

Algorithmique et Programmation

Initiation à la modélisation objet

.....

Support de Cours
Dpt Informatique Polytech'Nantes
Fabien Picarougne

La conception objet : Plan

.....

- Introduction / historique
- Concept objet
 - Modularité / Réutilisabilité
 - Vue d'ensemble
- Principe de la modélisation UML
 - Présentation générale
 - Démarche de modélisation
 - Diagramme de classe
 - Objet et classes
 - Encapsulation
 - Associations / agrégations
 - Héritage
 - Polymorphisme
 - Diagramme d'états-transitions
 - Diagramme de communication

Introduction

.....

- Thèse de Church-Turing :
 - « *tout langage de programmation non trivial équivaut à une machine de Turing* »
 - tout programme écrit dans un langage pourrait être également écrit dans n'importe quel autre langage
 - différence entre les langages ne réside donc pas dans ce qu'ils **permettent de programmer**, mais dans ce qu'il est **facile, commode de programmer** avec chaque langage
- But de la modélisation
 - Faciliter la programmation (plus de personnes ont accès à la programmation)
 - Fiabiliser le développement
 - Rendement accru

Introduction

.....

- Évolution des principes de programmation
 - Initialement langages impératifs
 - Assembleur / Fortran / Basic / Lisp / ...
 - Très proche du langage machine
 - jmp -> goto ...
 - Très difficile de réutiliser du code existant
 - ❖ Utilisation d'organigrammes
 - Langages procéduraux
 - Pascal / C / ...
 - Programme = structure de données + algorithmes
 - Typage fort et types structurés
 - Code structuré (*Goto considered harmful* Dijkstra)
 - Meilleure structuration mais programmes encore trop proche du contexte d'utilisation
 - ❖ Définition des structures de données, représentation des pointeurs

Introduction

.....

- Évolution naturelle : langages orientés objets
 - Delphi / C++ / Java / ...
 - Langages fonctionnel imposent la séparation **données/algorithmes**
 - Langages orientés objets (LOO) imposent la séparation **fonctionnalité/implémentation**
 - **Encapsulation** : on ne voit d'un objet que ce qu'on a besoin de voir
 - **Héritage** : permet l'abstraction et la réutilisation
 - **Polymorphisme** : facilite la lisibilité/simplicité du code
- ❖ Utilisation de méthodologies orientée objet (OOSE, OMT, UML, ...)

Introduction

.....

- Méthodologies de programmation
 - On parle de **conception** orientée objet plutôt que de **programmation** orientée objet
 - Étape de modélisation **indépendante** du langage utilisé
 - Nécessité de réfléchir à la structuration du modèle/programme avant de le coder/implémenter
 - Au cours du temps plusieurs méthodologies se succèdent
 - Énormément de littérature sur le sujet à partir de 90 (plus de théorisation que pour les modèles fonctionnels)
 - Question philosophiques/métaphysiques sur la nature d'un objet
 - « *Thus, an object-oriented analysis can be regarded as a form of syllogism moving from the **Particular** (classes) through the **Individual** (instances) to the **Universal** (control)* »

Introduction

.....

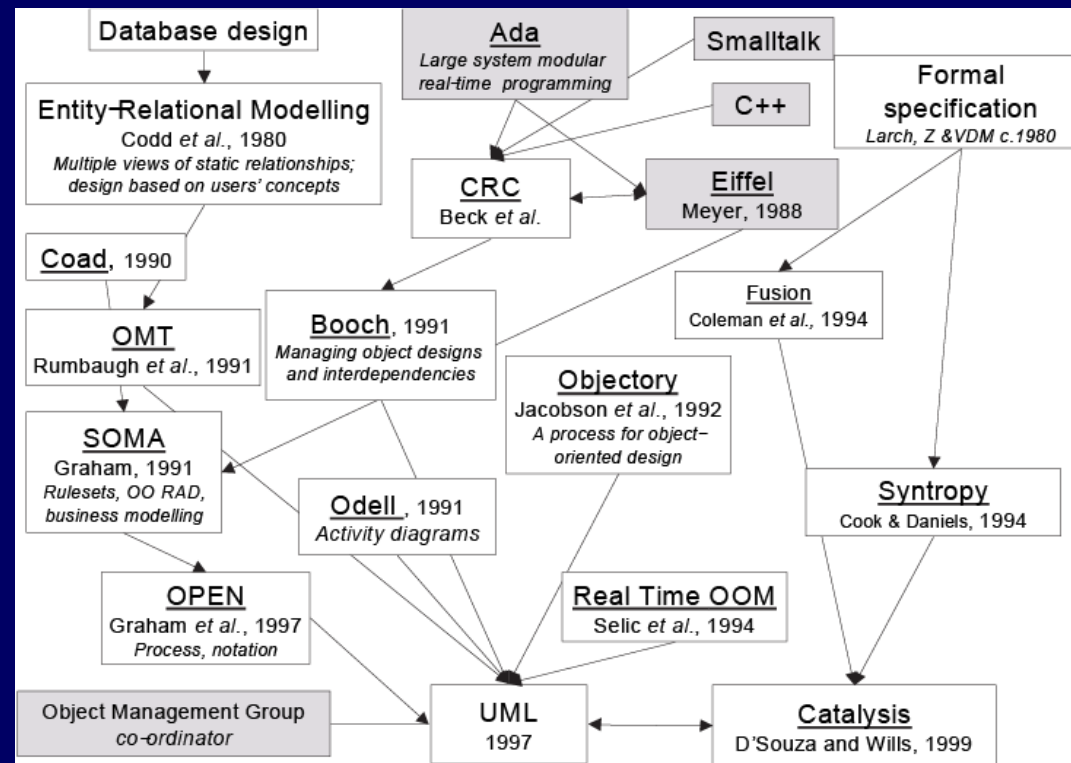
- Historique

- Années 60, **Simula-67** (travaux sur les langages de simulation) prémisses de la programmation objet
- Fin des années 70, années 80, OO = recherche en IA
- Fin des années 80, **Smalltalk** (inspiré en partie par Simula) base des concepts objets (objet, messages, encapsulation, polymorphisme, héritage, redéfinition)
- Fin des années 80, **Booch** écrit *Object Oriented Design* qui parle de **ADA**
- Début des années 90, explosion des modélisations et langages (début de **C++**, mais beaucoup de bugs dans la spécification)
 - 1991 **OMT**, J. Rumbaugh
 - 1992 **OOSE**, I. Jacobson
 - 1994 **BOOCH**, G. Booch
 - 1994 : au moins 72 méthodes ou fragment de méthode OO, situation intenable
- Création de l'**UML** (*Unified Modeling Language*) par l'Object Management Group (**OMG**) issue du rapprochement de plusieurs méthodes

Introduction

- 1995 : Méthode unifiée 0.8 (intégrant les méthodes Booch'93 et OMT)
- 1995 : UML 0.9 (intégrant la méthode OOSE)
- 1996 : UML 1.0 (proposée à l'OMG)
- 1997 : UML 1.1 (standardisée par l'OMG)
- 1998 : UML 1.2
- 1999 : UML 1.3
- 2000 : UML 1.4
- 2001 : UML 1.5
- 2003 : UML 2.0
- 2007 : UML 2.1.2
- 2009 : UML 2.2
- 2010 : UML 2.3
- 2011 : UML 2.4.1
- 2015 : UML 2.5

- Schéma proposé par Florent de Dinechin



La conception objet : Plan

.....

- Introduction / historique
- **Concept objet**
 - **Modularité / Réutilisabilité**
 - **Vue d'ensemble**
- Principe de la modélisation UML
 - Présentation générale
 - Démarche de modélisation
 - Diagramme de classe
 - Objet et classes
 - Encapsulation
 - Associations / agrégations
 - Héritage
 - Polymorphisme
 - Diagramme d'états-transitions
 - Diagramme de communication

Concept Objet

.....

- La conception objet est issue des réflexions effectuées autour de la qualité
 - la *correction* ou la *validité* : c'est-à-dire le fait qu'un logiciel effectue exactement les tâches pour lesquelles il a été conçu
 - l'*extensibilité* : c'est-à-dire la capacité à intégrer facilement de nouvelles spécifications
 - la *réutilisabilité* : les logiciels écrits doivent pouvoir être réutilisables, complètement ou en partie
 - Ceci impose lors de la conception une attention particulière à l'organisation du logiciel et à la définition de ses composantes
 - la *robustesse* : c'est-à-dire l'aptitude d'un logiciel à fonctionner même dans des conditions anormales. Le logiciel est capable de détecter qu'il se trouve dans une situation anormale.

Concept Objet

.....

- Modularité
 - Difficile de respecter ces critères lorsque l'architecture d'un logiciel est obscure, monolithique
 - Mise en place d'une **architecture flexible** pour palier à des changements de spécification : les **modules**
 - Concentration des **connaissances** liées à une entité logique **à l'intérieur d'un module**
 - Seul le module exploite ces connaissances
 - Confine la maintenance au module concerné
 - 2 méthodes de conception
 - **Descendante**
 - Procède par **décomposition de problème** en un certain nombre de sous-problèmes, chacun de complexité moindre
 - Récursivement jusqu'à ce que chacun des sous-problèmes soit trivial
 - **Ascendante**
 - **Composition** de briques logicielles simples
 - Cas des bibliothèques de sous-programmes, librairies

Concept Objet

.....

- Critères de qualité de modularité
 - *Compréhensibilité modulaire*
 - Les modules doivent être clairs et organisés de manière compréhensible dans le système
 - Les modules communiquent avec peu de modules
 - L'enchaînement des différents modules doit être logique
 - Ex : on ne doit pas avoir à utiliser plusieurs fois de suite un module pour produire une action atomique
 - *Continuité modulaire*
 - Une petite modification des spécifications n'entraîne qu'un nombre limité de modifications au sein d'un petit nombre de modules, sans remettre en cause les relations qui les lient
 - *Protection modulaire*
 - Toute action lancée au niveau d'un module doit être confinée à ce module, et éventuellement à un nombre restreint de modules
 - Ce critère ne permet pas de corriger les erreurs introduites, mais de confiner autant que possible les erreurs dans les modules où elles sont apparues.

Concept Objet

.....

- Principes de conception modulaire
 - *Interface limitée*
 - Se restreindre à un nombre limité d'actions bien définies
 - *Communications limitées*
 - Conséquence du principe de modularité, d'autant mieux respectée que les modules jouent leur rôle propre.
 - Si les échanges sont trop importants, la notion même de module devient floue, limitant l'intérêt de cette technique.
 - *Interface explicites*
 - *Masquage de l'information*
 - toutes les informations contenues dans un module doivent être **privées** au module, à l'exception de celles explicitement définies **publiques**
 - Les communications autorisées sont ainsi celles explicitement définies dans l'*interface* du module
 - Le module spécifié **ne doit pas s'adapter** au langage de programmation
 - Au contraire le langage de programmation doit proposer une structure permettant d'implanter le module tel qu'il a été spécifié
 - Par exemple, si le langage de programmation ne permet pas d'effectuer le masquage de l'information (comme le langage C), il n'est pas adéquat
- **Donc pas de variables globales**
 - Peuvent être utilisées et modifiées par n'importe quelle composante d'un programme

Concept Objet

.....

- Réutilisabilité
 - Présente depuis longtemps en informatique
 - Ex : bibliothèques de fonctions / librairies
 - Mais **limitée**
 - Pas capables de s'adapter aux **changements de types** ou **d'implantation**
 - Solution : fournir une multitude de fonctions (ex : fonction `cos` pour `int`, `float`, `double`)
 - Conception OO : formalise un peu plus cette notion et propose de nouvelles techniques

Concept Objet

.....

- Conception de modules réutilisables
 - Doit pouvoir **manipuler plusieurs types** différents
 - Doit **s'adapter** aux différentes structures de données manipulées dotées de méthodes spécifiques.
 - Ex : pouvoir rechercher de la même manière dans un tableau, une liste, un fichier, ...
 - Doit offrir des opérations aux clients qui l'utilisent **sans que ceux-ci connaissent l'implémentation** de l'opération
 - Conséquence directe du masquage de l'information. Protection vis-à-vis de changement de spécifications
 - Les **opérations communes** à des modules doivent pouvoir être **factorisées** dans un même module
 - Ex : accès aux listes, tableaux dotés d'opérations de même nom permettant l'accès, la manipulation des éléments
 - Permettre entre autres de définir des algorithmes communs (recherche, ...)

Concept Objet

.....

- Conception de modules réutilisables : nouvelles techniques
 - Notion de **paquetage**
 - **Regroupement**, au sein d'un même module, d'une **structure de données** et des **opérations** qui lui sont propres
 - Notion de **surcharge**
 - Des **opérations** appartenant à des modules différents peuvent être **associées au même nom**
 - Ex : permet de définir une fonction *insérer* dans chaque module de stockage, permettant d'écrire de manière uniforme : `insérer(elt, container)` quelque soit le type de container (liste, tableau, fichier...)
 - Notion de **généricité**
 - Permet de définir des **modules paramétrés par le type** qu'ils manipulent
 - Notion très intéressante, car elle va permettre la définition de méthodes (façon de travailler) plus que de fonctions (plus formelles)

Concept Objet

.....

- Vue d'ensemble des concepts / vocabulaire
 - **Objet**
 - Une **identité** : distinction entre deux objets
 - Propriétés (**attributs**) : définies par la classe d'appartenance de l'objet
 - Un **état** : valeurs de ses attributs à un instant donné
 - Comportement (**méthodes**) : l'ensemble des opérations qu'il peut effectuer en réponse à des messages envoyés par d'autres objets
 - **Classe**
 - **Abstraction** d'un ensemble d'objets qui possède une structure et un comportement identique
 - **Liste des attributs** (~ champs d'une structure de donnée des langages classiques)
 - **Liste des opérations** (~ fonctions/procédures qui dépendent sémantiquement de la classe)
 - Un objet est une **instance** d'une et une seule classe
 - Une **classe abstraite** est une classe qui n'a pas d'instance

Concept Objet

.....

- Encapsulation Traitement / Données

- Regroupement dans une même classe de la description de la structure des attributs et des opérateurs
- Autonomie et indépendance de chaque classe : **réutilisabilité**

- Association

- Relation entre plusieurs classes
- *Abstraction* des liens entre les objets dans le monde réel
- Définie totalement par :
 - Le **rôle** des objets intervenant
 - La multiplicité des objets (**cardinalité**)

- Agrégation

- Forme particulière d'association
- Exprime le fait qu'une classe est composée d'une ou plusieurs sous-classe
 - ex : organigramme d'une entreprise

Concept Objet

- Généralisation

- **Factorisation** dans une classe (appelée super-classe) des attributs et des méthodes
- Permet de réaliser une hiérarchie de classes (au sens de l'inclusion)

- Spécialisation

- Inverse de la généralisation
- **Dérive** les sous-classes (spécialisées) d'une classe initiale
- Spécialisation se fait par **héritage** de la super-classe

Concept Objet

.....

- Polymorphisme

- Capacité donnée à une même opération (ex : affichage) de **s'exécuter différemment suivant le contexte** (ex : image ou tableau)
 - Couplé à l'héritage : une opération définie dans une super-classe peut s'exécuter différemment selon la sous-classe

- Persistance

- Capacité donnée à un objet de **continuer à exister après la fin du programme**
- Par défaut aucun objet n'est persistant

La conception objet : Plan

.....

- Introduction / historique
- Concept objet
 - Modularité / Réutilisabilité
 - Vue d'ensemble
- Principe de la modélisation UML
 - Présentation générale
 - Démarche de modélisation
 - Diagramme de classe
 - Objet et classes
 - Encapsulation
 - Associations / agrégations
 - Héritage
 - Polymorphisme
 - Diagramme d'états-transitions
 - Diagramme de communication

Principe de la modélisation UML

.....

- UML (*Unified Modeling Language*)
 - Langage **graphique** de **modélisation** des données et des traitements
 - **Standard** défini par l'OMG (*Object Management Group*)
 - Non défini comme une méthode (utilisation laissée à l'appréciation de chacun)
 - UML se décompose en plusieurs sous-ensembles
 - Les **vues** : Les vues sont les observables du système
 - Décrivent le système d'un point de vue donné, qui peut être organisationnel, dynamique, temporel, logique, ...
 - En combinant toutes les vues il est possible de définir (ou retrouver) le système complet.
 - Les **diagrammes** : Les diagrammes sont des éléments graphiques
 - Décrivent le contenu des vues, qui sont des notions abstraites
 - Les **modèles d'élément** : Les modèles d'élément sont les briques des diagrammes UML
 - Ces modèles sont utilisés dans plusieurs types de diagramme
 - Ex : cas d'utilisation (*CU*), classe, association, etc.
 - Comporte 13 types de diagrammes (9 en UML 1.3)

Principe de la modélisation UML

.....

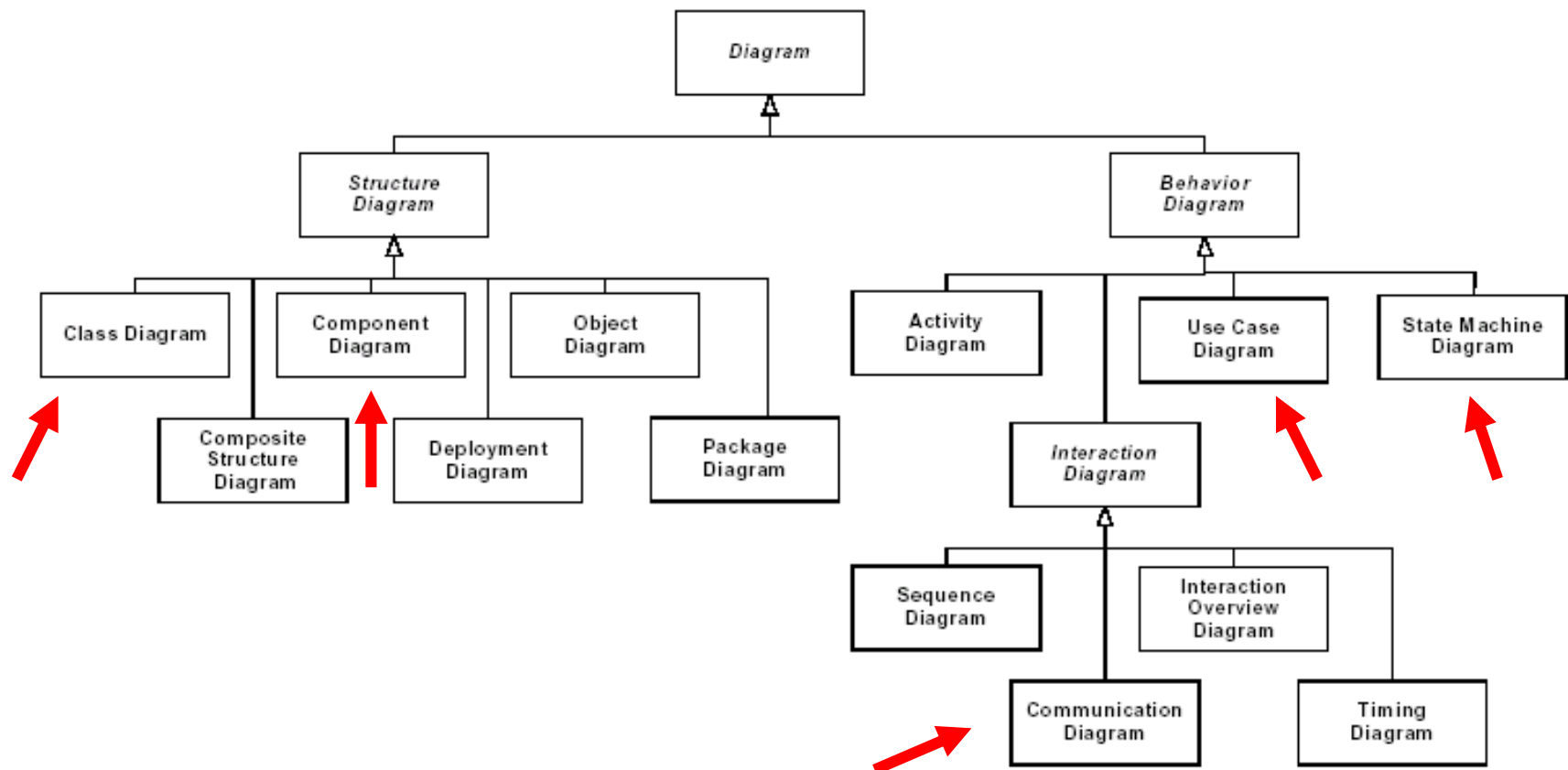
- Les 13 diagrammes UML sont dépendants hiérarchiquement et se complètent
 - Diagrammes Structurels ou Diagrammes statiques
 - **Diagramme de classes** : représente les classes intervenant dans le système ainsi que leurs relations (association, agrégation, généralisation, spécialisation)
 - **Diagramme d'objets** : sert à représenter les instances de classes (objets) utilisées dans le système
 - **Diagramme de composants** : permet de montrer les composants du système d'un point de vue physique (fichiers, bibliothèques, bases de données...)
 - **Diagramme de déploiement** : sert à représenter les éléments matériels (ordinateurs, périphériques, réseaux, ...) et la manière dont les composants du système sont répartis sur ces éléments et interagissent avec eux
 - **Diagramme des paquetages** : un paquetage étant un conteneur logique permettant de regrouper et d'organiser les éléments dans le modèle UML, le Diagramme de paquetage sert à représenter les dépendances entre paquetages
 - **Diagramme de structure composite** : permet de décrire sous forme de boîte blanche les relations entre composants d'une classe

Principe de la modélisation UML

.....



- Diagrammes Comportementaux
 - **Diagramme des cas d'utilisation (use-cases)** : identifie les possibilités d'interaction entre le système et les acteurs (fonctionnalités que doit fournir le système)
 - **Diagramme états-transitions** : décrit sous forme de machine à états finis le comportement du système ou de ses composants
 - **Diagramme d'activité** : décrit sous forme de flux ou d'enchaînement d'activités le comportement du système ou de ses composants
- Diagramme d'interactions ou Diagrammes dynamiques
 - **Diagramme de séquence** : représentation séquentielle du déroulement des traitements et des interactions entre les éléments du système et/ou de ses acteurs.
 - **Diagramme de communication** : représentation simplifiée d'un diagramme de séquence se concentrant sur les échanges de messages entre les objets.
 - **Diagramme global d'interaction** : décrit les enchaînements possibles entre les scénarios préalablement identifiés sous forme de diagrammes de séquences
 - **Diagramme de temps** : décrire les variations d'une donnée au cours du temps.

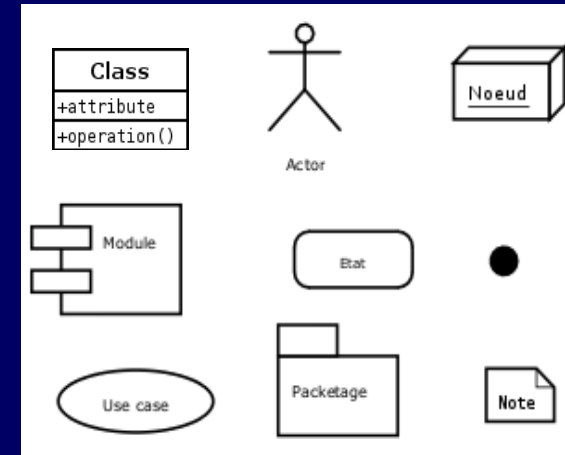
Principe de la modélisation UML



Principe de la modélisation UML

- Les modèles d'éléments de type communs

- État initial (Initial state) 
- État terminal (Final state) 
- Interface (Interface)



- Les modèles d'éléments de type relation

- Dépendance 
- Généralisation
- Association
- Agrégation
- Composition

La conception objet : Plan

.....

- Introduction / historique
- Concept objet
 - Modularité / Réutilisabilité
 - Vue d'ensemble
- Principe de la modélisation UML
 - Présentation générale
 - **Démarche de modélisation**
 - Diagramme de classe
 - Objet et classes
 - Encapsulation
 - Associations / agrégations
 - Héritage
 - Polymorphisme
 - Diagramme d'états-transitions
 - Diagramme de communication

Principe de la modélisation UML

.....

- Démarche en 3 phases (issue de la méthode OMT)
 - Analyse
 - À partir de l'expression des besoins
 - Élaboration des modèles (diagrammes)
 - Conception du système
 - À partir des modèles construits
 - Affinés et complétés afin de décrire l'architecture de l'ensemble du système
 - Découpage en sous-système
 - Conception des objets (implémentation)
- Cycle itératif et incrémental
 - Chaque cycle : analyse, conception, implémentation, appréciation

Principe de la modélisation UML

.....

- 1 - Phase d'analyse
 - But : comprendre et modéliser l'application
 - **Donnée initiale** : définition du problème incluant
 - Le problème à résoudre
 - Une vue d'ensemble des concepts du système
 - **Données supplémentaires**
 - Discussion avec le consommateur
 - Discussion avec les experts
 - **Résultats** : modèle formel qui saisit
 - Les objets et leur relation
 - Le flux dynamique de contrôle
 - Les transformation et contraintes sur les données

Principe de la modélisation UML

.....

- Analyse détaillée
 - Écrire ou obtenir une description initiale du problème
 - Construire le modèle objet
 - Identifier les **classes d'objets**
 - Commencer à remplir un **dictionnaire de données** contenant la description des **classes** et **attributs/méthodes** et des **associations**
 - Ajouter les associations entre les classes
 - Ajouter les attributs/méthodes des objets et des liens
 - Organiser les classes d'objet en les faisant **hériter**

Principe de la modélisation UML

.....

- Développer un modèle dynamique
 - Préparer des scénarios de séquences d'interactions typiques
 - Identifier les **événements** entre objets
 - Préparer un **diagramme de cas d'utilisation** pour chaque scénario
 - Développer un **diagramme d'états** pour chaque classe ayant un comportement dynamique important
 - Vérifier la cohérence et la complétude des événements partagés parmi tous les diagrammes d'états

Principe de la modélisation UML

.....

- Construire un modèle fonctionnel
 - Identifier les **valeurs d'entrées et de sortie**
 - Utiliser les **diagrammes de séquence** pour mettre en évidence des dépendances fonctionnelles
 - Décrire ce que fait chaque fonction
 - Identifier les contraintes **internes** et **externes** aux objets

Principe de la modélisation UML

.....

- Vérifier, itérer et affiner les 3 modèles
 - Ajouter au modèle objet les opérations cités mis à jour par le modèle fonctionnel
 - Ne pas prévoir toutes les opérations pendant l'analyse (trop lourd)
 - Vérifier que les classes, associations, attributs et opérations sont cohérents et complets à chaque niveau
 - Comparer les 3 modèles à la définition du problème et aux connaissances recueillies
 - Développer des scénarios plus élaborée (gestion des erreurs, ...)

Principe de la modélisation UML

.....

- 2 - Conception système
 - But : déterminer **l'architecture d'ensemble** du système (*les grands axes*)
 - Guide : le modèle objet
 - Organisation du système en **sous-systèmes**
 - **Regroupement** des objets et des tâches
 - Décision globale prise sur
 - Communication inter-processus
 - Stockage mémoire
 - Implémentation du modèle dynamique

Principe de la modélisation UML

.....

- Conception système détaillée
 - Organiser le système en sous-systèmes
 - Identifier les **concurrences** inhérentes
 - Allouer les sous-systèmes à des ressources
 - Choisir la stratégie de base pour implémenter le stockage des données en terme de
 - Structure de données
 - Fichiers
 - BD
 - Identifier les ressources globales et déterminer les **mécanismes de contrôle d'accès**
 - Examiner les conditions critiques
 - Établir **compromis et priorités**
 - Document de conception système = *structure de l'architecture de base du système et décisions stratégiques de haut niveau*

Principe de la modélisation UML

.....

- 3 - Conception objet (implémentation)
 - Les modèles d'analyse sont élaborés et raffinés pour produire une conception concrète
 - L'accent n'est plus mis sur les concepts de l'application, mais sur les concepts informatiques
 - **Choix des algorithmes** de bases pour les fonctions importantes
 - **Optimisation** de la structure des objets pour obtenir une implémentation efficace
 - Organisation des sous-systèmes en **modules**

Principe de la modélisation UML

.....

- Conception objet détaillée
 - Obtenir les opérations pour le modèle objet à partir des autres modèles
 - Trouver une opération pour chaque traitement du **modèle fonctionnel**
 - Définir une opération pour chaque traitement du **modèle dynamique**
 - Concevoir des algorithmes pour implémenter les **opérations**
 - Choisir les algorithmes qui minimisent le coût d'implémentation
 - Sélectionner les structures de données propres aux algorithmes
 - Définir de **nouvelles classes internes** et de **nouvelles opérations** si nécessaire

Principe de la modélisation UML

.....

- Optimiser les chemins d'accès aux données
 - Ajouter des associations redondantes pour minimiser les coûts d'accès
 - Réorganiser les calculs pour plus d'efficacité
- Ajuster la structure de classe pour augmenter l'héritage
 - Réorganiser les classes et les opérations pour accroître l'héritage
 - Abstraire le comportement commun à un sous-groupe de classes
- Concevoir l'implémentation des associations
 - Implémenter chaque association comme un objet distinct ou en ajoutant des attributs à une des classes associées
 - Déterminer la représentation exacte des attributs de l'objet
 - Empaqueter les classes et les associations au sein des modules physiques

Principe de la modélisation UML

.....

- Impact d'une approche OO
 - L'accent du développement est mis sur l'analyse
 - Permet le plus souvent une implémentation rapide
 - Les évolutions futures sont facilitées
 - L'accent est mis sur les structures de données plus que sur les fonctions
 - Concept unique : le concept objet qui regroupe données et fonctions
 - Les autres concepts sont organisés autour des objets
 - Organisation autour des données plutôt qu'autour des processus donne une stabilité grâce à l'encapsulation
 - Le processus de développement est sans rupture
 - On retravaille les objets sans repartir de zéro
 - On augment le niveau de détail

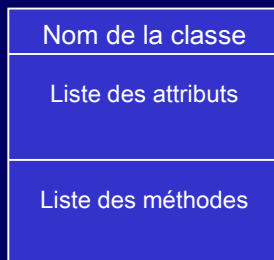
La conception objet : Plan

.....

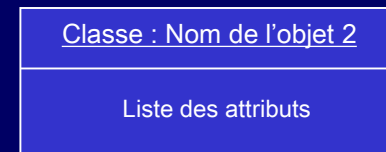
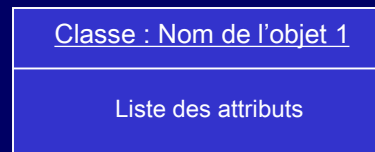
- Introduction / historique
- Concept objet
 - Modularité / Réutilisabilité
 - Vue d'ensemble
- Principe de la modélisation UML
 - Présentation générale
 - Démarche de modélisation
 - **Diagramme de classe**
 - **Objet et classes**
 - **Encapsulation**
 - **Associations / agrégations**
 - **Héritage**
 - **Polymorphisme**
 - Diagramme d'états-transitions
 - Diagramme de communication

Principe de la modélisation UML

- Diagramme de classe (modèle objet)
 - Définitions
 - Classe = groupement d'objets ayant les mêmes propriétés (**attributs**), un même comportement (**opérations/méthodes**) et une **sémantique commune** (domaine et définition)
 - Objet = une **instance** d'une classe
 - Une classe représente l'abstraction de ses objets



Classe



Objet

Principe de la modélisation UML

- Attributs de classes
 - Définition
 - Un attribut est une **propriété élémentaire** d'une classe
 - Caractéristiques
 - Chaque attribut est désigné par un nom unique dans sa classe
 - À chaque instance de classe correspond une valeur unique de chaque attribut
 - L'identifiant d'une classe ne doit pas être un attribut artificiel ajouté par le concepteur

Nom de la classe
Nom attribut : (type) = (valeur initiale)

Voiture
Couleur : (texte) = noire
Immatriculation : (texte) = 1 A 44

<u>Voiture : Clio</u>
Couleur = grise
Immatriculation = 456 ABC 44

Principe de la modélisation UML

- Méthodes (opérations) de classes
 - Définition
 - Une méthode est une **fonction** qui est applicable aux objets d'une classe
 - Elle permet de décrire le comportement d'un objet
 - Caractéristiques
 - L'implémentation d'une opération est une **méthode**
 - Le **polymorphisme** permet d'utiliser une même opération pour des classes différentes

Nom de la classe
Nom attribut : (type) = (valeur initiale)
Nom d'opération (paramètres) : résultat

Objet_Géométrique
Couleur : (texte) = rouge Position : (vecteur) = {0,0,0}
Déplacer (v : vecteur) Tourner (alpha : angle)

Principe de la modélisation UML

.....

- Encapsulation
 - Pour chaque objet, il y a
 - Ceux qui le **programment**
 - Ceux qui **l'utilisent** (par ex. dans d'autres programmes)
 - Pour chaque objet, on définit
 - Des attributs et méthodes internes, utilisées pour le **programmer**
 - Des attributs et des méthodes externes, utilisées pour **l'interfacer**

Principe de la modélisation UML

.....

- Un objet a une **interface** par laquelle on le manipule, et que tout le monde connaît
 - Cette interface fait partie de sa **spécification**
 - Elle est tout ce dont ont besoin les **utilisateurs** de l'objet
- Les utilisateurs **n'ont pas besoin** de connaître les détails de l'implémentation d'un objet
 - Ils ont une **vue** plus simple, plus lisible
 - Leur code ne risque pas **d'interférer** avec le code interne de l'objet
 - On peut **changer l'implémentation** d'un objet de manière transparente pour ses utilisateurs

Principe de la modélisation UML

- En pratique
 - Les attributs ou méthodes sont
 - Soit **publics (+)**, visibles par tout le monde et destinés à être utilisés par tous les utilisateurs de l'objet
 - Soit **privés (-)**, relatif au fonctionnement interne de l'objet, que les utilisateurs n'ont pas besoin de connaître
 - Il y a en général au moins un niveau d'accès supplémentaire appelé **protégé (#)**
 - Permet l'accès à du code *ami* (typiquement les autres objets d'une classe issue de l'héritage)

Objet_Géométrique
+ Couleur : (texte) = rouge
- Position : (vecteur) = {0,0,0}
Déplacer (v : vecteur)
+ Tourner (alpha : angle)

- *On peut implémenter cette philosophie dans n'importe quel langage mais les langages OO apportent une **garantie mécanique** qu'elle est respectée*

Principe de la modélisation UML

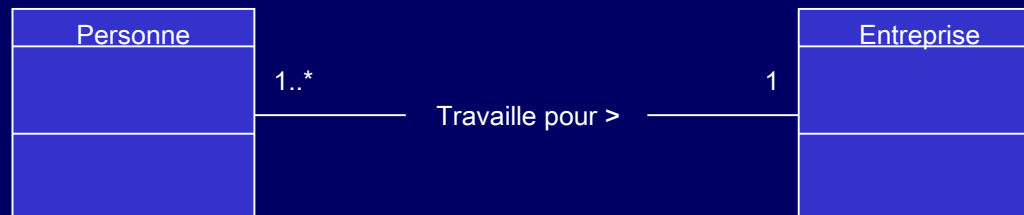
.....

- Associations / agrégations et liens
 - Définition
 - Un lien est une **connexion** physique ou conceptuelle **entre instances** donc entre objets
 - Une **association** décrit un groupe de liens ayant une **même structure** et une **même sémantique**
 - Un lien est une instance d'une association
 - Chaque association peut être identifiée par son nom
 - Une association exprime une connexion sémantique bidirectionnelle ou unidirectionnelle (> ou <) entre deux classes



Principe de la modélisation UML

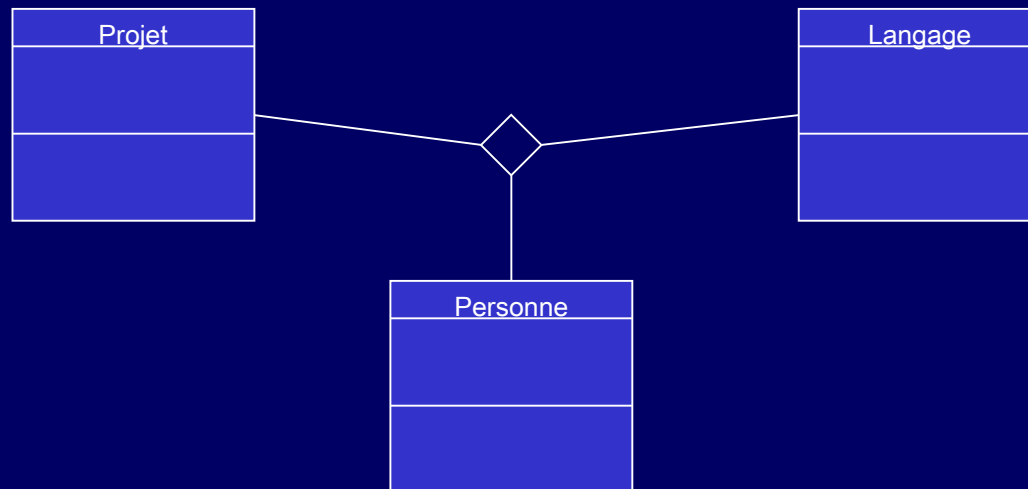
- Multiplicité des associations
 - Définition
 - La multiplicité correspond à la **cardinalité** des associations par rapport aux instances des classes impliquées
 - **Contraint** le nombre d'objets de la classe cible pouvant être associés à un seul objet donné de la classe source
 - Exemple de définition
 - Exactement un : 1 ou 1..1
 - Plusieurs : * ou 0..*
 - Au moins un : 1..*
 - De un à six : 1..6



Principe de la modélisation UML

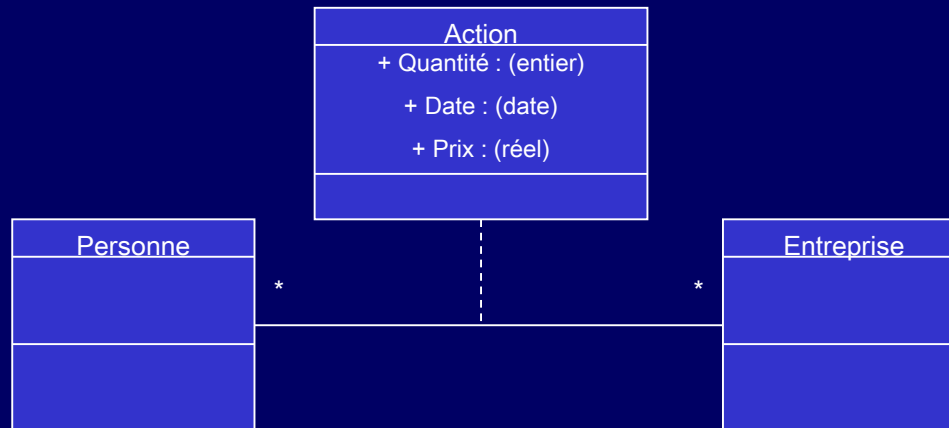
- Associations ternaires

- L'association ternaire permet de mettre en relation une association entre 3 classes
- Associations sont **difficiles à déchiffrer** et peuvent induire en erreur
 - **Limiter** leur utilisation, en définissant de nouvelles catégories d'associations



Principe de la modélisation UML

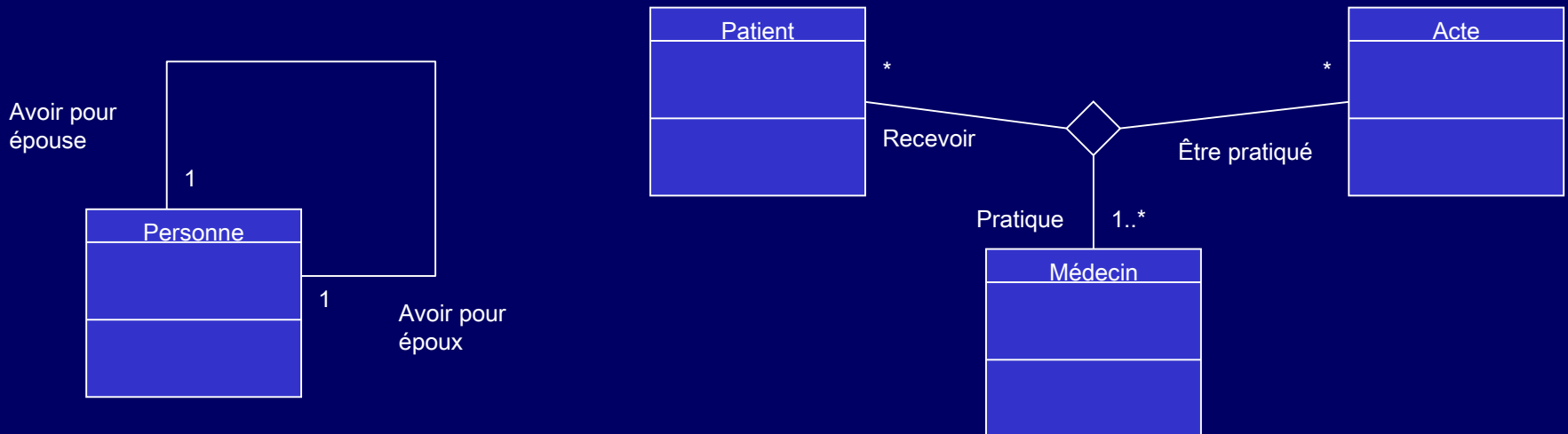
- Classe d'association
 - Définition
 - Classe qui réalise la navigation entre les instances d'autres classes
 - Une classe-association possède les propriétés des associations et des classes
 - Se connecte à deux ou plusieurs classes
 - Possède également des attributs et des opérations



Principe de la modélisation UML

- Rôle d'une association

- Chaque association peut être qualifiée par 2 rôles (un à chaque extrémité)
- Le rôle **explicite chaque sens** de l'association
- Particulièrement utile dans le cas d'association **réflexive** ou **ternaire**



Principe de la modélisation UML

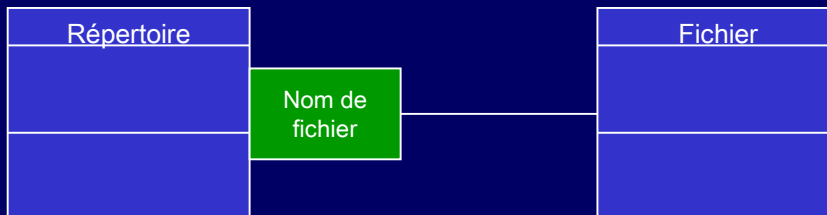
- Qualification d'une association

- Définition

- La qualification d'une association entre 2 classes permet de **préciser la sémantique** de l'association
 - Elle permet de qualifier de manière restrictive les liens entre les instances : seules les instances possédant l'attribut indiqué dans la qualification sont concernés



Un répertoire contient 0 à n fichiers



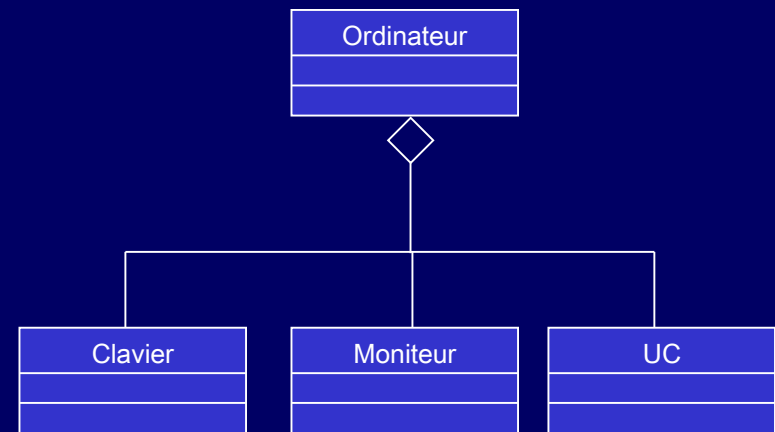
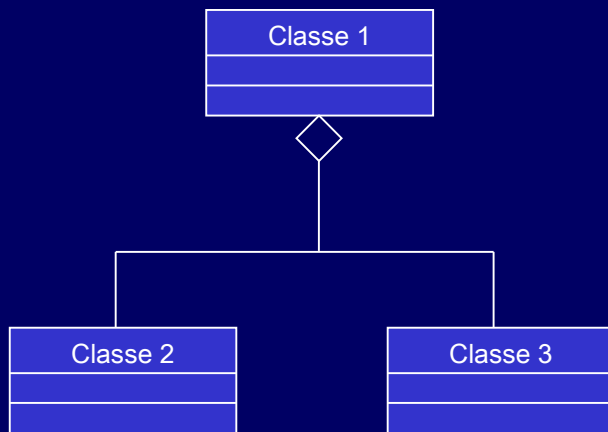
(répertoire, nom de fichier) <-> 1 fichier

Principe de la modélisation UML

- Agrégation

- Définition

- L'agrégation est une relation de type **composé/composant**
 - Elle représente un lien structurel entre une classe et une ou plusieurs autres classes
 - Contrairement à une association simple, l'agrégation est **transitive**

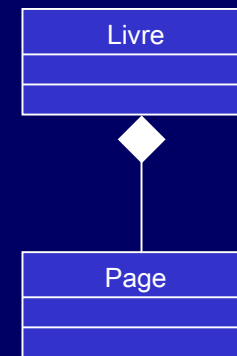
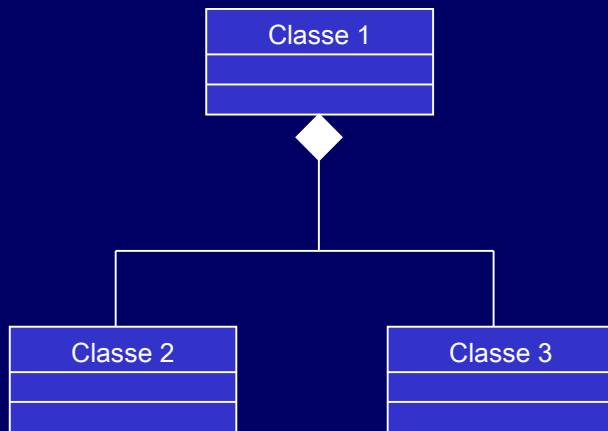


Principe de la modélisation UML

- Composition

- Définition

- Une composition est une agrégation **forte**
 - A un même moment, une instance de composant ne peut être liée qu'à **un seul** agrégat
 - Les "objets composites" sont des instances de classes composées



Principe de la modélisation UML

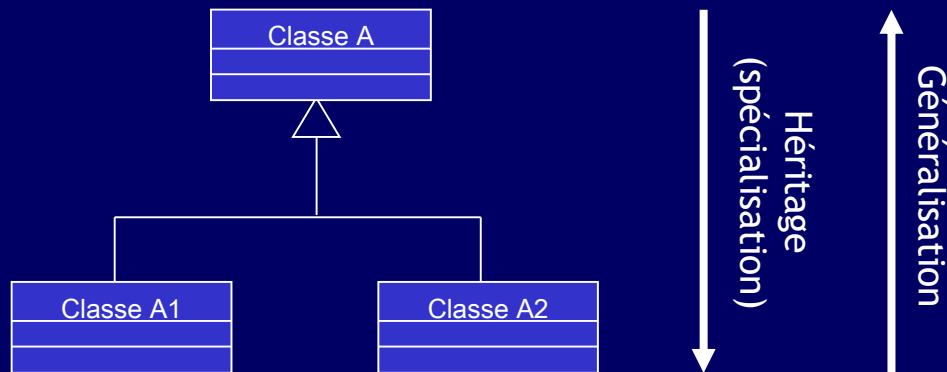
.....

- Généralisation / Héritage
 - Définition
 - **Généralisation** : relation entre une classe et deux autres classes (ou plus) partageant un sous-ensemble commun d'attributs
 - Le classe qui est affinée s'appelle la **super-classe**
 - Les classes affinées s'appellent les **sous-classes**
 - Création d'une super-classe à partir des sous-classes = **généralisation**
 - Création des sous-classes à partir d'une super-classe = **héritage**
 - Rôle
 - Une sous-classe peut hériter des attributs et des opérations de sa super-classe
 - Un discriminant peut être utilisé pour exploiter le critère de spécification
 - Ex : la classe **machin** hérite de la classe **truc** signifie
 - un **machin** est un **truc**

Principe de la modélisation UML

• Généralisation / Héritage

- La **classe enfant** possède **toutes les propriétés** des ses classes parents (attention toutefois à la visibilité)
- Une **classe enfant** peut **redéfinir** (même signature) une ou plusieurs méthodes de la classe parent
- Un **objet** utilise les **opérations les plus spécialisées** dans la hiérarchie des classes
- Toutes les **associations** de la classe parent s'appliquent aux **classes dérivées**
- Une instance d'une classe peut être utilisée partout où une instance de sa classe parent est attendue
- Une classe peut avoir plusieurs parents ; on parle alors d'héritage multiple



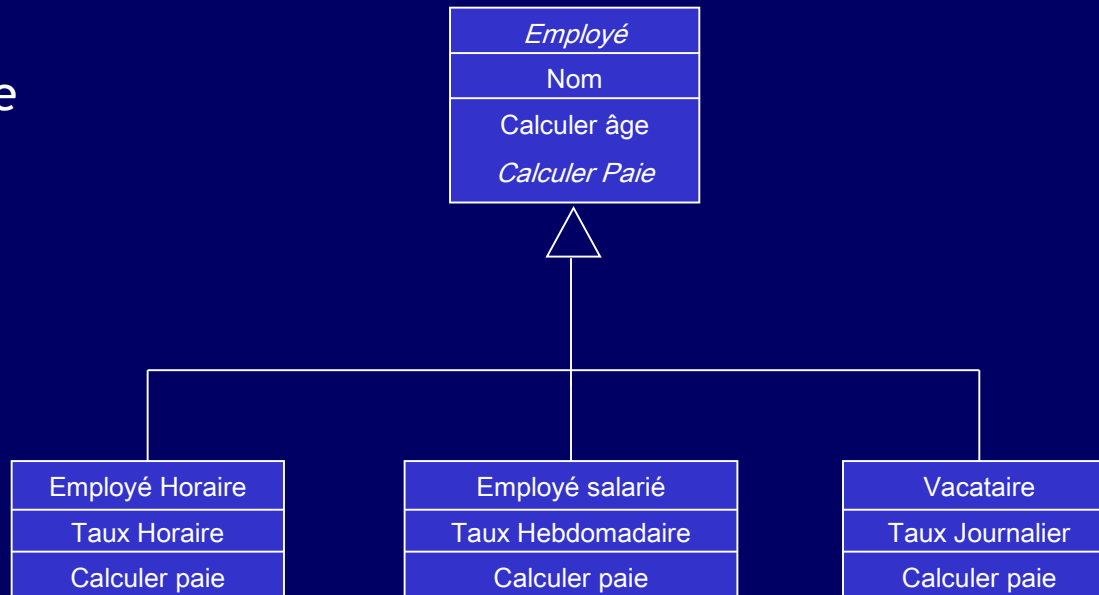
Principe de la modélisation UML

- Classe abstraite

- Définition

- Une **classe abstraite** est une classe qui n'a **pas d'instance directe** mais dont les classes descendantes peuvent avoir des instances
 - Les méthodes de la classe abstraite **peuvent** être implémentées ou non
 - Si elle ne le sont pas, elles doivent **obligatoirement** l'être dans les classes descendantes

- Exemple



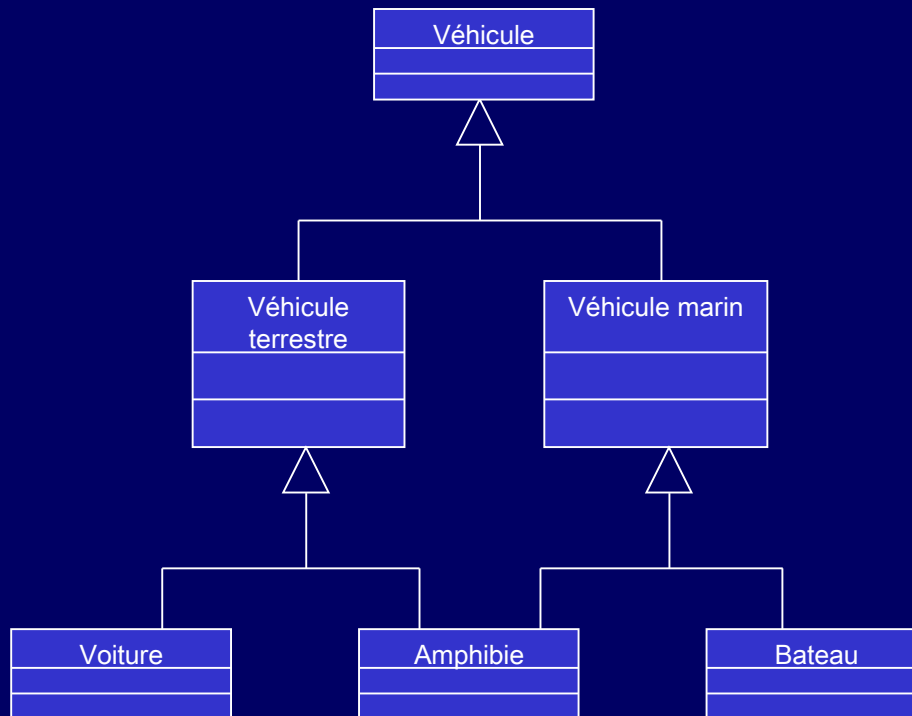
Principe de la modélisation UML

.....

- But : définir des **méthodes** qui doivent **impérativement être héritées** par les sous-classes
 - Définir des méthodes héritées dans les sous-classes
 - Factoriser des méthodes communes
 - Une classe abstraite **définit un protocole** pour une opération (signature) **sans fournir** de méthodes correspondante
- Dépendance du niveau de détail considéré
 - Une classe abstraite peut devenir concrète à un niveau de détail moins élevé (disparition de ses sous-classes)
 - Réciproque également vraie

Principe de la modélisation UML

- Héritage multiple
 - Il peut être nécessaire de faire hériter une même classe de deux parentes distinctes



Principe de la modélisation UML

.....

- Polymorphisme

- Intérêt de la **redéfinition** des opérateurs (opérations)
 - Extension : la nouvelle opération est la même que celle héritée à l'exception de la **modification de son comportement** du aux nouveaux attributs de la sous-classe
 - Ex : fenêtre ; redessiner() / fenêtreAvecNom ; redessiner()
 - Restriction : spécification des arguments à la sous-classe
 - Ex : Ensemble ; ajouter(objet) / EnsembleEntiers ; ajouter(entier)
 - Optimisation interne
 - Tirer avantage des contraintes propres à la sous-classe
 - Même protocole externe et même résultat mais sa représentation interne change
 - Ex : Ensemble ; rechercher(objet) / EnsembleTrié ; rechercher(objet)

Principe de la modélisation UML

.....

- Règles à appliquer
 - Toutes les opérations de requêtes (accès aux attributs) sont hérités
 - Les opérations de mise à jour sont héritées au travers de toutes les extensions
 - Les opérations de mise à jour sur des attributs contraints sont bloqués par restriction
 - Ex : Ellipse ; changerPetitRayon() / Cercle ; -
 - Les opérations ne peuvent pas être redéfinies pour avoir un comportement différent de celui des opérations héritées
 - Ex : Fenêtre ; déplacer() / FenêtreAvecNom ; déplacer()
 - On peut affiner les opérations héritées en leur ajoutant un comportement supplémentaire
 - Ex : Fenêtre ; afficher() / FenêtreAvecNom ; afficher()

Principe de la modélisation UML

.....

- Diagramme de classe (modèle objet) : Trucs pratiques
 - Atout principal : **simplicité**
 - Ne pas entrer immédiatement dans un niveau de détail maximum
 - Pas de référence et de pointeur comme attribut : à **modéliser comme relation**
 - Les **noms de classe** et **d'association** sont très importants
 - Pensez aux multiplicités en second lieu
 - Évitez si possible les association n -aire ($n > 2$)
 - Pas d'attributs de liens dans les classes
 - Utilisez les associations qualifiées le plus souvent possible

Principe de la modélisation UML

.....

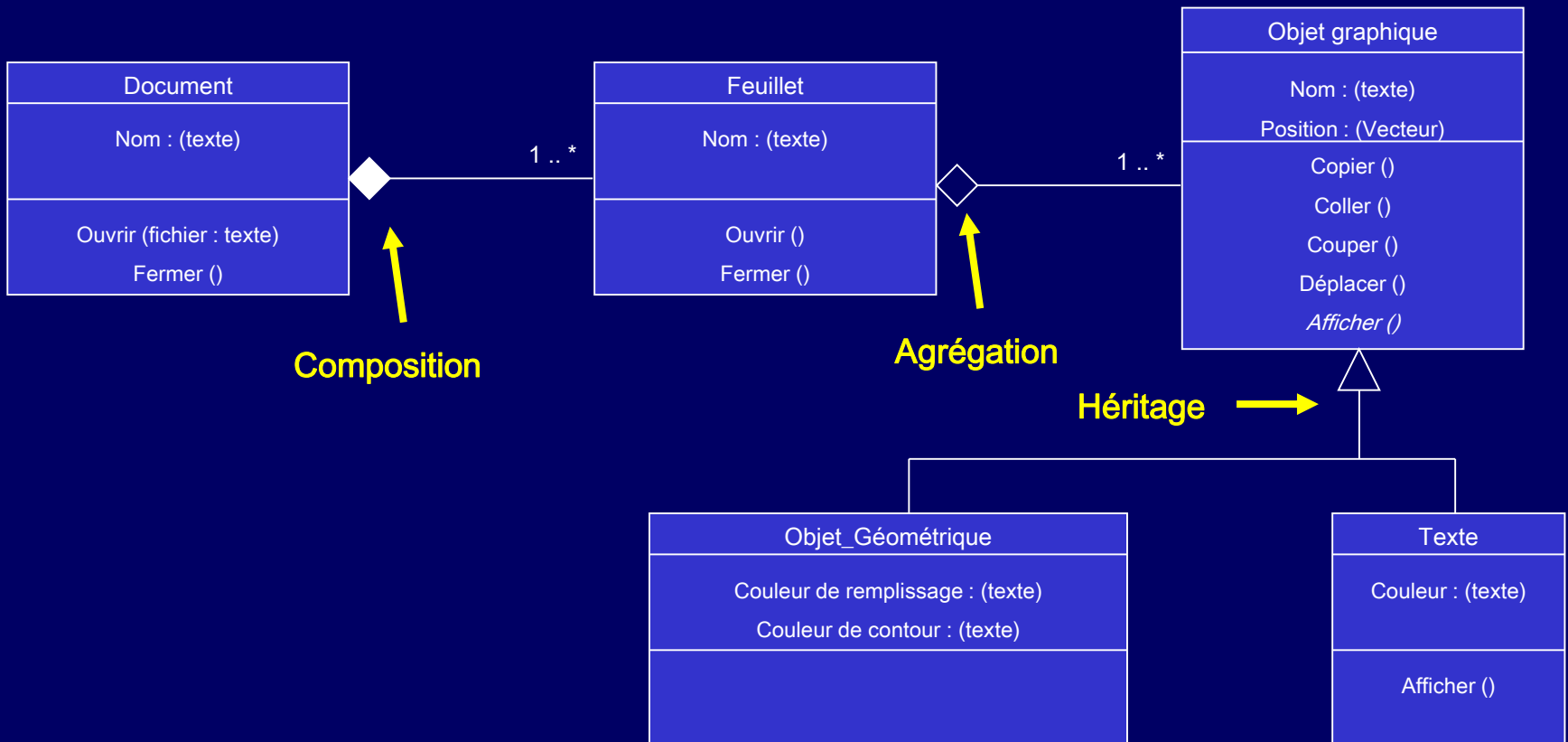
- Un modèle objet se construit par **révision successives**
 - Augmenter le niveau de détail
 - Clarifier les noms
- Soumettez vos modèles à des **avis extérieurs**
- **Documentez** systématiquement vos modèles
 - Le diagramme rend compte de la structure mais n'explique pas les choix
 - Une explication écrite des choix est souvent nécessaire
- **Ne surchargez pas les modèles** avec les constructions UML
 - Limitez-vous aux notations qui sont **essentielles**

Principe de la modélisation UML

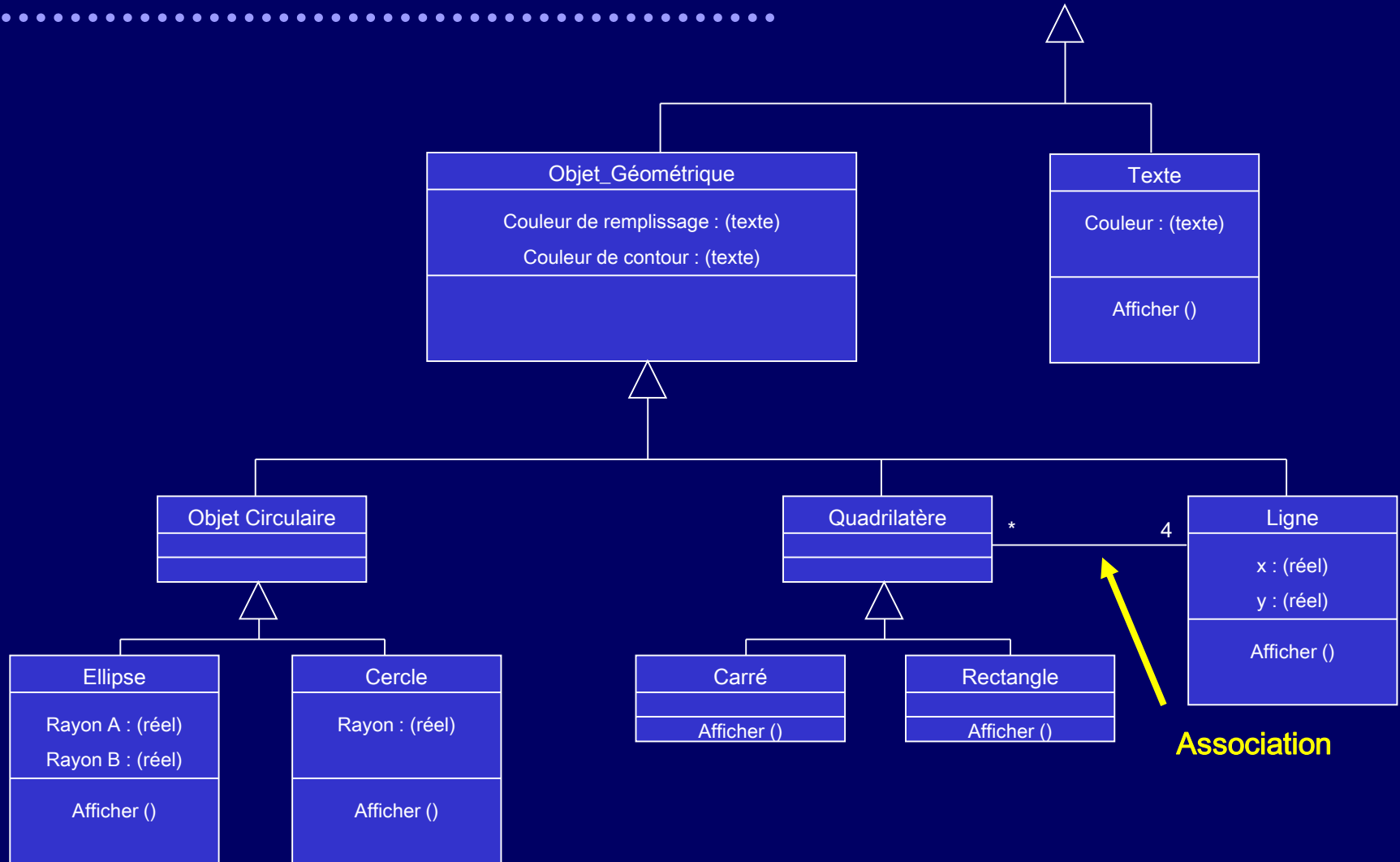
.....

- Exemple
 - Gestion technique de documents
 - Chaque document comporte un ou plusieurs feuillet
 - Un feuillet comporte des objets graphiques
 - Du texte
 - Des objets géométriques
 - Opérations sur les objets graphiques
 - Sélectionner, couper, copier, coller, déplacer
 - Objets géométriques
 - Cercle
 - Ellipse
 - Carré
 - rectangle

Principe de la modélisation UML



Principe de la modélisation UML



La conception objet : Plan

.....

- Introduction / historique
- Concept objet
 - Modularité / Réutilisabilité
 - Vue d'ensemble
- Principe de la modélisation UML
 - Présentation générale
 - Démarche de modélisation
 - Diagramme de classe
 - Objet et classes
 - Encapsulation
 - Associations / agrégations
 - Héritage
 - Polymorphisme
 - **Diagramme d'états-transitions**
 - Diagramme de communication

Principe de la modélisation UML

.....

- Diagramme d'états-transitions
 - Plusieurs concepts sont associés à ce diagramme
 - États
 - Évènement
 - Transition
 - Condition
 - Action
 - Activité

Principe de la modélisation UML

.....

- États

- L'**état** d'un objet se définit à un instant donné par les **valeurs** de ses propriétés
- Seuls certains états caractéristiques du domaine sont étudiés
- Ex : pour un employé donné d'une entreprise les états intéressants peuvent être
 - Recruté
 - En activité
 - En congé
 - Malade
- Un diagramme d'état est un **graphe d'état et d'évènement**
 - décrit le comportement interne d'un objet à l'aide d'un automate à états finis
 - offre une vision complète et non ambiguë de l'ensemble des comportements d'une instance

Principe de la modélisation UML

.....

- Évènement
 - Un diagramme d'états-transitions spécifie les **réactions à des évènements**
 - Fait survenu qui fait passer un objet d'un état à un autre état
 - Considéré comme **instantané**
 - Les évènements peuvent être ou non liés à des relations de causalité
 - Ex : le vol 426 pour Paris doit partir après le vol 243 pour Madrid : les évènements sont les « départs »
 - 2 évènements sans lien de causalité sont dits **concurrents**
 - Un évènement est aussi une voie de transmission d'information d'un objet vers un autre

Principe de la modélisation UML

.....

- Transition
 - Elle correspond au passage d'un état à un autre état
 - Certaines **transitions** sont **automatiques** : effectuées dès que les activités associées à un état sont finies
 - Si des conditions sont associées alors la transition automatique doit être validée
- Condition
 - Valeur **booléenne** des valeurs de l'objet
 - Elle peut être valide pendant un intervalle de temps
 - Utilisées comme **garde** sur les transitions
 - Une transition gardée est franchie si l'évènement surgit ET si la condition est vérifiée

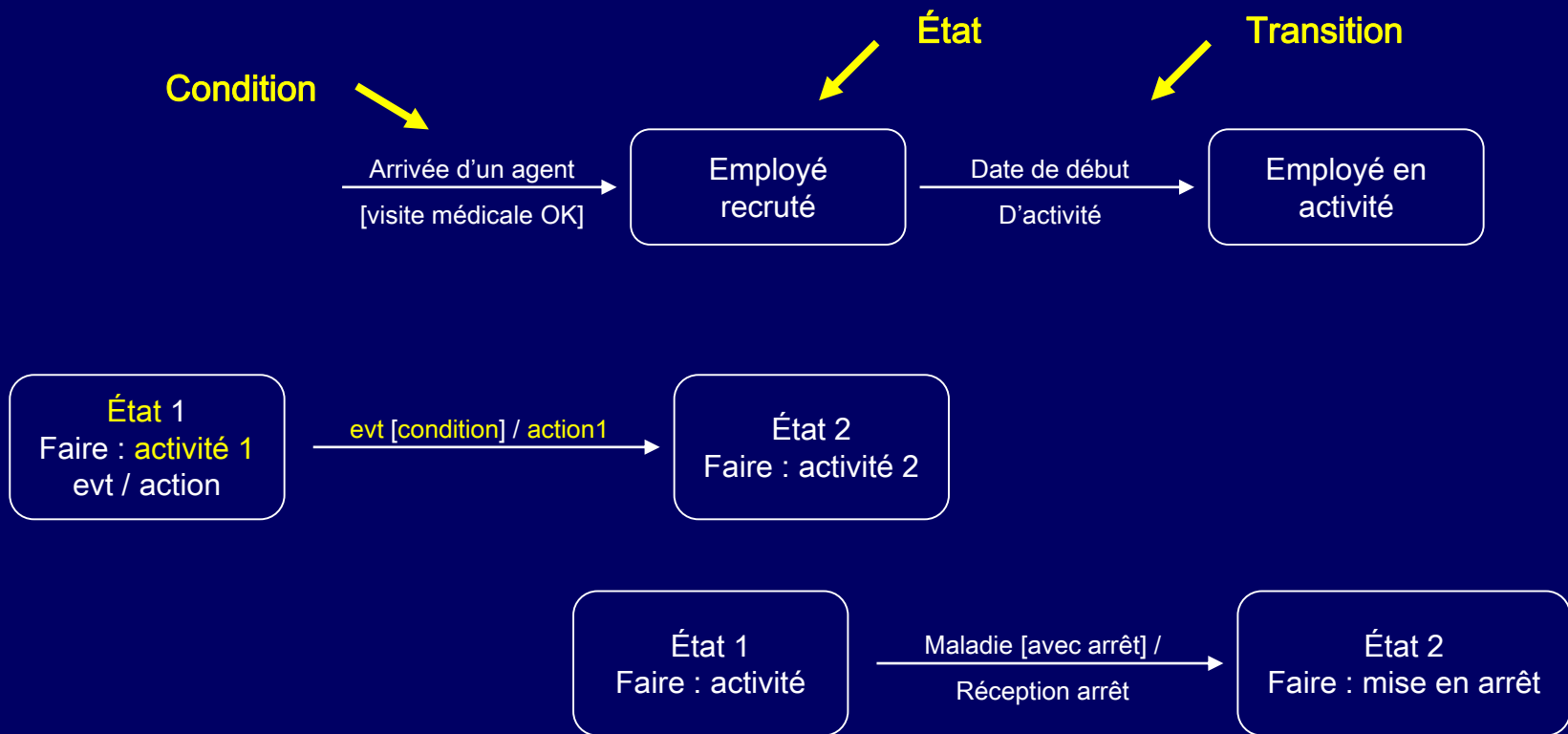
Principe de la modélisation UML

.....

- Action
 - Une action est une **opération instantanée** qui **ne peut être interrompue**, elle est associée à un évènement
 - Certaines actions peuvent être associées à un état : elles surviennent avec un évènement qui ne fait pas changer l'objet d'état
- Activité
 - Une activité est une opération d'une **certaine durée** qui peut être interrompue, elle est associée à l'état d'un objet
 - « faire : A » activité débute avec l'état et se termine avec lui si un évènement provoque le changement d'état, l'activité est interrompue

Principe de la modélisation UML

- Résumé concepts



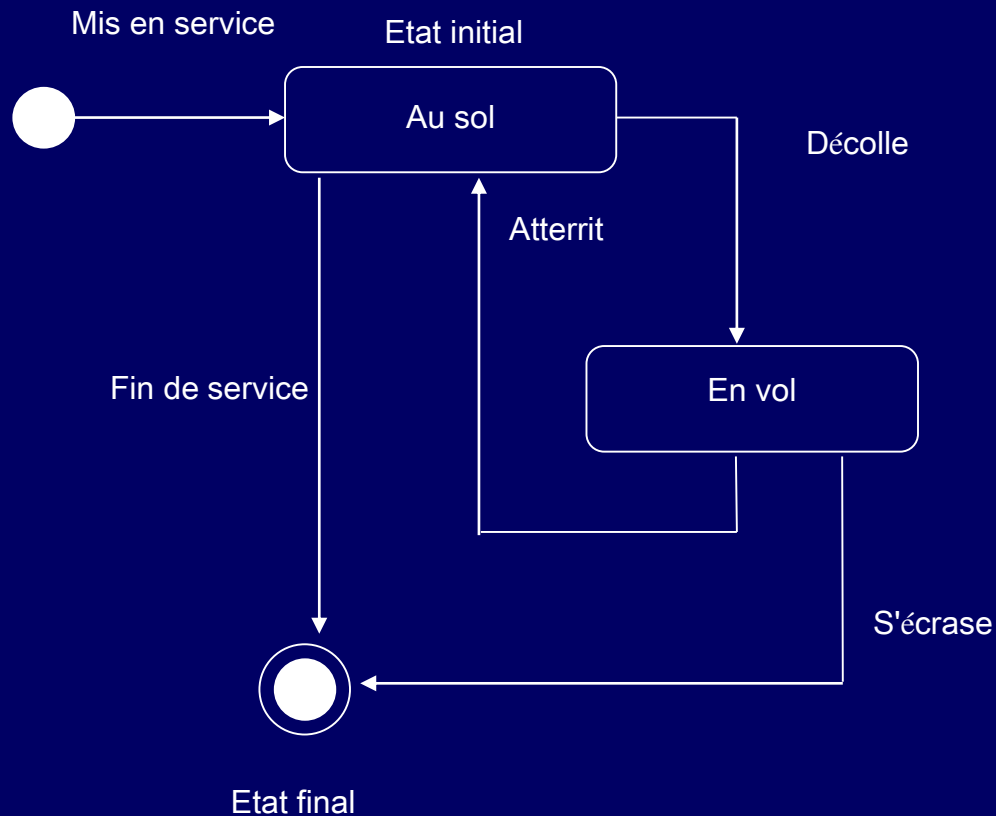
Principe de la modélisation UML

- Diagramme d'états d'un objet
 - Définition
 - L'enchaînement de tous les états caractéristiques d'un objet constitue le diagramme d'état
 - Un diagramme d'état par classe ; puisque les instances ont un comportement commun
 - Les nœuds sont les états, les arcs sont les transitions
 - Un diagramme d'état débute toujours par un état initial
 - Il se termine par un ou plusieurs états finaux (sauf si on représente une boucle !)



Principe de la modélisation UML

- Exemple



Principe de la modélisation UML

.....

- Diagrammes d'états imbriqués
 - But : permettre une description facile des systèmes complexes
 - Décrire les activités à un *haut niveau*
 - Décrire par la suite les activités en détaillant les activités imbriquées
 - Les évènements peuvent être organisés en une hiérarchie de généralisation avec héritage (hiérarchie de classe)
 - Un évènement d'entrée déclenche des transitions sur des types d'évènements ancêtres
 - Ex : l'évènement « entrée utilisateur » a deux fils : « bouton souris » et « caractère clavier »

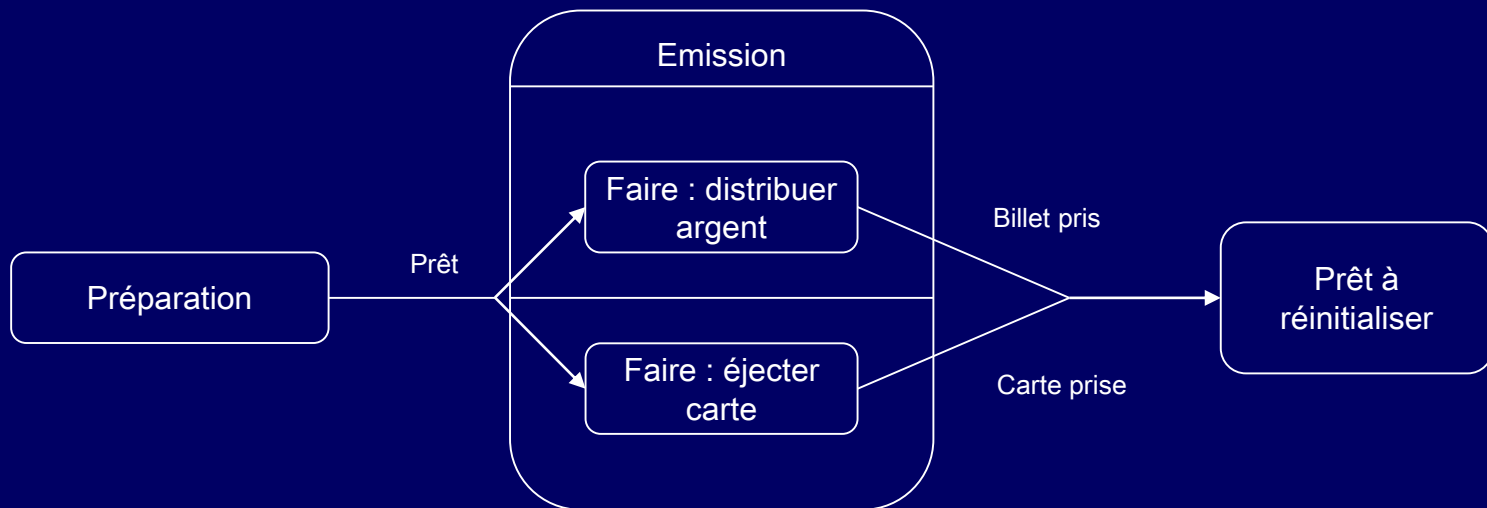
Principe de la modélisation UML

.....

- Concurrence
 - Modèle dynamique = ensemble d'objets concurrents
 - Chaque objet a ses propres états
 - L'état du système entier est l'union de tous les états
 - Il est possible que les états des objets **concurrents interagissent** : les transitions pour un objet peuvent dépendre d'un autre objet
 - Concurrence à l'intérieur d'un objet
 - Si l'objet peut être partitionné en sous-ensemble ayant des sous-diagrammes d'états
 - Le même évènement peut provoquer des transitions dans plusieurs sous-diagrammes

Principe de la modélisation UML

- Synchronisation des activités concurrentes
 - Activités exécutées concurremment dont les étapes internes ne sont pas synchronisées, mais les activités doivent être terminées avant une transition vers un autre état
 - Ex : distributeur de billets



Principe de la modélisation UML

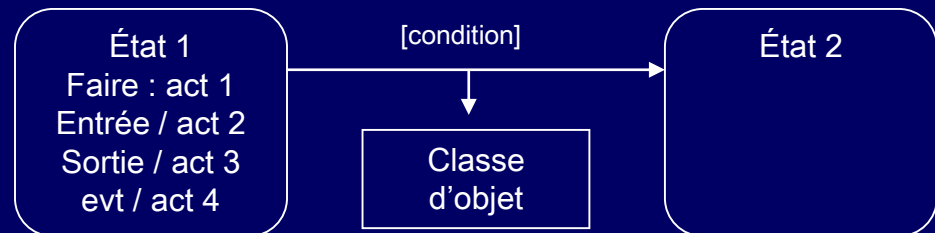
- Envoyer des évènements

- Définition

- Un objet peut exécuter une action **d'envoi d'un évènement** à un autre objet
 - Un évènement peut être destiné à un ou plusieurs objets
 - Un système objet **interagit** par **l'échange d'évènement**
 - Si l'évènement est envoyé à une personne ou un périphérique = action
 - Ex : tonalité-sonne est un évènement *sonne* envoyé à une personne

- Symbolisme

- « envoyer-E(attribut) » comme action i.e. liée à une transition
 - Autre symbolisme



Principe de la modélisation UML

.....

- Règles pratiques
 - Ne construire des diagrammes d'états que pour les classes d'objet **ayant un comportement dynamique** !
 - Ne considérer que les attributs pertinents d'un objet pour définir son état
 - Considérez les **besoins de l'application** quand vous prenez des décisions sur la « granularité » des évènements et des états
 - L'application guide le choix entre **actions** (instantanées) et **activités** (sur un laps de temps significatif)
 - La hiérarchie d'évènements est **indépendante** de la hiérarchie de classes
 - Les sous-classes **héritent** des états et des transitions de leurs ancêtres : le diagramme d'état d'une sous-classe doit être un ajout indépendant du diagramme d'état des ancêtres

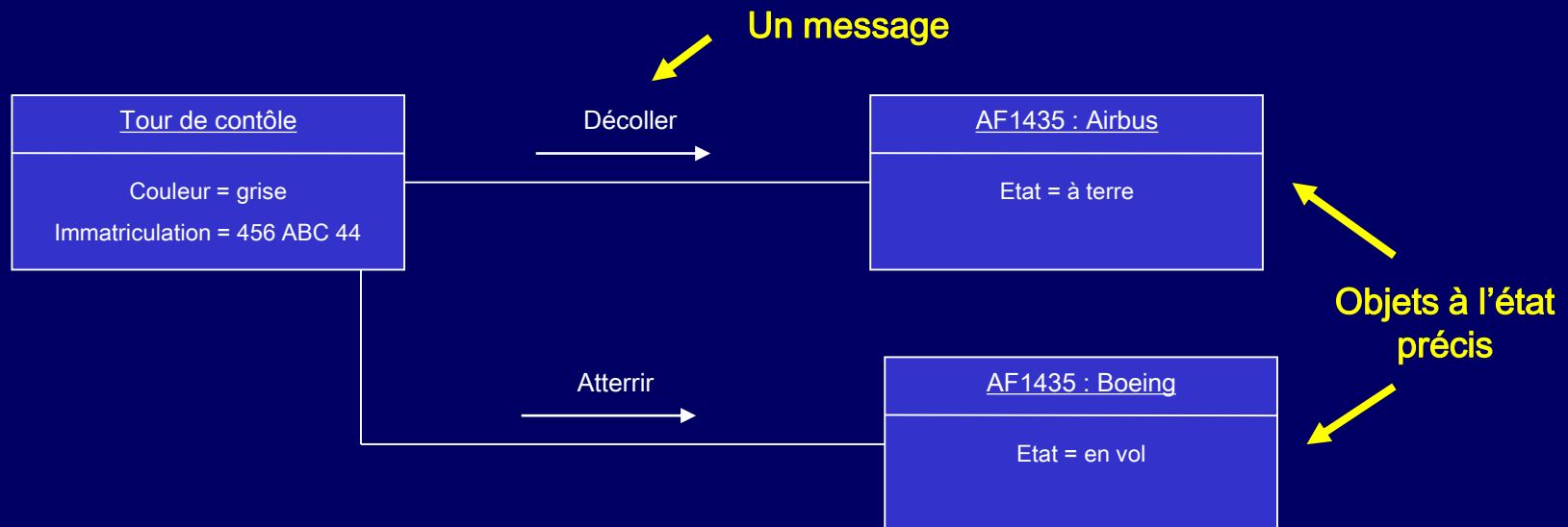
La conception objet : Plan

.....

- Introduction / historique
- Concept objet
 - Modularité / Réutilisabilité
 - Vue d'ensemble
- Principe de la modélisation UML
 - Présentation générale
 - Démarche de modélisation
 - Diagramme de classe
 - Objet et classes
 - Encapsulation
 - Associations / agrégations
 - Héritage
 - Polymorphisme
 - Diagramme d'états-transitions
 - **Diagramme de communication**

Principe de la modélisation UML

- Diagramme de collaboration
 - Communication entre objets
 - Une application OO est une société **d'objets collaborant**
 - L'achèvement d'une tâche par une application repose sur la communication entre les objets qui la composent
 - L'unité de communication entre les objets est le **message**



Principe de la modélisation UML

.....

- Le concept de message
 - Il existe cinq catégories principales de messages
 - les **constructeurs** qui créent des objets
 - les **destructeurs** qui détruisent des objets
 - Pas dans tous les langages objets
 - les **sélecteurs** qui renvoient tout ou partie de l'état d'un objet
 - les **modificateurs** qui changent tout ou partie de l'état d'un objet
 - les **itérateurs** qui visitent l'état d'un objet ou le contenu d'une structure de données qui contient plusieurs objets

En python

.....

```
class Point :  
    def __init__( self , x , y ) :  
        self.x = x  
        self.y = y
```

- Déclaration d'une fonction `__init__`
 - Constructeur
 - Appelée automatiquement à la création d'une variable `Point`
 - Permet d'initialiser les attributs de la classe
- Le paramètre `self` est un paramètre caché
 - Géré automatiquement par Python
 - Permet de référencer directement la nouvelle instance de la classe créé

En python

.....

- Créer une variable `p` contenant une instance de la classe `Point`

```
p1 = Point(1,2)
p2 = Point(3,4)
print(p1.x," - ",p1.y)
print(p2.x," - ",p2.y)
p1.x=10
print(p1.x)
```