée, productivité

L'évolution des langages

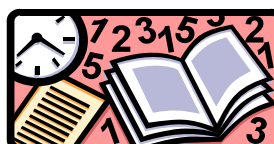
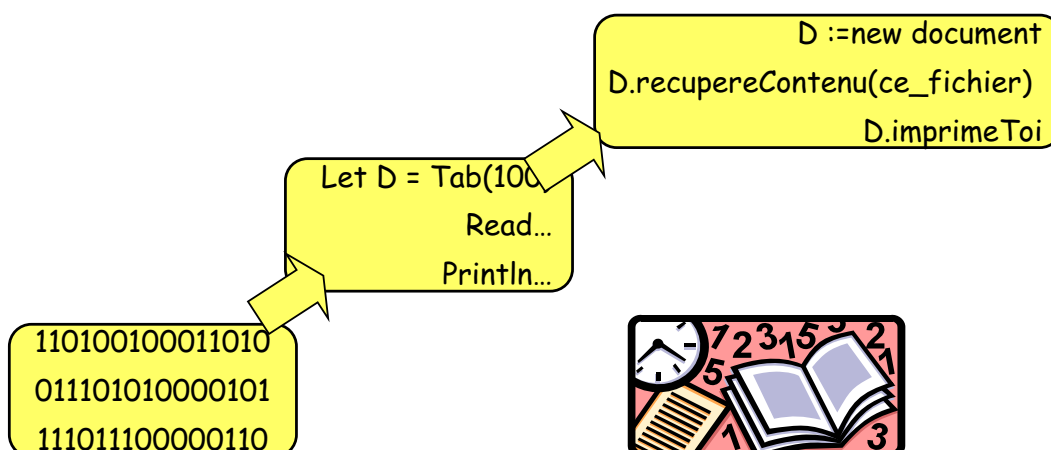
Lisibilité : ☺
Réutilisabilité : ☺
Modularité : ☺



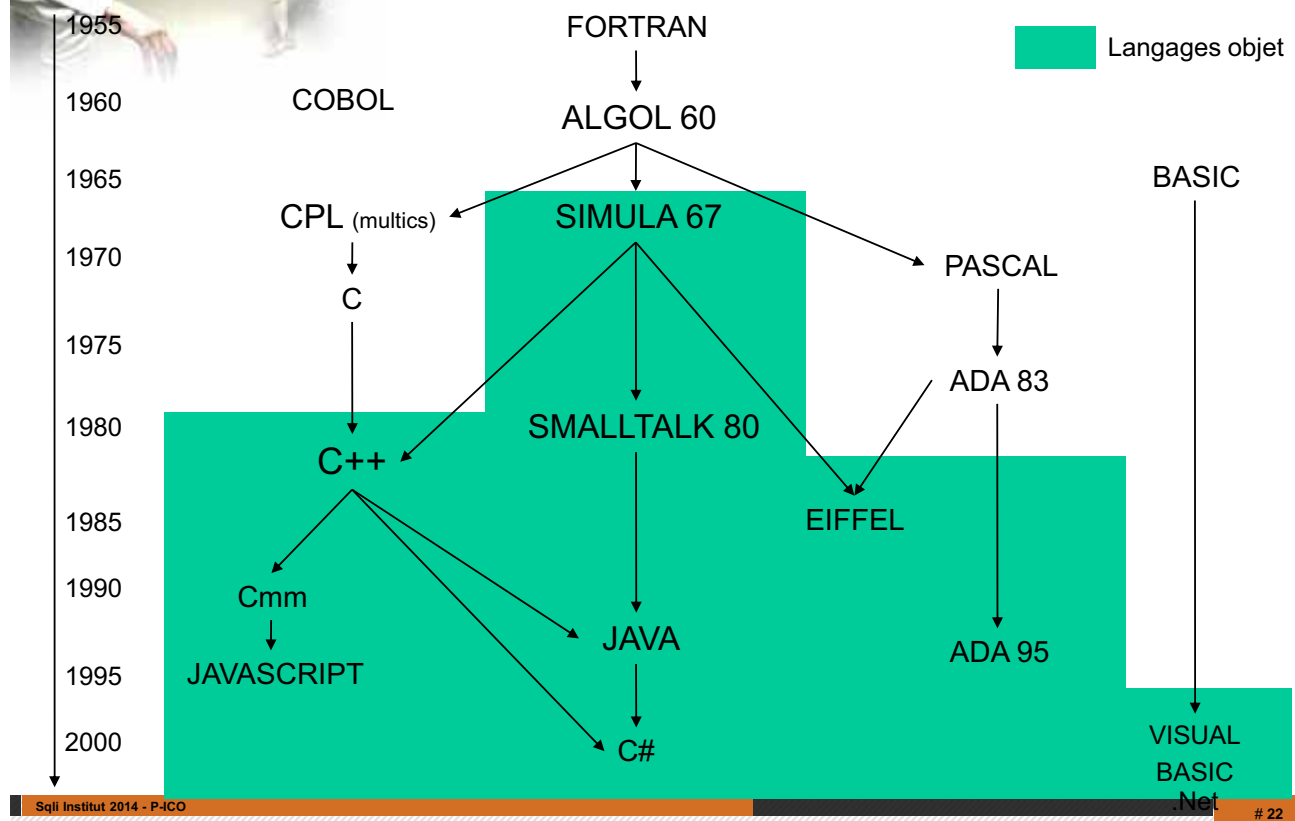
Proche de la pensée humaine

Un champ de vision qui s'élargit

- Au début, la programmation était très focalisée sur la machine
- Au fur et à mesure, la vision du programmeur est devenue de plus en plus macroscopique
- ... jusqu'à ce qu'il puisse s'abstraire de la gestion explicite de certains mécanismes



Historique : synthèse



Historique des langages objet (1)

- **1967 : Simula**
 - apparition de la notion d'objet
 - comme son nom l'indique, ce langage a été initialement conçu pour résoudre des problèmes de simulation
- **1972 – 1980 : Smalltalk**
 - directement inspiré de Simula
 - plusieurs versions avant le célèbre Smalltalk-80
 - 100% objet
 - concept de machine virtuelle
 - aujourd'hui, de très grands projets sont écrits en Smalltalk
 - malheureusement, un manque de consensus de la part des éditeurs et des erreurs marketing ont empêché une large diffusion

Historique des langages objet (2)

- 1983 : C++
 - inspiré de Simula et du langage C
 - très performant (héritage du C)
 - plutôt compliqué (héritage du C)
 - de nombreux programmes écrits en C++ ne peuvent pas être considérés comme étant « objet » (simples extensions de C)
- 1995 (lancement officiel) : Java
 - directement inspiré de Smalltalk, malgré une syntaxe proche du C
 - langage plus simple que C++
 - a directement bénéficié de l'explosion d'Internet
 - stratégie marketing parfaitement maîtrisée (Sun)
 - véritable engouement
 - standardisation (label Java, norme J2EE...)

Autres langages objet

- Objective-C
- Eiffel
- Clos (LISP « objet »)
- D'autres langages ont été « teintés » objet
 - Cobol objet
 - Pascal objet
 - ADA 95 (ADA faisait déjà apparaître certains concepts objet comme l'encapsulation, la généricité...)
 - Visual Basic .Net !!
- C#
 - Langage objet conçu par Microsoft dans le cadre de sa stratégie .Net
 - Clone de JAVA
- Des langages interprétés : Javascript, Ruby, Python

L'objet aujourd'hui

- Des langages matures et standards : Java, C#, Smalltalk, C++
- Une technologie qui donne des résultats
 - simplicité des concepts
 - productivité accrue
 - réduction des coûts de maintenance
- Une technologie adaptée aux nouvelles architectures
 - Internet, Intranet, n-tiers
- ➔ Quelques signes qui ne trompent pas :
 - Utilisation sur tous les OS, y compris mainframe
 - Adoption par les ERP (SAP...)
 - Adoption par Microsoft
 - Adoption par les universitaires (enseignement, recherche)
 - Adoption par la communauté OPEN SOURCE
 - Et adoption par les entreprises !

L'informatique demain

- Les tendances se poursuivent
 - toujours plus d'abstraction
 - toujours plus proche de l'homme
 - toujours plus vite
 - toujours plus interconnectable
- Il y a déjà des recherches sur des évolutions des objets
 - notion d'acteur et d'agent (des objets encore plus intelligents et moins passifs)

Sommaire

1. Evolution des langages vers l'objet

1. Recule sur un projet Informatique
2. Apparition, historique, évolution et situation actuelle
3. Comparaison avec les approches traditionnelles linéaire, procédurales, modulaire

2. Caractéristiques de l'approche objet

Annexe: Introduction à UML

Concept = Objet

- La technologie objet repose sur quelques idées simples :
 - Idée 1 - tout « concept » du monde réel est modélisé dans le système par un objet (= représenter la réalité au plus proche)
 - Idée 2 - les traitements sont rattachés aux objets et l'on utilise la communication par message.
 - Idée 3 - un objet est programmé comme une « boîte noire » : l'état de l'objet n'est pas accessible en direct, mais uniquement via une interface.
 - Idée 4 - Les objets constituent une abstraction de plus haut niveau qui permet de simplifier des problèmes classiques ... ou gérer des problèmes plus complexes
 - Idée 5 - L'assemblage et la maintenance des objets sont beaucoup plus faciles

Exemple simple

Heure = 10
Minute = 30
PM? = 1

L'état de l'objet

```
Affiche l'heure
HeureVisualisée : =
    Heure + PM?*12;
Écrire(« HeureVisualisée »);
    Écrire (« : »);
    Écrire(« minute »);
    Wait(5); Eteindre.
```

Le fonctionnement



Met à jour l'heure

Met à jour les minutes

Incrémente le compteur

Affiche l'heure

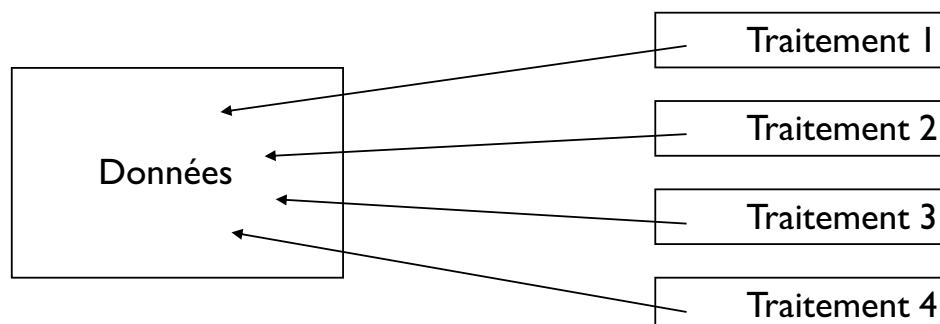
L'interface de l'objet

Une révolution conceptuelle

- Un objet est une entité autonome
 - véritable petit programme informatique à lui tout seul !
 - possède une durée de vie
 - possède un état (données locales à l'objet)
 - possède un comportement...
 - ...accessible par le « monde extérieur » via une interface
- Il n'y a plus de séparation entre traitements et données
 - c'est un véritable changement par rapport à la programmation structurée
- Un programme informatique est constitué d'objets autonomes mais qui vont interagir entre eux

Programmation « classique »

- Les traitements (procédures ou fonctions) accèdent à un ensemble de données
 - la portée des données est vaste

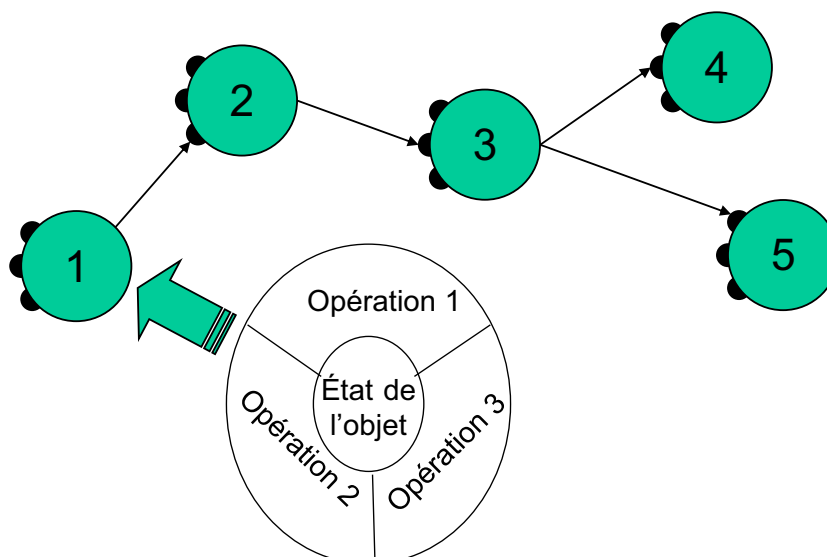


Programmation « procédurale »

- Elle met l'accent sur l'action que doit exécuter un programme
- Cette action sera exécutée par étapes successives de plus en plus fine, jusqu'à obtenir des fonctions élémentaires regroupées en bibliothèques.
- Visibilité des variables locale ou globale.
- Visibilité des fonctions globale.

Programmation objet

- Les interactions entre les objets seront décrites plus loin dans ce cours



Impacts sur la modélisation logicielle

- La modélisation objet diffère d'une modélisation « classique »
 - chaque objet définit les moyens d'interagir avec lui (notion d'interface vis à vis du monde « extérieur »)
 - Je peux demander à mon document de s'afficher sur un écran, de s'imprimer, de s'archiver sur une bande....
 - Beaucoup d'objets se ressemblent entre eux : notion de classes et d'instances, (détaillées plus loin)
 - Le manuel utilisateur powerpoint, le manuel de référence du langage JAVA...sont tous des documents techniques
 - Un objet n'est jamais « seul », il est relié à d'autres objets : notion d'association et d'héritage (détaillées plus loin)
 - Un document comprend des chapitres, un chapitre comprend des paragraphes...
 - les articles de presse, les notes techniques, les catalogues de formation, les relevés bancaires...sont tous des documents, avec un titre, un auteur, une date de création...

Impacts sur la modélisation logicielle (II)

- Le mode de pensée « objet » se rapproche beaucoup plus du raisonnement humain
 - ➔ nécessité de ne plus concevoir uniquement sur la base de structures informatiques
 - ➔ la phase de spécification débute par le repérage des principaux concepts, et leur traduction en objet
- D'où une question clé : « où sont les objets de mon système ? »
 - Le client, le compte bancaire, l'adresse du client...
 - Le document de présentation du produit d'assurance X, la facture, la lettre de relance...
 - L'imprimante, l'écran du PC...

Sommaire

1. Evolution des langages vers l'objet
2. Caractéristiques de l'approche objet
 1. Présentation des concepts, philosophie
 2. Abstraction, encapsulation, modularité, hiérarchie, héritage, polymorphisme
 3. L'Objet dans les projets

Annexe: Introduction à UML

Qu'est ce qu'un objet ?

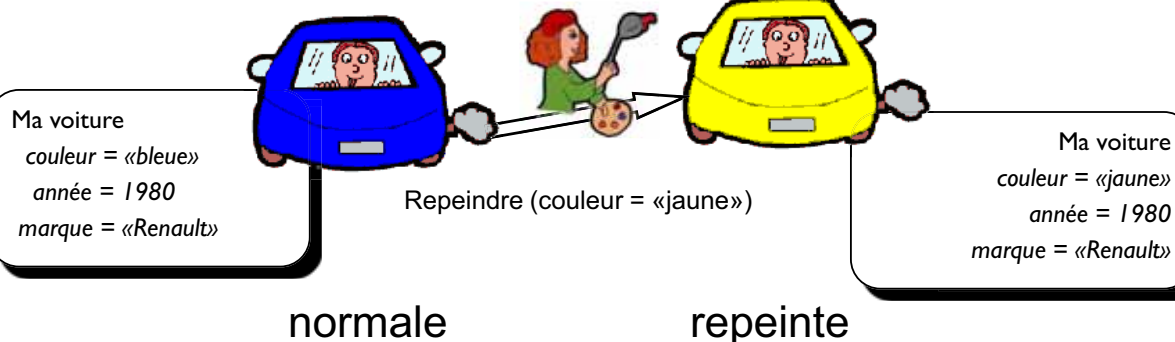
- C'est une entité logicielle qui possède
 - une identité
 - un état
 - un comportement, spécifié par son interface
 - un cycle de vie
- Un objet possède donc une partie statique (identité, comportement) et une partie dynamique (état, cycle de vie)

Etat d'un objet

- Un objet possède des données (entiers, chaînes de caractères, autres objets...)
- Ces données sont appelées « Attributs », « Champs » ou « Variables d'instance »

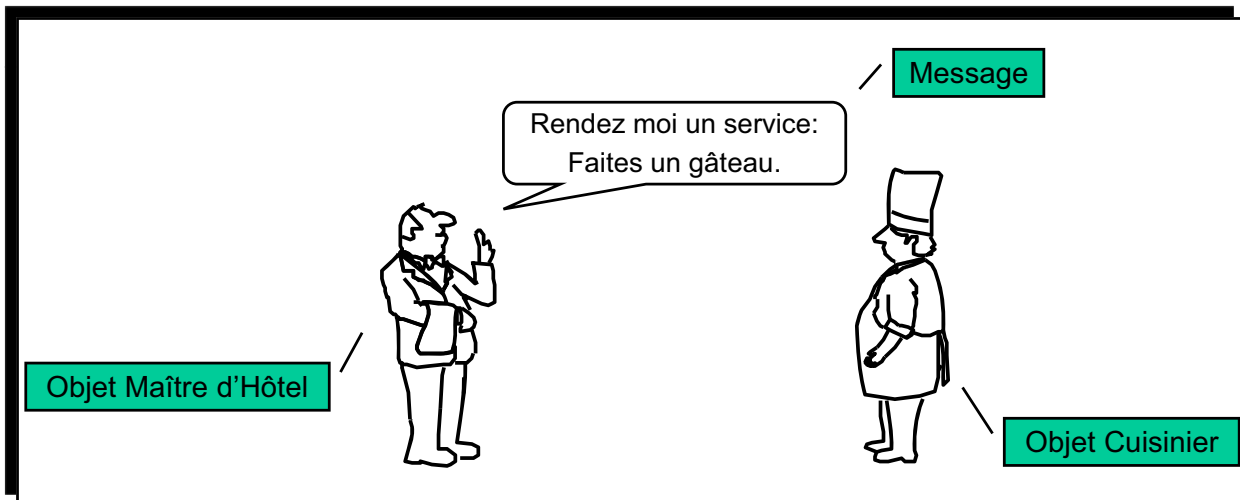
Voiture
couleur : chaîne
année : entier
marque : chaîne

Les valeurs de ces données forment l'état de l'objet



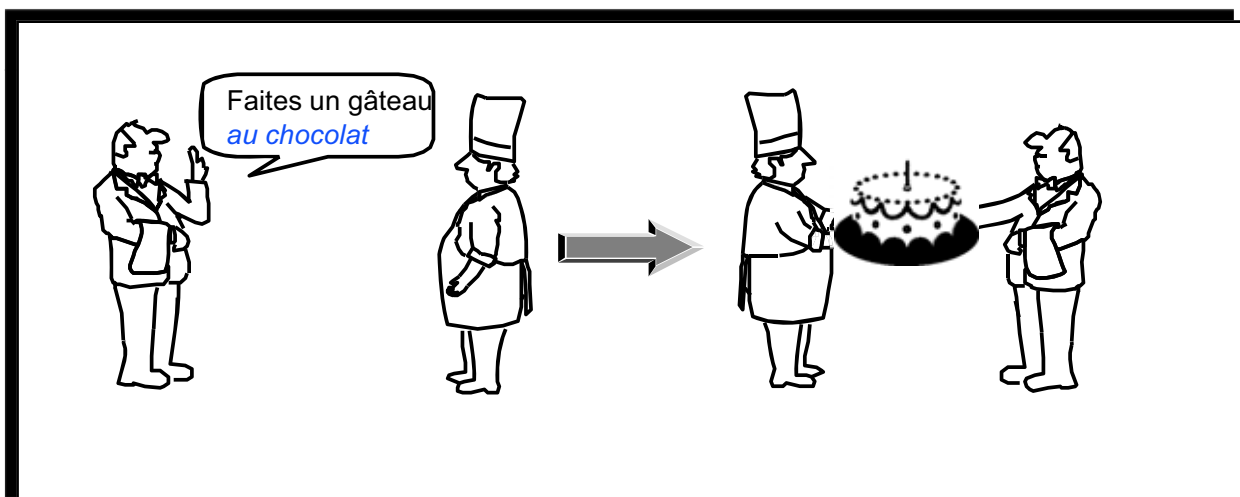
Comportement d'un objet services et messages (1)

- Service = traitement qu'un objet peut effectuer
- Interface = l'ensemble des services qu'un objet peut fournir
- Message = le moyen de déclencher un service de l'objet



Comportement d'un objet services et messages (2)

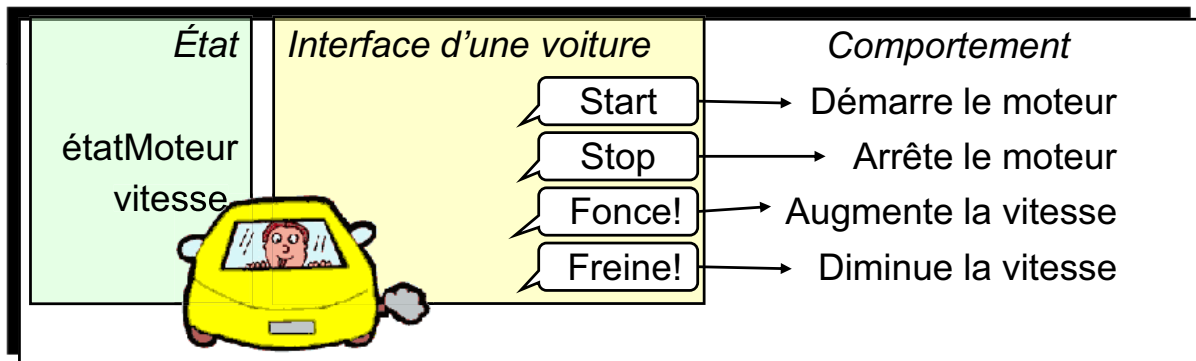
- Un message peut inclure des paramètres
- Un message implique souvent une réponse



Comportement d'un objet comportement et méthodes (1)

Le comportement d'un objet représente la façon dont il réagit aux messages qu'il reçoit.

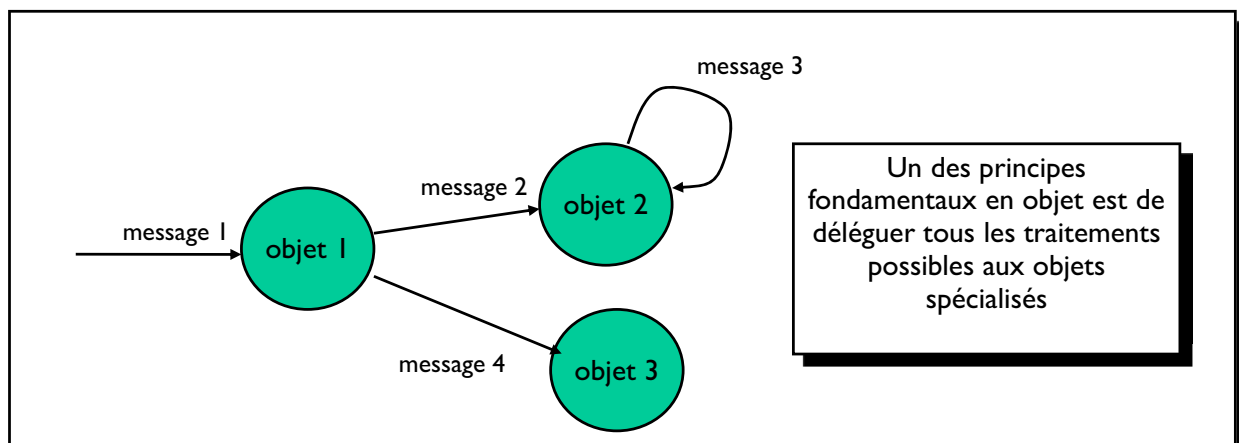
Le comportement dépend de l'état de l'objet, et peut le modifier



- Méthode = implémentation d'un service de l'interface
elle est codée par le concepteur du programme

Comportement d'un objet comportement et méthodes (2)

- L'exécution d'une méthode peut déclencher l'exécution d'autres méthodes (par de nouveaux envois de messages)
 - Dans l'objet lui-même
 - Dans d'autres objets (délégation)



Exercice 1

- **Modéliser un document :**
 - Définir ses principaux attributs
 - Donner des exemples de valeurs d'attribut
 - Définir son interface

Identité

- Tout objet est une entité bien définie, tout comme l'est une personne, votre voiture, la ville de Paris, ...
- Mot clé pour se désigner soi-même : *this* ou *self*
- Mêmes valeurs d'état n'impliquent pas mêmes objets

Voiture 1



Couleur devient « bleu »



Voiture 2



Stop



Exercice 2

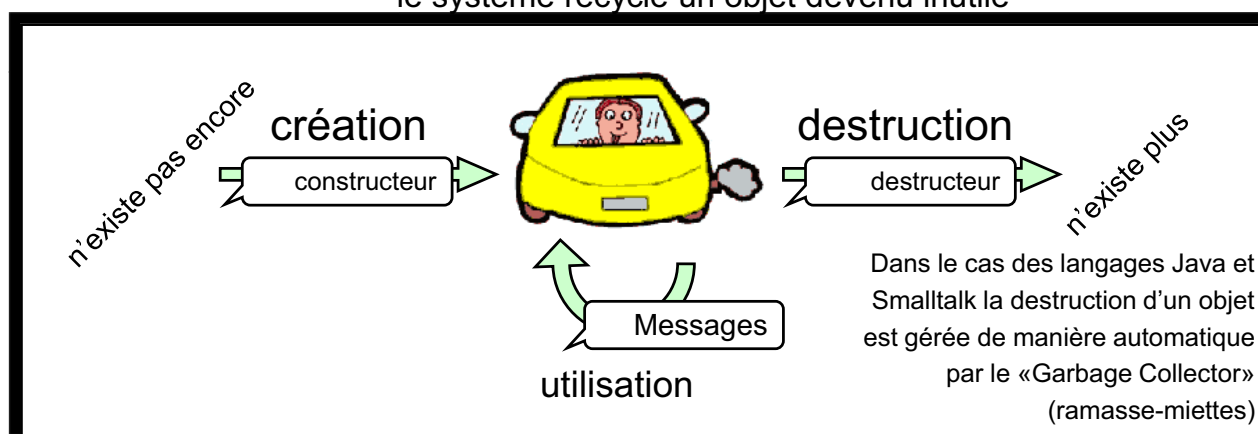
- Illustrons l'utilisation de « self » pour écrire (en pseudo code) la méthode « imprimeToi » de l'objet Document...
 - On commence par imprimer le titre, puis le texte du document, puis l'auteur
- On admet qu'il existe un objet de base « chaine », capable de recevoir des chaînes de caractères de longueur indéfinie, et qui offre un service d'impression « print »
 - On écrit (par exemple) :


```
chaîne ma_chaine ;
ma_chaine := « ceci est une chaîne » ;
ma_chaine.print ;
...
```

Cycle de Vie

Un objet est créé, utilisé puis détruit

- Création = le système fait appel à un « constructeur »
- Utilisation = les autres objets font appel aux services
- Destruction = le développeur fait appel au « destructeur »
le système recycle un objet devenu inutile





Le « garbage collector » (Java, C# et Smalltalk)



- Les objets qui ne sont plus utilisés sont automatiquement détruits → évite la prolifération des objets et économise ainsi la mémoire RAM
- Mécanisme très utile
 - Facilite grandement le travail des développeurs
- Un objet n'est plus utilisé lorsqu'il n'est plus référencé
 - Cette notion est détaillée dans les formations spécifiques aux langages Java ou Smalltalk
- Le « garbage collector » agit de façon ponctuelle
 - Un objet « inutile » n'est pas immédiatement détruit mais seulement lorsque le « garbage collector » se déclenche



La langage UML comme support de modélisation



- Le langage UML comme support de modélisation
- Les mécanismes de base des langages orientés objet :
 - Encapsulation & accesseurs
 - Classes & instances
 - Relations entre classes : association et héritage
 - Polymorphisme

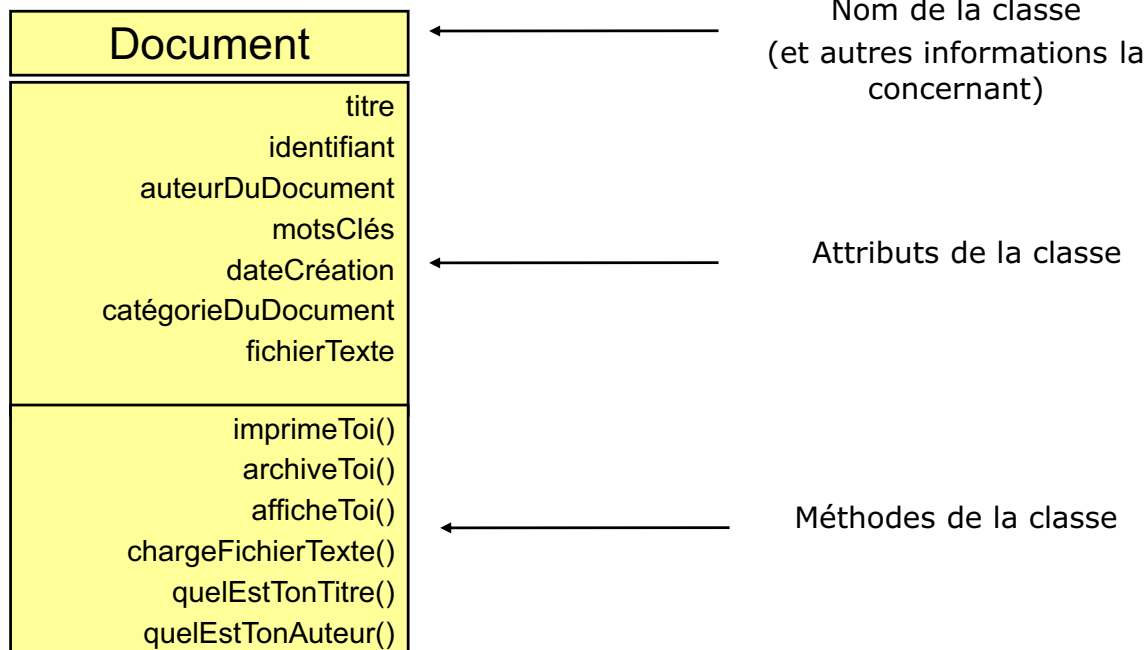
Utilisation du langage UML

- UML est un langage de modélisation objet
 - 9 diagrammes standardisés (facettes complémentaires d'un système)
 - Support des phases d'Analyse et de Conception orientée objet
- UML est un langage de communication
 - utilisation d'un même formalisme par tous les intervenants
 - permet de lever les ambiguïtés du langage naturel
- UML est un langage simple de haut niveau
 - facile à appréhender car visuel
 - indépendant de tout langage de programmation

Le diagramme de classes

- S'apparente à un **diagramme entité-association** (MERISE). Il présente les différents objets (classes) du système ainsi que les liens entre ces objets (associations)
- Diagramme UML **le plus important** dans une modélisation objet
- Notation utilisée largement pour représenter des classes, souvent même de manière simplifiée
 - Cf. les quelques schémas déjà présentés
- Dans la suite de cette formation, on continuera d'utiliser UML pour représenter les modèles de classes

Exemple UML



Sommaire

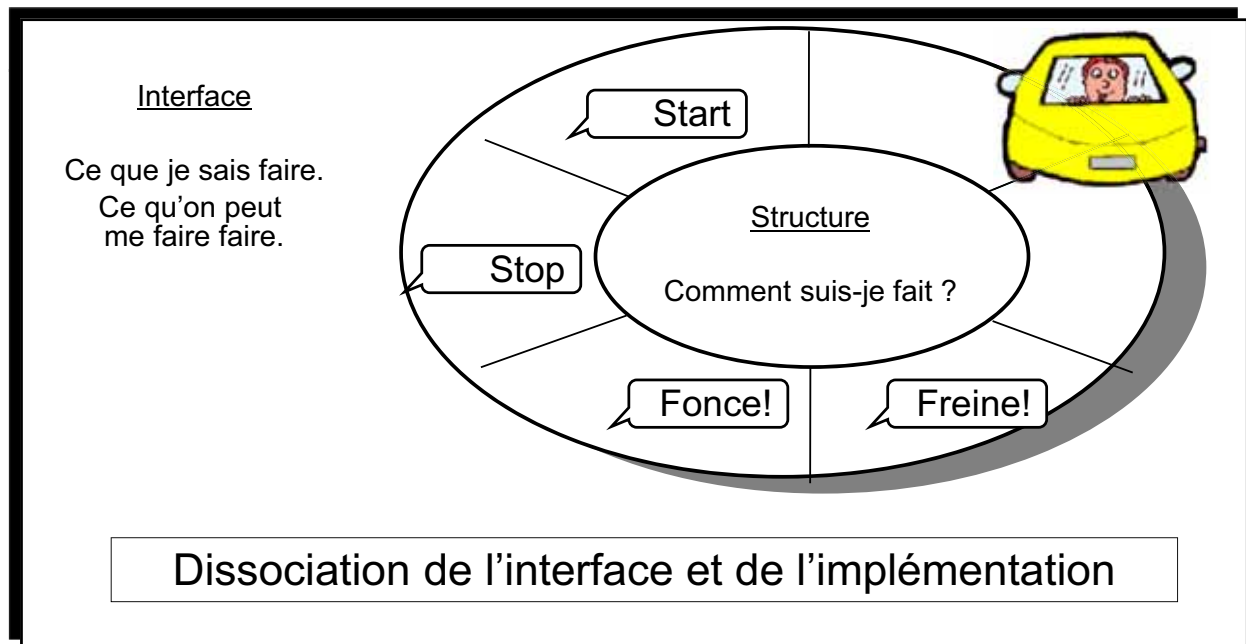
1. Evolution des langages vers l'objet
2. Caractéristiques de l'approche objet
 1. Présentation des concepts, philosophie
 2. Abstraction, encapsulation, modularité, hiérarchie, héritage, polymorphisme
 3. L'Objet dans les projets

Annexe: Introduction à UML

Encapsulation

Principe de la boîte noire

- Boîte noire = on ne voit pas ce qu'il y a à l'intérieur



Encapsulation

Avantage

- **Concept fondamental des technologies objet**
 - focalisation sur les services rendus par les objets plutôt que sur leur structure
- **Facilite la maintenance**
 - protection des objets
 - accès uniquement par l'interface de l'objet
 - les données internes (attributs) sont inaccessibles de l'extérieur, il faut passer par l'interface de l'objet pour les manipuler
 - → par exemple, possibilité de tracer tous les accès à un attribut...
 - évolution des applications
 - Pouvoir changer l'implémentation d'un objet sans changer tous les logiciels utilisant l'objet, sans effet de bord désastreux

Encapsulation Niveaux de visibilité

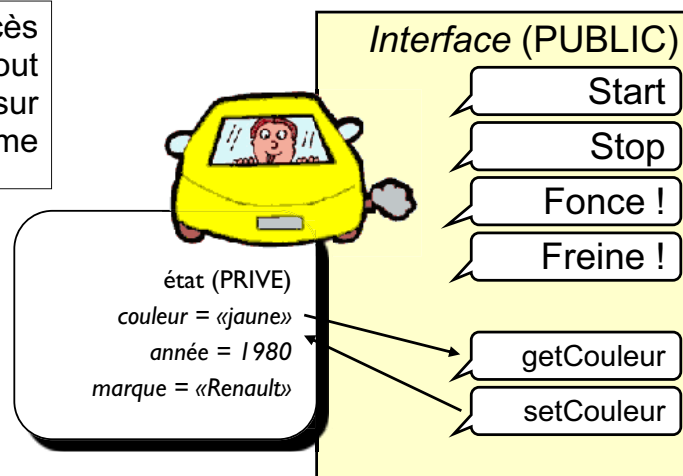
- Certains langages (Java, C++) proposent des niveaux de visibilité pour les attributs et les méthodes
 - **public** : visible par tous les autres objets
 - **protected** : visibilité limitée à un ensemble d'objets « proches »
 - **private** : visible seulement depuis l'intérieur même de l'objet
 - seul l'objet lui-même peut s'envoyer des messages privés, et voir ou modifier des attributs privés
- D'autres langages imposent les niveaux de visibilité (Smalltalk)
 - toutes les méthodes sont publiques
 - tous les attributs sont privés

Encapsulation Accesseurs

- Les attributs devraient toujours être privés !
- Ils peuvent être manipulés grâce à l'envoi de messages appelés accesseurs (« getxxx » et « setxxx »)

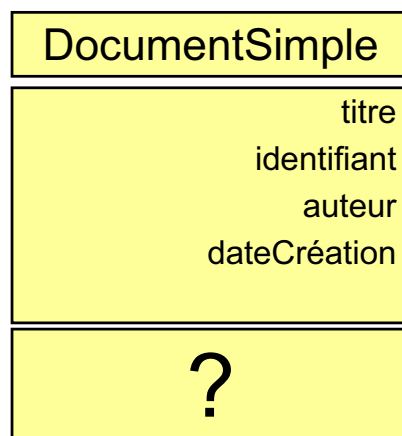
Les accesseurs permettent l'accès et la modification des attributs, tout en gardant un contrôle total sur l'attribut même

(bornes min/max, conversion, ...)



Exercice 3

- Définir les accesseurs de l'objet Document (simplifié !)



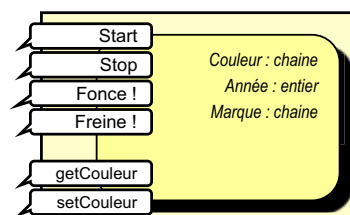
Typage Concept de classe (1)

La définition d'une catégorie d'objets est appelée « Classe »

- Classe**
 - regroupe tous les objets correspondant au même concept
 - définit les attributs
 - définit l'interface de ces objets
 - définit l'implémentation
 - permet de créer les objets (→ chaque classe fournit un constructeur)



Instances : des objets voitures



Classe : Voiture

Typage + Concept de classe (2)

La classe permet de créer des instances

Java (constructeur)
new Voiture();

Smalltalk
Voiture new.

La classe définit la structure de ses instances

Java
public class Voiture {
 String couleur;
 int annee;
 String marque;
 ...
}

Smalltalk
Object subclass: #Voiture
instanceVariableNames: 'couleur
annee marque'
...

Voir page suivante
→

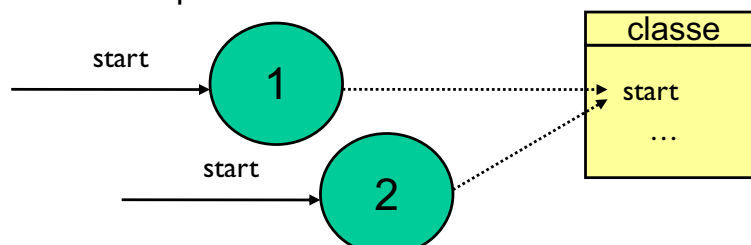
Typage + Concept de classe (3)

La classe définit le comportement de ses instances

Java
(dans la classe Voiture)
public void start() {
 ...
}

Smalltalk
(dans la classe Voiture)
start
...

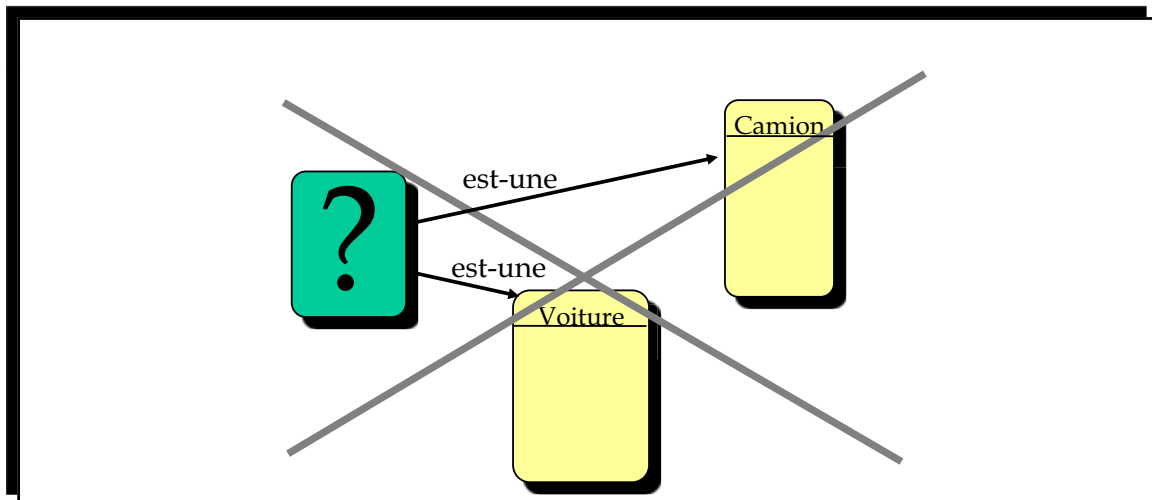
Lorsqu'une instance reçoit un message, elle recherche la méthode correspondante dans la classe dont elle est issue



Typage

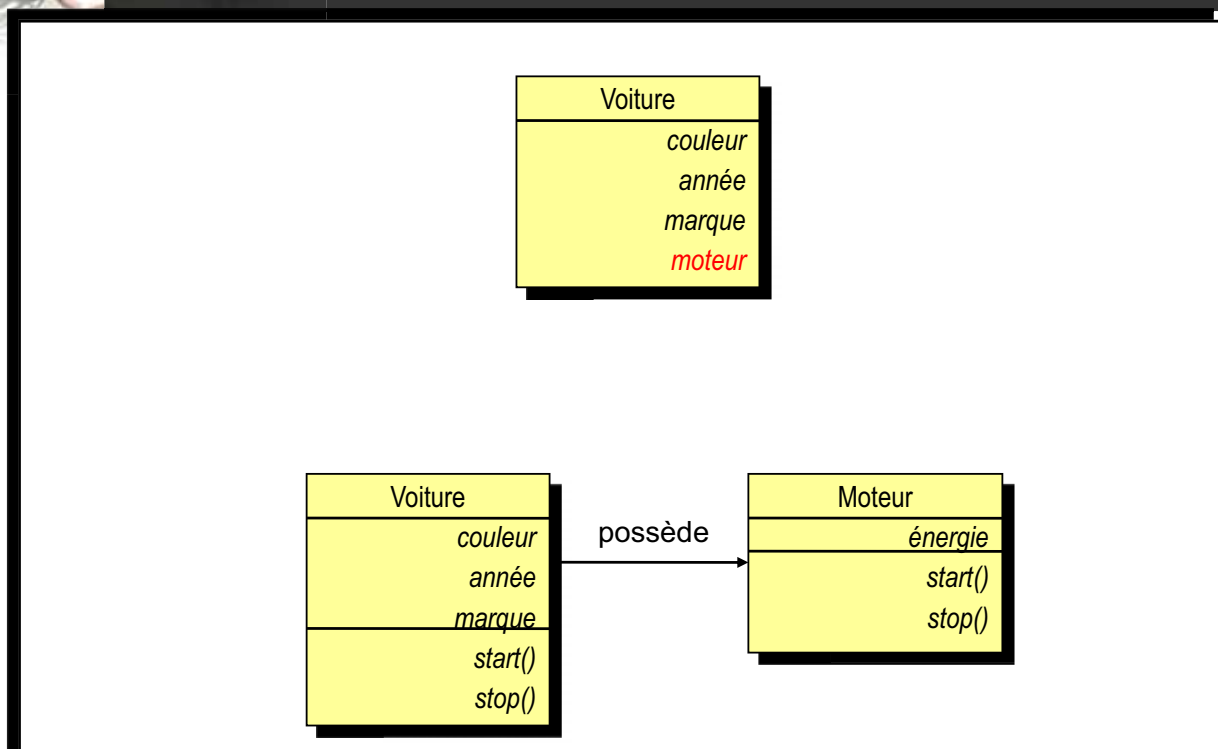
Contraintes sur l'instantiation

- Un objet est toujours instance d'une et d'une seule classe
 - rappel : la classe définit la structure et l'interface (les services)
- Un objet ne change pas de classe au cours de son existence

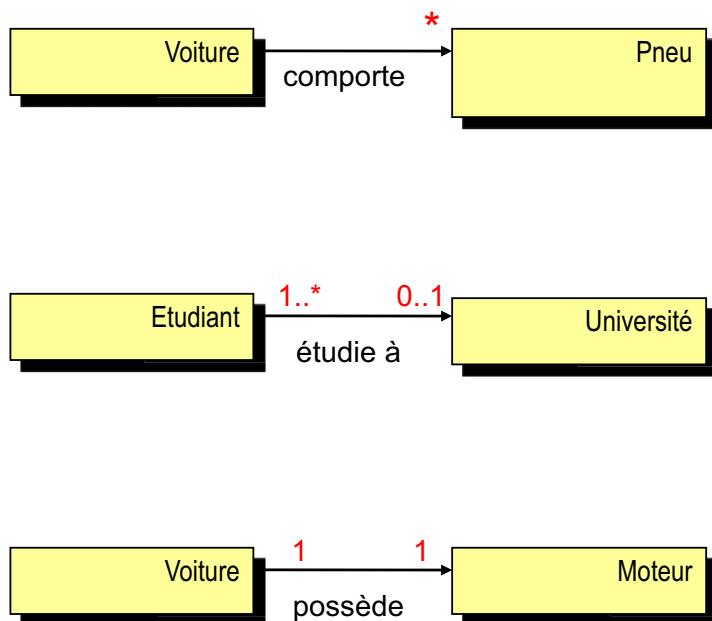


Relations entre Objets

Associations



Relations entre Objets Cardinalités



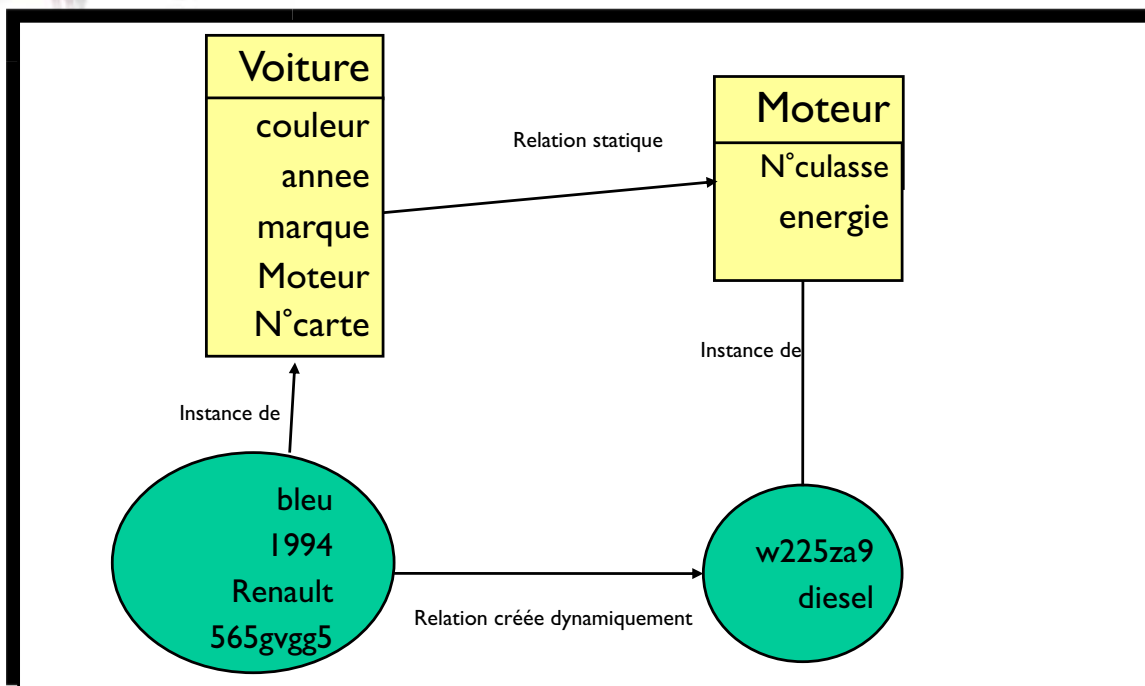
Une voiture comporte **plusieurs pneus**

Un étudiant étudie dans **au plus une** université. Dans chaque université, il y a **au moins un** étudiant

Une voiture possède **exactement un** moteur.

1 est la cardinalité par défaut.

Relations entre Objets Création dynamique des relations (1)



Relations entre Objets

Création dynamique des relations (2)

- Les relations entre classes définissent les relations que les objets peuvent avoir entre eux ;
- Les relations entre objets sont créées dynamiquement
 - il faut obligatoirement que les instances (objets) soient créées avant d'établir une relation entre elles
 - une relation est simplement établie en affectant une instance à un attribut

```
uneVoiture = new Voiture();
unMoteur = new Moteur();
uneVoiture.moteur = unMoteur;           // Attention : encapsulation non respectée !
```

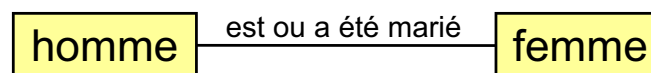
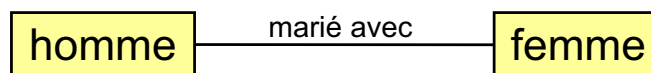
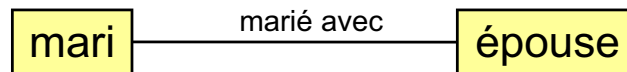
// le code correct serait plutôt : `uneVoiture.setMoteur(unMoteur);`

Exercice 4

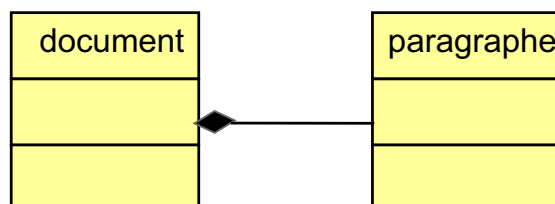
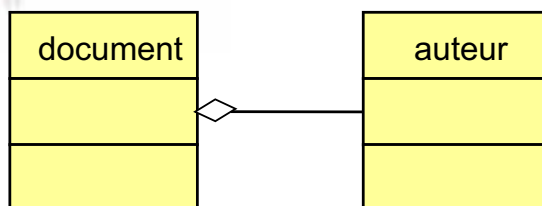
- Reprenons la modélisation de nos documents : introduisons un autre objet, l'auteur :
 - Définissez l'objet « auteur »
 - Reliez l'objet « document » et l'objet « auteur », c'est à dire définissez les cardinalités

Exercice 5

➤ Définissez les cardinalités des associations suivantes



Relations entre objet agrégation ou composition ?



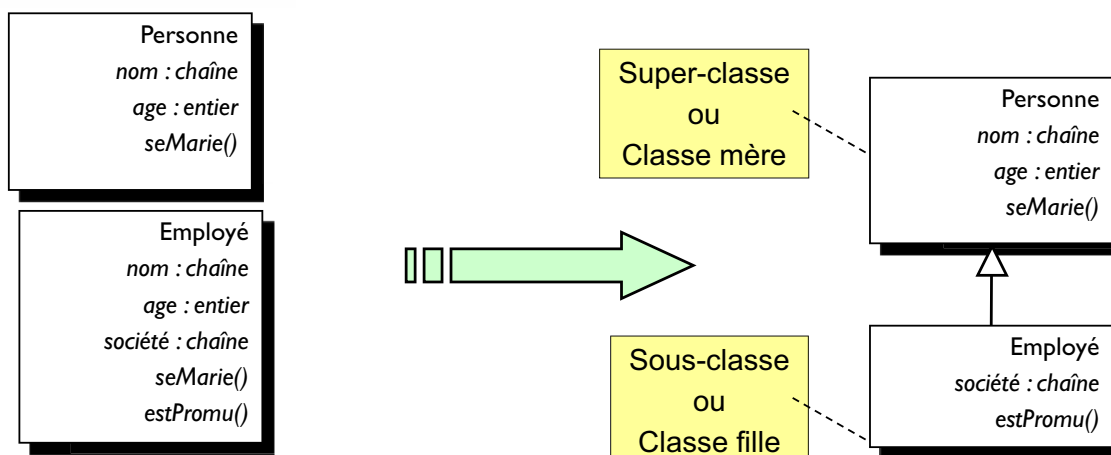
Héritage : le besoin

- Un employé est « une sorte de » personne
- Un homme est « une sorte de » mammifère
- Une voiture, un camion, un car sont « des sortes de » véhicules motorisés ; un véhicule motorisé est « une sorte de » véhicule
 - Une voiture, un camion, un car partagent donc le fait d'avoir un moteur à explosion, des roues etc.
 - Ils diffèrent par ce qu'ils transportent (des passagers, des marchandises...)

➔ comment éviter de réécrire pour chaque classe (voiture, camion...) les attributs et les comportements communs ?

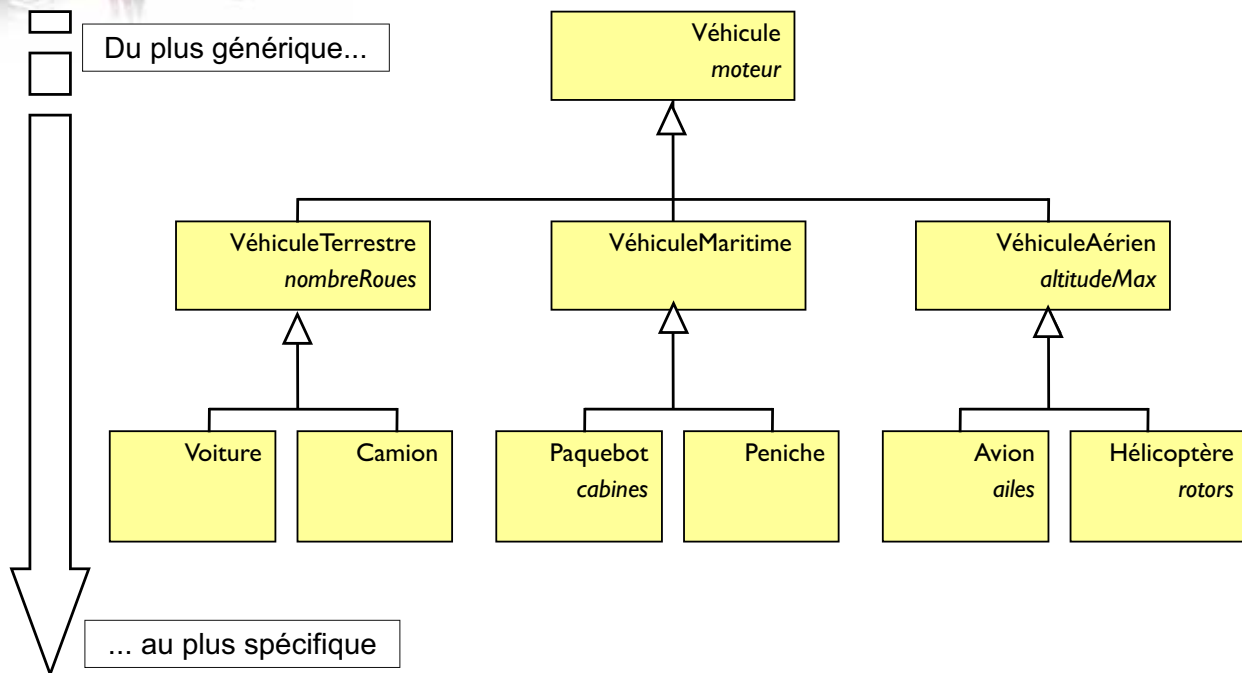
Héritage Règle du « est une sorte de »

- Un employé est une sorte de personne, travaillant dans une société



L'héritage permet la mise en commun d'attributs et de comportements

+ Héritage + Arborescence



+ Arrêtons nous un instant... + les relations entre objets

- Relation d'instanciation
 - La Peugeot 956 CJH 75 « est une » voiture
- Relation d'héritage
 - Une voiture « est une sorte de » véhicule terrestre
- Relation d'association (ou de composition)
 - Une voiture « contient » un moteur
 - Une voiture « appartient à » un propriétaire

Exercice 6

- Apprenons à reconnaître les différents types de relation...
 - les relations ci-dessous sont-elles des relations d'héritage, d'association ou d'instanciation ?

Voiture – Pneus
 Renault – ConstructeurAutomobile
 Camion – Véhicule
 2 – Entier
 Entier – Nombre
 Fraction – Entier
 ChaîneDeCaracactères – Caractère
 Rectangle - Carre

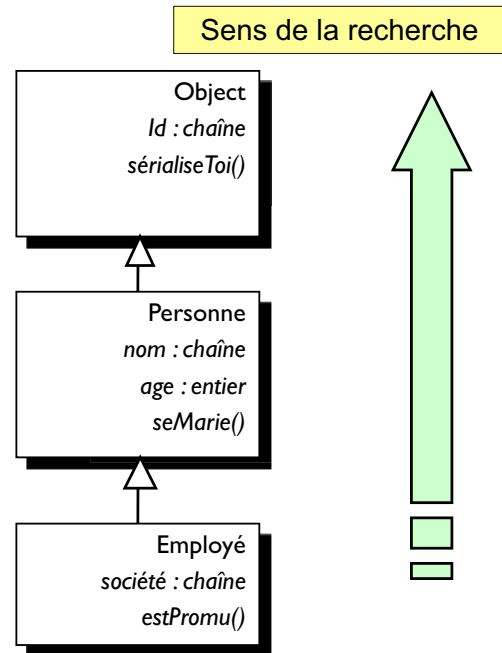
Héritage Racine

- L'ensemble de toutes les classes forme un seul arbre
- La racine est souvent nommée **Object**
- **Object** offre des services universels génériques
 - représentation textuelle de tout objet
 - écriture de tout objet sur un disque (sérialisation)
 - clonage d'un objet
 -
- Cette classe ne possède pas d'instance
- Cette racine unique n'existe pas pour certains langages comme le C++

Héritage

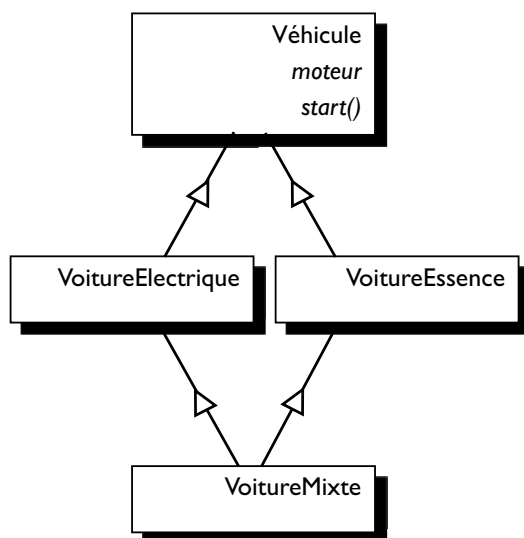
Recherche d'une méthode

- La recherche de la méthode à exécuter en réponse à un message s'effectue en « remontant » l'arbre d'héritage vers sa racine
 - La recherche débute dans la classe de l'objet receveur
 - La recherche s'arrête dès la première implémentation trouvée
 - Une erreur est déclenchée si aucune implémentation n'est trouvée
 - à la compilation pour les langages typés (Java, C++, C#)
 - à l'exécution pour les langages non typés (Smalltalk, CLOS)



Héritage

Héritage multiple



- N'est pas supporté par tous les langages
- Semble naturel, mais complique énormément l'implémentation
- Génère des difficultés...
 - La voiture mixte a-t-elle un moteur ou deux moteurs ?
 - Quel moteur est démarré par le service *start()* ?

On peut généralement s'en passer, grâce à une bonne conception



+ Héritage + Utilisation

- L'utilisation de l'héritage implique une vision statique du problème
 - ne pas oublier que le but principal de l'héritage est de factoriser structure et comportement
- → Utiliser l'héritage à bon escient
 - en cas de doute lors de la conception, il est déconseillé d'utiliser l'héritage car la maintenance de l'application pourra s'avérer très délicate en cas de réorganisation d'une hiérarchie de classes
 - De même, il faut se méfier de la prolifération des classes filles
 - très souvent, l'association entre objets est préférable : le logiciel obtenu est dynamique (utilisation de la délégation)



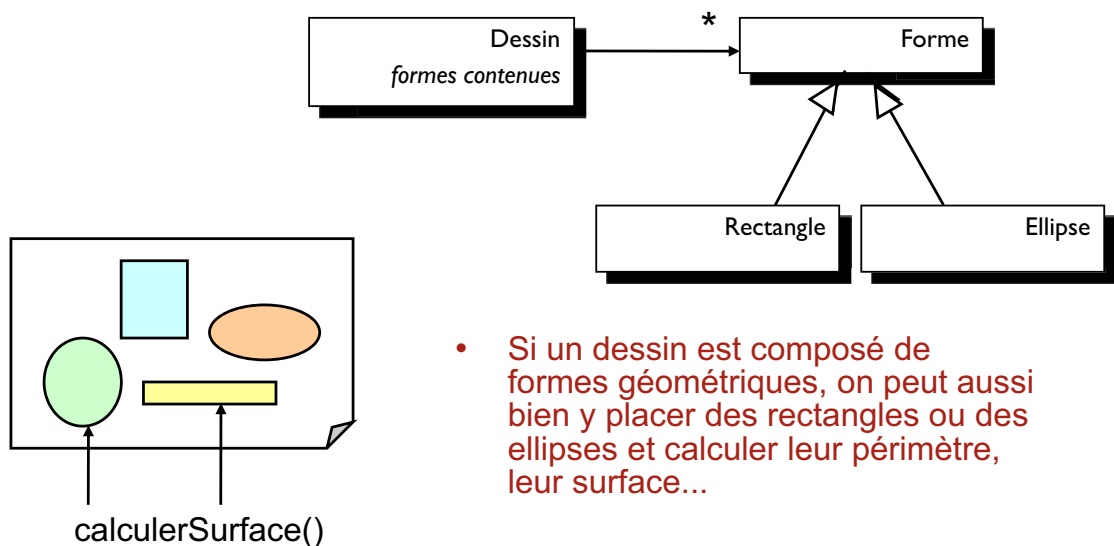
+ Exercice 7

- Modélisons une bibliothèque capable de recevoir l'ensemble de la production des éditeurs d'un pays :
 - Définissez un arbre d'héritage de la famille des documents...
 - Les différents documents possibles : les livres, les articles de presse, les catalogues (d'art, de VPC)...

- Reprenons l'exemple de la bibliothèque : nous devons modéliser la gestion d'une bibliothèque électronique gérée par les responsables d'une cellule de veille économique.
 - La bibliothèque va contenir toutes sortes de documents électroniques (article de presse, note technique, photo...), on ne connaît pas à l'avance les différentes catégories de document.
 - Une catégorie définit notamment le format du document (.PDF, .DOC, .TXT, .JPEG, .AVI...etc), c'est donc une information importante pour afficher à l'écran ou imprimer le document
 - Comment alors modéliser le logiciel, pour qu'un bibliothécaire (c'est à dire un « non informaticien ») puisse lui-même ajouter ultérieurement de nouvelles catégories de document (vidéo, etc), sans devoir faire évoluer le logiciel ?
- Problématique courante

Polymorphisme Définition

Capacité pour une classe fille à se faire passer pour sa classe mère. Une classe fille peut rendre les mêmes services que sa classe mère mais selon une implémentation qui lui est propre.

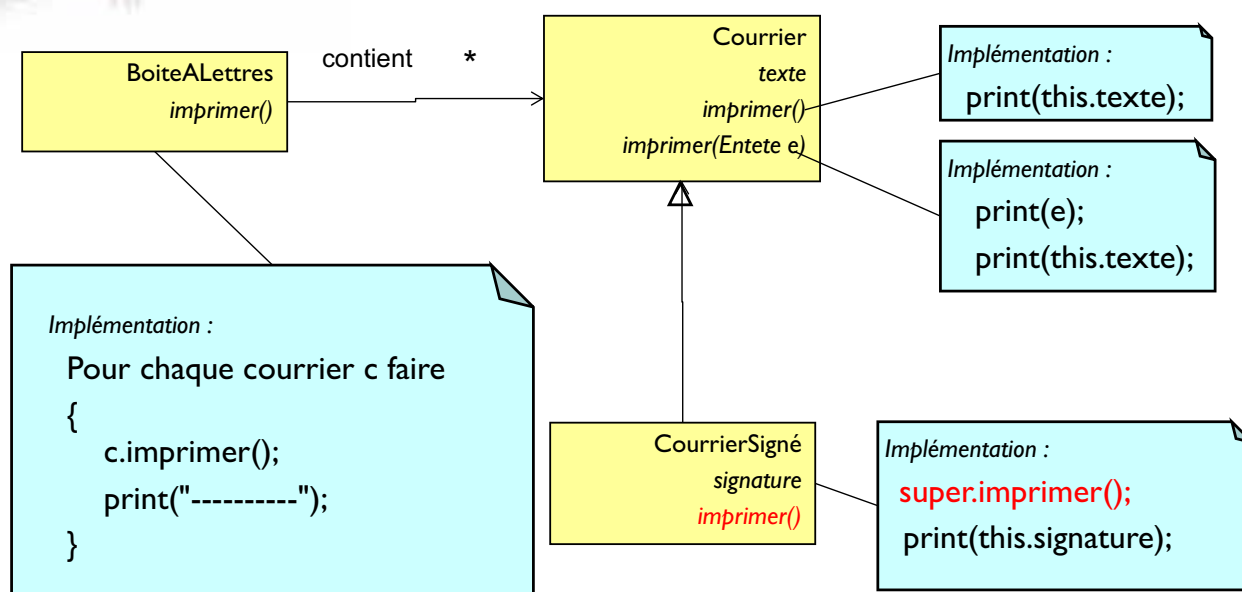


- Si un dessin est composé de formes géométriques, on peut aussi bien y placer des rectangles ou des ellipses et calculer leur périmètre, leur surface...

Surcharge et redéfinition

- **Surcharge** = plusieurs méthodes de même nom (mais avec des signatures différentes) peuvent être définies dans la même classe
 - la signature d'une méthode est déterminée par le nombre d'arguments et leur type (ex: calculPrixTTC ())
- **Redéfinition** = une méthode est redéfinie dans une sous-classe pour modifier le comportement hérité
 - la signature d'une méthode redéfinie doit être strictement identique à la méthode d'origine (de la super-classe)
 - certains langages imposent des restrictions sur la visibilité de la méthode redéfinie par rapport à la méthode d'origine
 - en Java, une méthode redéfinie ne peut lever plus d'exceptions que ne le présume ses super-classes

Polymorphisme redéfinition et surcharge (1)



Polymorphisme redéfinition et surcharge (2)

maBoite : BoiteALettres

C1 : Courrier
texte = «Bonjour»

C2 : CourrierSigné
texte = «Ce soir...»
signature = «Charles»

C3 : CourrierSigné
texte = «Comment ...»
signature = «André»

C4 : Courrier
texte = «Je viens...»

Résultat de
maBoite.imprimer() :

Bonjour

Ce soir...
Charles

Comment...
André

Je viens...

Classes et méthodes abstraites (1)

Les classes abstraites n'ont pas d'instances

- Elles définissent des concepts généraux ou abstraits
 - un véhicule n'est rien s'il n'est pas une voiture ou un camion ou ...
 - un mammifère est un chien ou un chat ou un homme ou ...

Les méthodes abstraites correspondent au comportement non définissable d'une classe

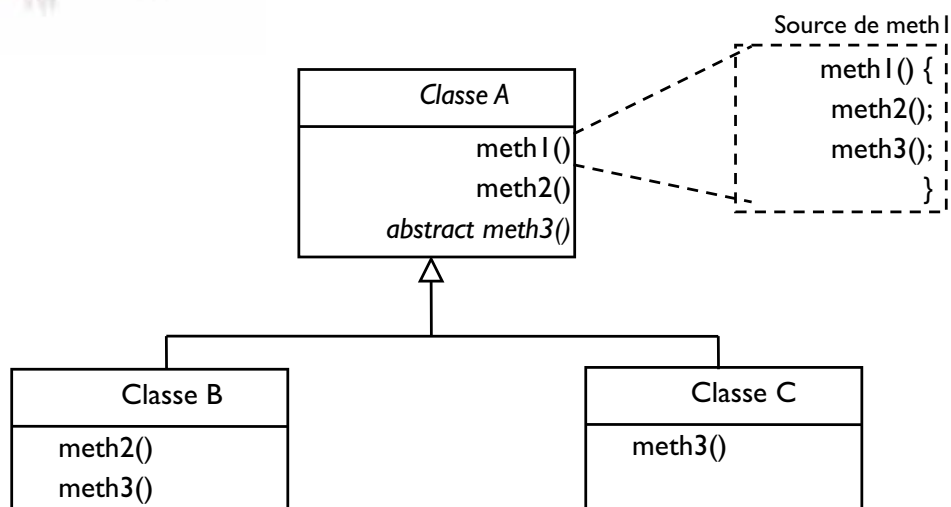
Toute figure géométrique peut être dessinée. Mais on ne sait dessiner que des carrés, des cercles, etc. On ne peut pas définir le dessin d'une figure géométrique de manière générale.

Une classe possédant une méthode abstraite ou plus, est automatiquement une classe abstraite.

Classes et méthodes abstraites (2)

- Certains langages proposent une syntaxe particulière pour la définition de classes et de méthodes abstraites
 - Java, C++ par exemple
 - Ils imposent des restrictions d'utilisation
- D'autres ne le proposent pas
 - Smalltalk par exemple
 - Ils laissent une liberté aux développeurs
- Une classe abstraite peut imposer à ces sous-classes de fournir une implémentation à une méthode à la signature spécifiée
- Selon les langages, une implémentation par défaut peut être spécifiée pour une méthode abstraite (Cf. C++)


Classes et méthodes abstraites (3)




L'envoi d'un message meth1 à une instance de la classe B déclenche l'exécution des méthodes meth2 et meth3 de la classe B.

L'envoi d'un message meth1 à une instance de la classe C déclenche l'exécution de la méthode meth2 de la classe A et de la méthode meth3 de la classe C.

Notez que la classe A est abstraite.



Le Tout-Objet Gestion des erreurs



- On ne se repose plus sur un code de retour
- Pour signaler une erreur on lève une exception
 - La signature de la méthode peut indiquer les erreurs potentielles
 - Le compilateur peut vérifier que le code appelant gère les éventuelles erreurs
 - La gestion des situations exceptionnelles ne pollue plus le code « normal »
- Les exceptions sont des objets comme les autres
 - Toutes les erreurs du langage sont organisées en une hiérarchie
 - Le développeur peut créer de nouvelles exceptions



Les erreurs à éviter



- Coder des méthodes trop longues
 - Utiliser si nécessaire des méthodes privées
- Ne pas utiliser correctement la délégation
 - Un objet ne doit réaliser que les opérations qui le concerne
 - Par exemple éviter : `unObj.meth1().meth2().meth3()`
- Coder des successions de tests (style procédural)
 - Traduit un manque de délégation et de polymorphisme
- Demander à un objet de quelle classe il est issu
- Se tromper entre la composition et l'héritage

Résumé

- **Qu'est-ce qu'un objet ?**
 - une entité typée, possédant des données et un comportement
 - a un cycle de vie
 - communique par messages
- **Puissance de l'objet**
 - Association (dynamisme des logiciels, puissance de la modélisation)
 - héritage (factorisation, programmation par différence)
 - polymorphisme (adaptabilité, généricité, flexibilité)
- **Avantages de l'objet**
 - vue plus naturelle
 - facilite l'évolution
 - orienté réutilisation

Sommaire

1. Evolution des langages vers l'objet
2. Caractéristiques de l'approche objet
 1. Présentation des concepts, philosophie
 2. Abstraction, encapsulation, modularité, hiérarchie, héritage, polymorphisme
 3. L'Objet dans les projets

Annexe: Introduction à UML

Impact culturel de l'objet

- L'objet a été très largement promu par Java et est aujourd'hui considéré comme un bon candidat pour toutes sortes de projets
- On retrouve l'objet dans :
 - **L'édition logicielle**
 - Beaucoup d'outils de développement sont notamment développés dans leur propre langage
 - **Les applications d'entreprise et systèmes d'information**
 - B2B, B2C, intranet, extranet, ...etc.
 - **Les applications embarquées**
 - Java tourne par exemple sur des téléphones portables
- Connaître le paradigme Objet est aujourd'hui un point fort

La formation : une étape essentielle

- Le paradigme Objet permet de modéliser le monde qui nous entoure de manière plus simple
 - On est habitué dans la vie de tous les jours à manipuler des objets
- Il n'est cependant pas facile à prendre en main
 - Il fait appel à des abstractions qu'il est nécessaire de maîtriser pour tirer toute la puissance de l'Objet
 - Des mécanismes encore plus complexe sont implémentés au dessus de l'objet
- Une formation est nécessaire pour bien mettre le pied à l'étrier
 - Notamment pour les personnes qui ont appris à programmer dans un paradigme autre que l'objet

Travailler en équipe

- Travailler en équipe devient plus facile à mettre en place dans les projets Objet
 - On sépare en général la branche technique de la branche fonctionnelle
 - Développement en Y
 - Les domaines fonctionnels sont mieux séparés en composants et en bibliothèques de composants
 - Il faut s'appuyer sur les interfaces (qui forment des APIs) pour un bon travail en équipe
- Des outils comme SVN, GIT, Mercurial,... permettent de faciliter le travail collaboratif

Les outils des projets Objet

- Outils de modélisation :
 - Rational XDE
 - Together
 - Poseidon for UML
 - EclipseUML
- Outils de développement :
 - Eclipse / WSAD
 - IntelliJ IDEA
 - NetBeans
 - ...

Les outils des projets Objet

- Outils de mapping O/R :
 - TopLink
 - CocoBase
 - JDO
 - Hibernate
 - ...
- Autres outils :
 - Junit : permet de faire des tests unitaires en Java
 - Ant : permet de faire de l'automatisation des builds
 - Maven : permet de gérer quasiment toute votre chaîne de développement

L'open-source

- L'open-source est très présente dans le monde du développement objet, notamment Java
 - => Disponibilité de beaucoup d'outils et de frameworks pour développer ses projets Objet
- Ex. :
 - Projets Apache : <http://www.apache.org>
 - Offre de nombreux frameworks, des serveurs, ...etc.
 - Ex. : Maven, Ant, ...
 - Projets Eclipse : <http://www.eclipse.org>
 - Autres projets
 - Ex. : Junit

Sommaire

1. Evolution des langages vers l'objet
2. Caractéristiques de l'approche objet

Annexe: Introduction à UML

1. Importance de la modélisation
2. Exemple de diagramme statique
3. Exemple de diagramme dynamique

Les méthodes objets

- Années 90, plus de 30 méthodes recensées
 - Booch, Classe-Relation, Fusion, HOOD, OMT, OOA/OOD, OOM, OOSE...

OMT Booch Method OOA/OOD Classe-Relation
 T HOOD OOSE Fusion OOM etc.

- Un foisonnement de concepts et de notations
- Une multitude d'acteurs en concurrence
- Une joyeuse pagaille qui ne fait plus progresser l'objet

Unification

- En 1995, trois acteurs majeurs du monde objet décident de lancer un projet ambitieux d'unification de leurs trois méthodes
- En 1997 ce projet donne naissance à UML



Pourquoi modéliser

- Moyen de clarification des idées, relations, décisions, besoins
- Support à la communication**
- Les éléments modélisés sont des objets du monde réel, des éléments logiciels, etc.
- La modélisation s'attache à représenter ces éléments sous l'angle nécessaire à leur compréhension dans le contexte du système à développer



Qu'est-ce qu'un diagramme ?

- Un diagramme est une représentation graphique, sous un certain point de vue d'un ensemble d'éléments faisant partie du modèle, y compris les relations entre eux (qui sont elles mêmes des éléments)
- Un même élément peut être présent dans plusieurs diagrammes avec des niveaux différents de représentation



Qu'est-ce qu'un diagramme ?

- Les types de diagrammes peuvent être distingués en fonction du point de vue particulier qu'ils cherchent à illustrer sur les éléments qu'il présentent
- L'utilisation d'un outil servant de référentiel aux éléments modélisés permet d'assurer la cohérence de ces éléments à travers leur représentation graphique

Introduction à UML Création d'UML

- Quatre objectifs pour «The Unified Method»
 - **Représenter des systèmes entiers** par des concepts objets.
 - Etablir un couplage entre les concepts et les artefacts exécutables.
 - Permettre la **modélisation à plusieurs échelles** d'abstraction.
 - Concevoir un **langage utilisable à la fois par les humains et les programmes**.

- En 1997 normalisation par l'OMG:
 - **Pas de « Unified Method » !**
 - Mais **un langage de modélisation** « The **U**nified **M**odeling **L**anguage for Object-Oriented Development».

UML est une notation (1/2)

- UML est un langage de modélisation objet
 - Plusieurs diagrammes standardisés (facettes complémentaires d'un système)

- UML est un langage de communication
 - Utilisation d'un même formalisme par tous les intervenants
 - Permet de lever les ambiguïtés du langage naturel

- UML est un langage simple de haut niveau
 - Facile à appréhender car visuel
 - Indépendant de tout langage de programmation

UML est une notation (2/2)

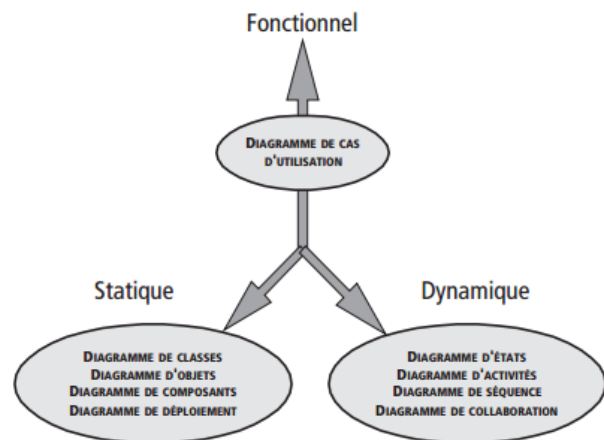
- UML est un langage semi-formel
 - Basé sur un méta-modèle décrit en UML
 - Les concepts manipulés et leur sémantique sont clairement établis par le méta-modèle
 - Les différents diagrammes sont cohérents entre-eux
 - UML est largement outillé
- UML est un langage ouvert
 - Possibilité d'ajouter des notes (texte)
 - Utilisation de stéréotypes permettant d'étendre la notation, améliorant ainsi la sémantique des éléments modélisés

UML n'est pas une méthode

- Méthode
 - Une démarche + des outils
 - Un processus + un langage de modélisation
- UML n'est pas une méthode, ce n'est qu'une notation
 - UML = Langage de modélisation
 - Attente de la part des utilisateurs d'une normalisation du formalisme
 - Définir une méthode de développement logiciel universel est illusoire
- UML est indépendant de toute démarche
 - Ce qui en fait un langage universel
 - Mais favorise la mise en œuvre d'un processus itératif et incrémental, fondé sur les cas d'utilisation et centré sur l'architecture

Plusieurs axes de modélisation

- **Axe structurel**
 - Modélisation statique du système
 - Quels objets manipule le système ?
 - Détermination du **QUOI**
- **Axe comportemental / dynamique**
 - Modélisation dynamique du système
 - Sous quelles conditions agit le système ?
 - Détermination du **QUAND**
- **Axe fonctionnel**
 - Modélisation des traitements offerts par le système
 - Que fait le système ?
 - Détermination du **COMMENT**



De façon générale rappelez vous que ...

- En UML :
 - (Presque) Tout est optionnel
 - ce qui est sur un diagramme existe, ce qui n'y est pas peut exister et n'être pas représenté
 - Un modèle n'est jamais complet
 - UML est conçu pour être étendu pour répondre aux besoins spécifiques
 - Personnalisation et extension à travers un *profil*
 - UML est ouvert à l'interprétation (dans une certaine mesure)
 - L'agrégation peut se traduire par un pointeur C++ et une composition par une référence C++, mais ça n'est pas dans la norme

Sommaire

1. Evolution des langages vers l'objet
2. Caractéristiques de l'approche objet

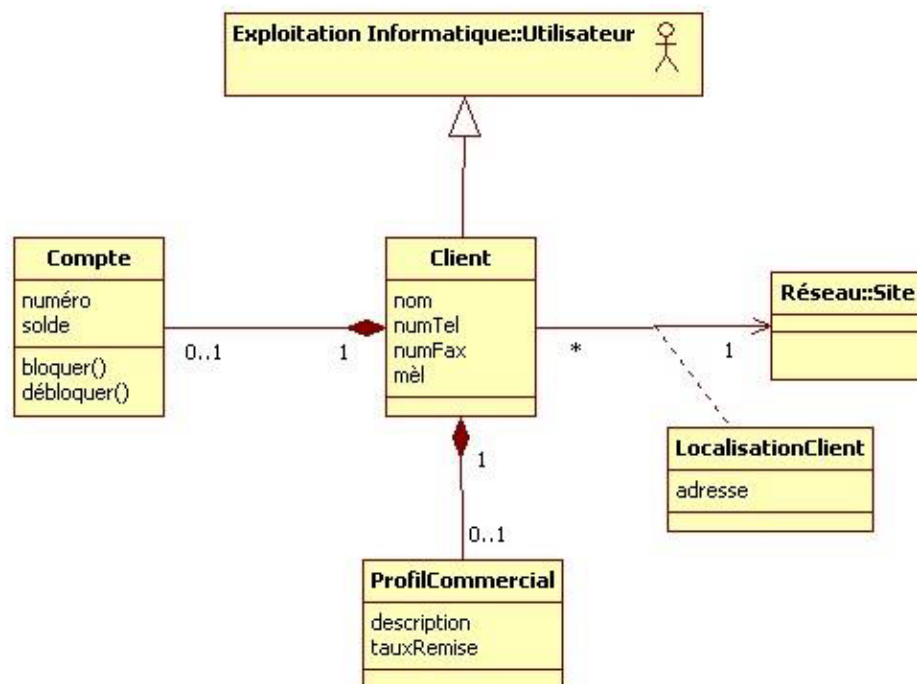
Annexe: Introduction à UML

1. Importance de la modélisation
2. Exemple de diagramme statique
3. Exemple de diagramme dynamique

Diagrammes de classes

- Le DC présente les entités et les relations (statiques) entre elles
- Il permet de représenter l'ensemble des informations formalisées ayant fait l'objet d'une définition sur le fond et sur la forme
 - Avec les utilisateurs quant aux entités du domaine
 - Avec les développeurs quant aux entités techniques

Diagrammes de classes



Sommaire

1. Evolution des langages vers l'objet
2. Caractéristiques de l'approche objet

Annexe: Introduction à UML

1. Importance de la modélisation
2. Exemple de diagramme statique
3. Exemple de diagrammes dynamiques

Diagramme d'activités

- Le DA identifie les flux entre 2 activités successives
- Il permet de représenter la dynamique de l'élément (processus, acteur, entité) auquel il est attaché

Diagramme d'activités

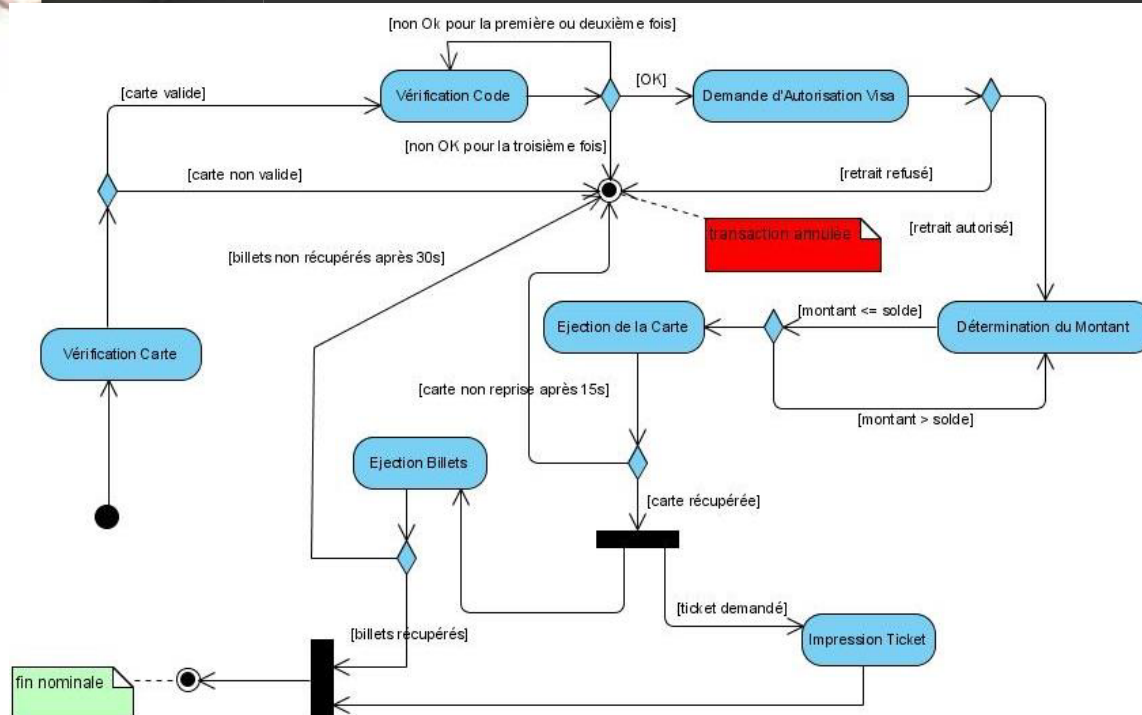


Diagramme de séquences

- Le DS met en évidence le type et l'ordre des messages échangés entre composants

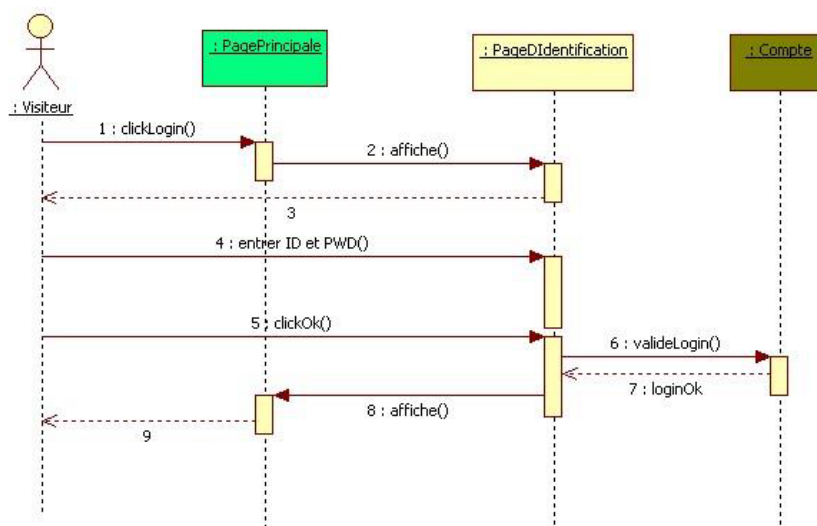


Diagramme de machine finie à états

- Le DE met en évidence les transitions d'états internes à un élément (classe, ensemble de composants, système tout entier) auquel il est associé

Diagramme de machine finie à états

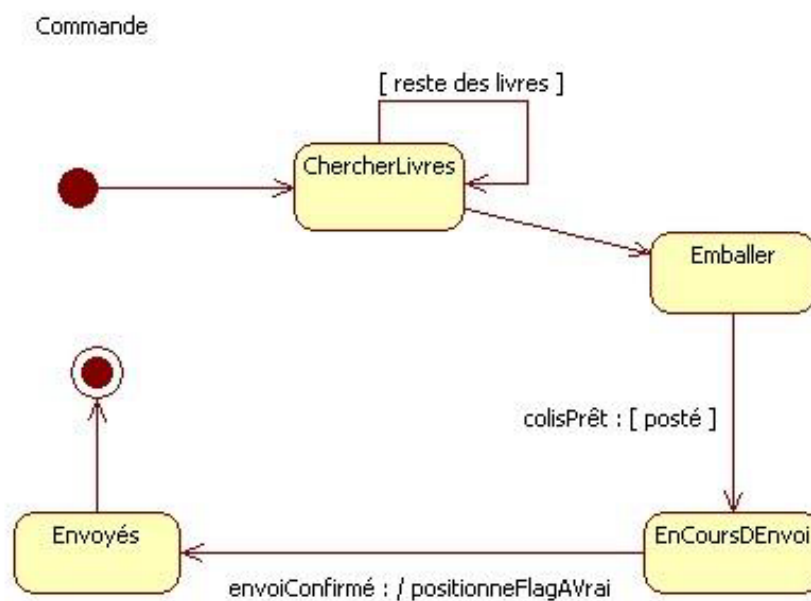


Diagramme de cas d'utilisation

- Le DCU exprime, conjointement à une description textuelle, les exigences (besoins) d'un système

Diagramme de cas d'utilisation

