1

Analyse et Conception de Systèmes Informatiques Orientés objets en UML

Abdelhak-Djamel SERIAI
http://www.lirmm.fr/~seriai
seriai@lirmm.fr
2013/2014

2

Objectifs du cours

1. **Introduction au paradigme orienté-objet**
2. **Maîtriser UML pas à pas**
   1. Les différents diagrammes
   2. Introduction à OCL
   3. Passage d'UML à la programmation et au BD
3. **Introduction aux patrons de conception**
4. **Introduction au processus unifié RUP et à l'IDM**
5. **Application d'UML pour le développement d'un mini-projet**

3

Partie I : Les motivations (Le pourquoi ?)

4

Le produit logiciel (01)

- **Informatique et logiciels**
  - Informatique de gestion
    - Systèmes d'information d'entreprises
    - Aide à la décision
    - Etc.
  - Informatique industrielle
    - Chaîne de production
    - Systèmes de contrôles : bateau, avion, train, etc.

  - Informatique médicale
    - Scanneur, décision, etc.
  - autres

5

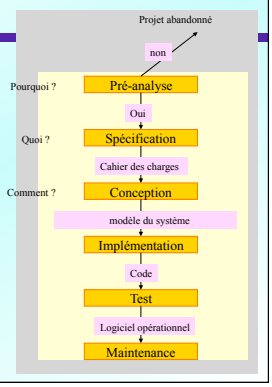
Le produit logiciel (02)

- **Un logiciel est un produit (manufacturé)**
  - Comme une voiture ou une machine à outils
  - Nécessité d'un processus de développement
    - Déterminer les besoins
    - Plan ou architecture du système
    - Conception du système
    - Réalisation du système
    - Etc.

6

Cycle de vie de logiciel

- **Cycle de vie de logiciel**
  - La phase de pré-analyse
    - Étude d'opportunité
  - La phase de spécification
    - Analyse des besoins
  - La phase de conception
    - La conception architecturale
    - Conception détaillée
  - La phase d'implémentation
    - La programmation
  - La phase de test
    - La validation et vérification
  - La phase de maintenance



7

Méthodes de développement(1)

- **Développer un modèle d'un système**
  - Un modèle est une représentation abstraite d'une réalité ou une partie d'une réalité
    - Exemple de modèles
      - Modèle d'un événement physique
      - Modèle d'une réaction chimique
      - Modèle d'un système informatique
      - Modèle d'un mécanisme de fonctionnement électronique
    - Exemple de formes de représentation
      - Mathématique
      - Graphique
      - Semi-graphique

8

Méthodes de développement(2)

- **Difficultés de la modélisation**
  - **Problèmes des spécifications**
    - parfois imprécises, incomplètes, ou incohérentes
  - **Taille et complexité des systèmes importantes et croissantes**
    - les besoins et les fonctionnalités augmentent
    - la technologie évolue rapidement
    - les architectures se diversifient
    - assurer l'interface avec le *métier* (domaine d'application)
  - **Évolution des applications**
    - évolution des besoins des utilisateurs
    - réorientation de l'application
    - évolution de l'environnement technique (matériel et logiciel)
  - **Problèmes liés à la gestion des équipes**
    - taille croissante des équipes
    - spécialisation technique
    - spécialisation *métier*

9

Méthodes de développement(3)

- **Les méthodes d'analyse et de conception**
  - proposent une démarche qui distingue les étapes du développement dans le cycle de vie du logiciel (modularité, réduction de la complexité, réutilisabilité éventuelle, abstraction)
  - s'appuient sur un formalisme de représentation qui facilite la communication, l'organisation et la vérification : Le langage de modélisation
  - Produisent des documents (modèles) qui facilitent les retours sur conception et l'évolution des applications

10

Méthodes de développement(4)

- **Classification des méthodes de développement**
  - La distinction fonctionnel (dirigée par le traitement)/orientée objet.
    - Les stratégies fonctionnelles :
      - Un système est vu comme un ensemble d'unités en interaction, ayant chacune une fonction clairement définie.
      - Exemple : SA, SADT, SA/RT, RAPID/USE, JSD, MASCOT, Automates; AEFC, RDP
    - Les stratégies orientées objet
      - Considèrent qu'un système est un ensemble d'objets interagissant.
      - Contrairement à la plupart des méthodes fonctionnelle, les méthodes orientée-objet sont à la fois des méthodes d'analyse et de conception
      - Exemples: OMT, UML

11

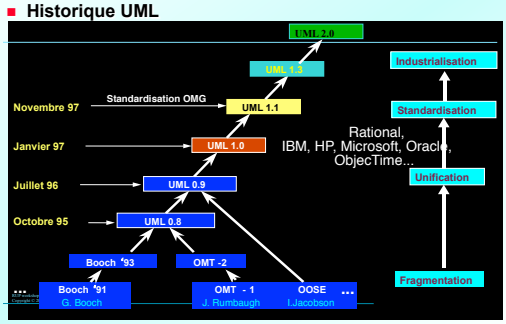
Méthodes de développement(5)

- **Classification des méthodes de développement**
  - La distinction composition/décomposition :
    - Les méthodes ascendantes consistent à construire un logiciel par composition à partir de modules existants
    - Les méthodes descendantes décomposent récursivement un système jusqu'à arriver à des modules programmables « simplement »
  - La distinction formelle/non formelle
    - Approche non formelle ou semi-formelle
      - Permettent de spécifier certaines parties de l'analyse ou/et de la conception à l'aide de langages formels tel que le langage OCL pour UML
    - Approches formelles
      - Approches mathématiques de spécification et de modélisation
      - Exemples : spécification en langage B et Z

12

Introduction UML(1)

- **Historique UML**



13

Introduction UML(2)

- **Présentation de la méthode UML**
  - UML est la forme contractée de *Unified Modeling Language* qui peut se traduire en français par *langage unifié pour la modélisation*
  - C'est un langage de modélisation objet
  - *UML* se base sur une notation graphique expressive.
  - Il permet de modéliser de manière claire et précise la structure et le comportement d'un système indépendamment de toute méthode ou de tout langage de programmation
  - La notation *UML* repose sur plusieurs diagrammes adaptés au développement logiciel pour les phases de spécification, conception et implémentation du cycle de vie.

14

Introduction UML(3)

- **Présentation du langage UML**
  - Différents types de diagrammes
    - Expression des besoins
      - Use cases et diagrammes d'activité
    - Identification des entités (de la structure) du système étudié
      - Diagrammes de classes : Classes, Associations, Attributs, Opérations; Assertions, invariants, pré-conditions, postconditions; Généralisations, Classes associations
    - Description du comportement attendu
      - Diagramms d'interaction : Descriptions des interactions entre « groupes » d'objets: messages échangés entre objets.
      - Diagrammes d'états :Description des états des objets

15

Introduction UML(5)

- **Comment modéliser avec UML ?**
  - Définition
    - UML est un langage qui permet de représenter des modèles
      - UML permet de définir et de visualiser un modèle à l'aide de diagrammes
      - Un diagramme UML est une représentation graphique, qui s'intéresse à un aspect précis du modèle ;
        - Combinés, les différents types de diagrammes UML offrent une vue complète des aspects statiques et dynamiques d'un système.
      - UML offre des représentations semi-formelles
        - Graphique
        - Contraintes mathématiques (formelles) dans le langage OCL (Object constraint language)
    - UML ne définit pas le processus d'élaboration des modèles

16

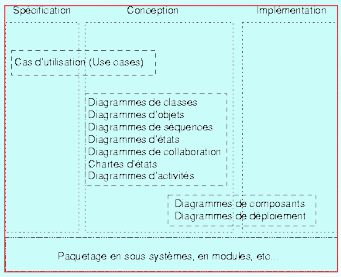
Introduction UML(6)

- **Comment modéliser avec UML ?**
  - Diagrammes UML
    - Vues statiques ou structurelles du système
      - diagrammes de cas d'utilisation
      - diagrammes d'objets
      - diagrammes de classes
      - diagrammes de composants
      - diagrammes de déploiement
    - Vues dynamiques ou comportementales du système
      - diagrammes de collaboration
      - diagrammes de séquence
      - diagrammes d'états-transitions
      - diagrammes d'activités

17

Introduction UML(7)

- **Comment modéliser avec UML ?**
  - Diagrammes UML



18

Les concepts objet à travers UML

19

Les concepts objet à travers UML (01)

- **Un peu d'histoire**
  - 1967 : Simula, 1er langage de programmation à implémenter le concept de type abstrait à l'aide de classes
  - 1976 : Smalltalk implémente les concepts fondateurs de l'approche objet (encapsulation, agrégation, héritage) à l'aide de :
    - classes
    - associations entre classes
    - hiérarchies de classes
    - messages entre objets
  - 1980 : le 1er compilateur C++. C++ est normalisé par l'ANSI et de nombreux langages orientés objets académiques ont étayés les concepts objets : Eiffel, Objective C, Loops, java, …
  - Standardisation d'UML 1.1 en 1997

20

Les concepts objet à travers UML (02)

- **Un peu d'histoire**
  - L'approche fonctionnelle n'est pas adaptée au développement d'applications qui évoluent sans cesse et dont la complexité croit continuellement
    - Les fonctions ne sont pas les éléments les plus stables
    - On peut pas décrire toujours un système par une fonction principale
    - Difficulté de la réutilisation

21

Les concepts objet à travers UML (03)

- **Système à objets ou orienté objet**
  - Les systèmes non objet
    - Procéduraux : sous-programmes
    - Fonctionnels : fonctions
    - Déductifs : règles
  - Les systèmes objets
    - Le logiciel est une collection d'objets dissociés définis par des propriétés
    - L'approche objet facilite le développement et l'évolution d'applications complexes.

22

Les concepts objet à travers UML (04)

- **Le paradigme Objet**
  - Programmation orientée objet
    - Méthode d'implantation dans laquelle le programme est organisé en collection d'objets coopératifs. Chaque objet est une instance de classe. Toutes les classes sont membres d'une hiérarchie de classes liées pas des relations d'héritage
  - Conception orientée objet
    - Méthode de conception qui mène à une décomposition orientée objet et utilise une notation pour représenter les différents aspects du système en cours de conception
  - Analyse orientée objet
    - Méthode d'analyse qui examine les besoins en termes de classes et d'objets trouvé dans l'espace du problème.

23

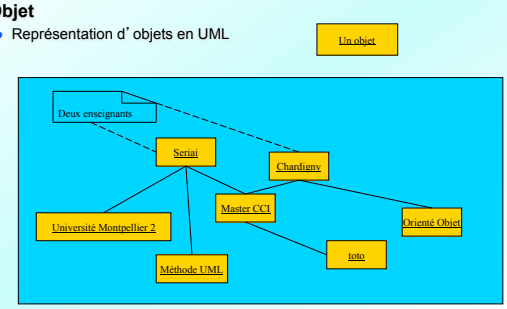
Les concepts objet à travers UML (05)

- **Objet**
  - Définition
    - Objet du monde réel :
      - Perception fondée sur le concept de masse
        - grains de sable, les étoiles
    - Objet informatique :
      - Est une unité atomique formée de l'union d'un état et d'un comportement
      - Définit une représentation abstraite d'une entité du monde réel ou virtuel, dans le but de la piloter ou de la simuler
        - Grain de sable, étoile
        - Compte en banque, police d'assurance, équation mathématiques, etc.
    - Les objets du monde réel et du monde informatique naissent, vivent et meurent

24

Les concepts objet à travers UML (06)

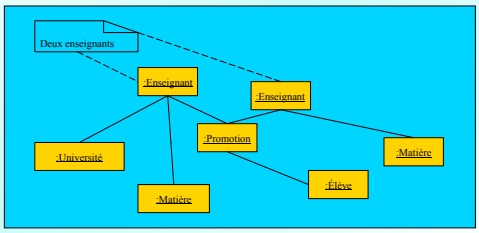
- **Objet**
  - Représentation d'objets en UML



25

Les concepts objet à travers UML (07)

- **Objet**
  - Représentation d'objets en UML



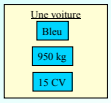
26

Les concepts objet à travers UML (08)

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

Objet = État + Comportement + Identité

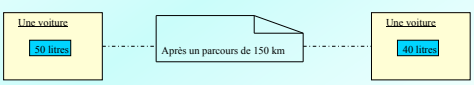
- État
  - Regroupe les valeurs instantanées de tous les attributs d'un objet :
    - Un attribut est une information qui qualifie l'objet qui le contient
    - Chaque attribut peut prendre une valeur dans un domaine de définition donné
  - Exemple : Un objet voiture regroupe les valeurs des attributs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



27

Les concepts objet à travers UML (09)

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

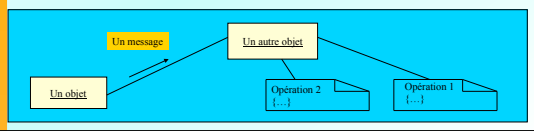


      - Certaines composantes de l'état peuvent être constantes
        - La marque de la voiture, pays de la construction de la voiture

28

Les concepts objet à travers UML (10)

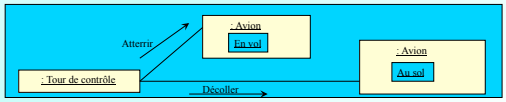
- **Objet**
  - Caractéristique fondamentales d'un objet (informatique)
    - Le comportement
      - Regroupe toutes les compétences d'un objet et décrit les actions et les réactions de cet objet
      - Chaque atome (partie) de comportement est appelé opération
        - Les opérations d'un objet sont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



29

Les concepts objet à travers UML (11)

- **Objet**
  - Caractéristique fondamentales d'un objet (informatique)
    - Le comportement
      - Le comportement à un instant donné dépend de l'état courant et l'état peut être modifié par le comportement
        - Exemple : il n'est pas possible de faire atterrir un avion que s'il est en train de voler: le comportement *Atterrir* n'est valide que si l'information *En vol* est valide
        - Après l'atterrissage, l'information *En vol* devient invalide et l'opération *Atterrir* n' plus de sens



30

Les concepts objet à travers UML (12)

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

31

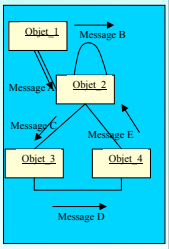
Les concepts objet à travers UML (13)

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

32

Les concepts objet à travers UML (14)

- **Objet**
  - Communication entre objets : le concept de message
    - Les systèmes informatiques à objets peuvent être vus comme des sociétés d'objets qui travaillent en synergie afin de réaliser les fonctions de l'application
    - Le comportement global d'une application repose sur la communication entre les objets qui la composent
    - L'unité de communication entre objets s'appelle message
    - Il existe cinq catégories principales de massages
      - Les constructeurs qui créent des objets,
      - Les destructeurs qui détruisent des objets,
      - Les sélecteurs qui renvoient tout ou partie de l'état de l'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.



33

Les concepts objet à travers UML (15)

- **Classe**
  - Définition
    - Une classe décrit une abstraction d'objets ayant
      - Des propriétés similaires
      - Un comportement commun
      - Des relations identiques avec les autres objets
      - Une sémantique commune
    - Par exemple Fichier (resp. Paragraphe, resp. Véhicule) est la classe de tous les fichiers (resp. paragraphes, resp. véhicules)
    - Une classe a trois fonctions:
      - Sert de ``patron'' (*template*) à objets : elle définit la structure générale des objets qu'elle crée en indiquant quelles sont les *variables d'instance* ;
      - Sert de ``conteneur'' d'objets : contient et donc donne l'accès à l'ensemble des objets qu'elle a créés;
      - Sert de ``réceptacle'' des *méthodes* que ses objets peuvent utiliser puisque tous les objets d'une classe utilisent les mêmes *méthodes* , il serait inutile de les dupliquer dans ces objets eux-mêmes.

34

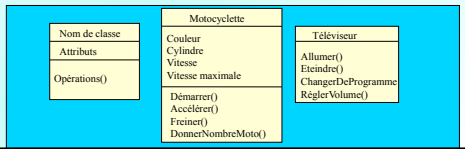
Les concepts objet à travers UML (16)

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

35

Les concepts objet à travers UML (17)

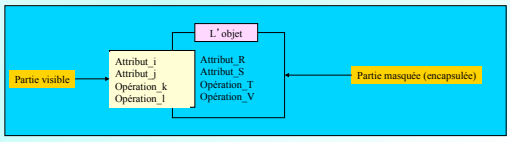
- **Classe**
  - Représentation graphique d'une classe en UML
    - Chaque classe est représentée sous la forme d'un rectangle divisé en trois compartiments
    - Les compartiments peuvent être supprimés pour alléger les diagrammes
    - Représentation des classes abstraites : le non d'une classe abstraite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



36

Les concepts objet à travers UML (18)

- **Encapsulation**
  - Consiste à masquer les détails d'implémentation d'un objet, en définissant une interface.
    - Est la séparation entre les propriétés externes, visibles des autres objets, et les aspects internes, propres aux choix d'implantation d'un objet.
  - L'interface est la vue externe d'un objet, elle définit les services accessibles (offerts) aux utilisateurs de l'objet.



37

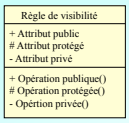
Les concepts objet à travers UML (19)

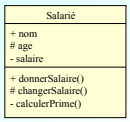
- **Encapsulation**
  - Présente un double avantage
    - facilite l'évolution d'une application car elle stabilise l'utilisation des objets : on peut modifier l'implémentation des attributs d'un objet sans modifier son interface
      - Les utilisateurs d'une abstraction ne dépendent pas de la réalisation de l'abstraction mais seulement de sa spécification : ce qui réduit le couplage dans les modèles
    - garantit l'intégrité des données, car elle permet d'interdire l'accès direct aux attributs des objets (utilisation d'accesseurs)
      - Les données encapsulées dans les objets sont protégées des accès intempestifs

38

Les concepts objet à travers UML (20)

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

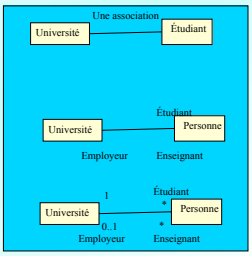




39

Les concepts objet à travers UML (21)

- **Les relations entre les classes**
  - Associations
    - Représentent les relations structurelles qui existent entre objets de différentes classes.
    - La plupart des associations sont binaires
    - Une association est caractérisée par:
      - un nom, qui peut être omis notamment quand les rôles des classes sont spécifiés
      - un rôle pour chacune des classes qui participent à l'association.
      - une multiplicité



40

Les concepts objet à travers UML (22)

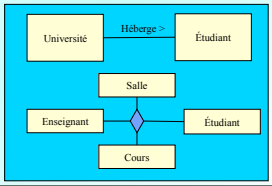
- **Les relations entre les classes**
  - Multiplicité
    - Chaque extrémité d'une association peut porter une indication de multiplicité (nombre d'occurrences) qui montre combien d'objets de la classe considérée peuvent être liés à un objet de l'autre classe.

1	Un et un seul
0..1	Zéro ou un
M..N	De M à N (entiers naturels)
*	De zéro à plusieurs
0..*	De zéro à plusieurs
1..*	D' un à plusieurs

41

Les concepts objet à travers UML (23)

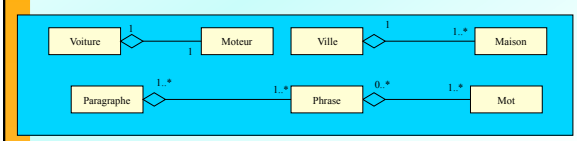
- **Les relations entre les classes**
  - Le nom de l'association peut être suivi d'une flèche de direction (triangle plein) qui précise le sens de lecture
  - Il existe des associations ternaires (ou d'ordre supérieur) qui sont représentées par un diamant relié à chacune des classes participant à la relation.



42

Les concepts objet à travers UML (24)

- **Les relations entre les classes**
  - L' agrégation
    - Est une forme particulière d' association
    - Permet de représenter des relations de type maître et esclaves, tout et parties ou composé et composants
    - Exprime un couplage plus fort entre classes que celui d' une association (où les classes associées restent relativement indépendantes l' une de l' autre)



43

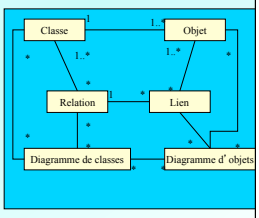
Les concepts objet à travers UML (25)

- **Les relations entre les classes**
  - L'agrégation
    - Les agrégations représentent des connexions bidirectionnelles dissymétriques
    - Une agrégation dénote une association de type ``Est composé de''.
      - se singularisent par le fait que les durées de vie des entités mises en relation sont fortement dépendantes: la durée de vie du ``composant'' est dépendante de la durée de vie du ``composite'' qui le contient. Si le composite est détruit, ses composants le sont avec.
    - Au sens mathématiques, l'agrégation est une relation transitive, non symétrique et réflexive
    - L'agrégat favorise la propagation des valeurs d'attribut et des opérations de l'agrégat vers les composants

44

Les concepts objet à travers UML (26)

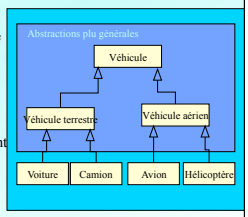
- **Les relations entre les classes**
  - Correspondance entres diagrammes de classes et diagrammes d'objets
    - Les diagrammes de classes et les diagrammes d'objets appartiennent à deux vues complémentaires d'un modèle
      - Un diagramme de classes montre une abstraction de la réalité, concentrée sur l'expression de la structure d'un point de vue générale
      - Un diagramme d'objets représente un cas particulier ou une situation concrète à un instant donné: il exprime à la fois la structure statique et un comportement



45

Les concepts objet à travers UML (27)

- **Les hiérarchies de classes**
  - Généralisation et spécialisation
    - La généralisation consiste à factoriser les éléments communs (attributs, opérations) d'un ensemble de classes dans une classe plus générale appelée super-classe
      - Les classes sont ordonnées selon une hiérarchie ; une super-classe est une abstraction de ses sous-classes
      - Un arbre (une hiérarchie) représentant une généralisation
        - se détermine en partant des feuilles (et non pas de la racine) car les feuilles appartiennent au monde réel alors que les niveaux supérieurs sont des abstractions construites pour ordonner et comprendre

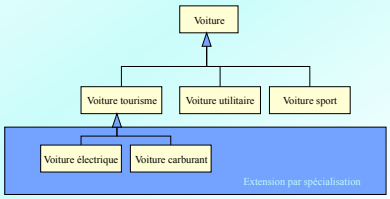


46

Les concepts objet à travers UML (28)

Les hiérarchies de classes

- La spécialisation permet de capturer les particularités d'un ensemble d'objets non discriminés par les classes déjà identifiées
  - Les nouvelles caractéristiques sont représentés par une nouvelle classe, sous-classe d'une des classes existantes



47

Les concepts objet à travers UML (29)

Les hiérarchies de classes

- La généralisation et la spécialisation sont deux point de vue antagonistes du concept de classification; elle expriment dans quel sens une hiérarchie de classe est exploitée
  - La généralisation est employée une fois que les éléments du domaine ont été identifiées afin de dégager une description détachée des solutions
  - La spécialisation est à la base de la programmation par extension et de la réutilisation
- La généralisation ne porte aucun nom particulier
  - Elle signifie toujours :
    - « est un » ou
    - « est une sorte de »
- La généralisation ne concerne que les classes, elle n'est pas instanciable en liens
- La généralisation ne porte aucune indication de multiplicité

48

Les concepts objet à travers UML (30)

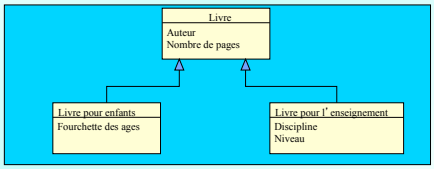
Les hiérarchies de classes

- La généralisation est une relation non réflexive : une classe ne peut pas dériver d'elle-même
- La généralisation est une relation non symétrique : si une classe B dérive d'une classe A, alors la classe A ne peut pas dériver de la classe B
- La généralisation est une relation transitive : si une classe C dérive d'une classe B qui dérive elle-même d'une classe A, alors C dérive également de A
- Les objets instances d'une classe donnée sont décrit par les propriétés caractéristiques de leur classe, mais également par les propriétés caractéristiques de toutes les classes parents de leur classe

49

Les concepts objet à travers UML (31)

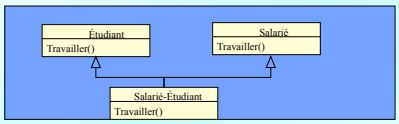
- **Les hiérarchies de classes**
  - Les propriétés d'une sous-classe englobe les propriétés de toutes ses super-classes
    - Ce qui est vrai pour un objet instance d'une super classe est vrai pour un objet instance d'une sous-classe



50

Les concepts objet à travers UML (32)

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



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

51

Les concepts objet à travers UML (33)

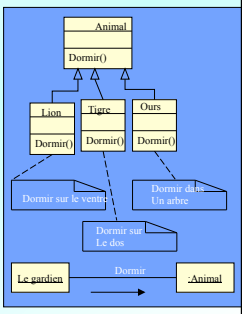
- **Les hiérarchies de classes**
  - L'héritage
    - Technique utilisée par les langages de programmation pour réaliser les hiérarchies de généralisation/spécialisation
  - Le polymorphisme
    - Le terme polymorphisme décrit la caractéristique d'un élément qui peut prendre plusieurs formes, comme l'eau qui se trouve à l'état solide, liquide ou gazeux
    - En informatique : le polymorphisme désigne le fait qu'un nom d'objet peut désigner des instances de classes différentes issues d'une même arborescence

52

Les concepts objet à travers UML (34)

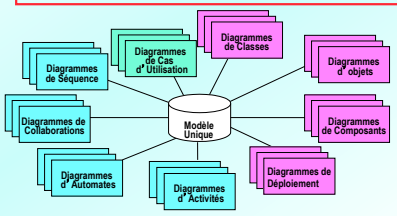
Les hiérarchies de classes

- Polymorphisme : notion de surcharge
  - Le polymorphisme signifie qu'un même opération peut se comporter différemment sur différents classes
    - L'opération « déplacer » agira de manières différentes sur un fichier, une fenêtre graphique ou un véhicule
  - Le polymorphisme signifie que les différentes méthodes d'une opération ont la même signature
    - Lorsque une opération est invoquée sur un objet, celui-ci « connaît » sa classe et par conséquent est capable d'invoquer automatiquement la méthode correspondante.



53

Partie II : Les diagrammes UML



54

Les vues statiques d'un système en UML

55

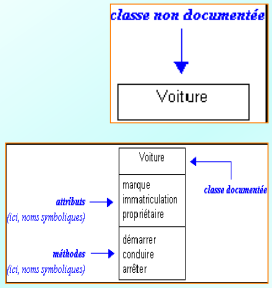
DIAGRAMME DE CLASSES (1)

- Sémantique
  - Un diagramme de classes est une collection d'éléments de modélisation statiques (classes, paquetages...), qui montre la structure d'un modèle.
  - Un diagramme de classes fait abstraction des aspects dynamiques et temporels
  - Pour un modèle complexe, plusieurs diagrammes de classes complémentaires doivent être construits.
    - On peut par exemple se focaliser sur :
      - les classes qui participent à un cas d'utilisation
      - les classes associées dans la réalisation d'un scénario précis,
      - les classes qui composent un paquetage,
  - Pour représenter un contexte précis, un diagramme de classes peut être instancié en diagrammes d'objets

56

DIAGRAMME DE CLASSES (2)

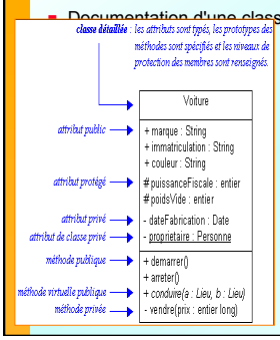
- Documentation d'une classe



57

DIAGRAMME DE CLASSES (3)

- Documentation d'une classe



Les attributs en ligne utilisent la notation suivante

Visibilité / nom : type multiplicité = défaut [propriétés et contraintes]

Visibilité ::= {+|-|# |~}

Multiplicité ::= [min..Max]

Visibilité : public, private, protected, package

/ : indique que l'attribut est dérivé

Nom : nom permettant de désigner l'attribut

Type : type de l'attribut

Multiplicité : spécifie le nombre de valeurs stockées dans l'attribut

Défaut : valeur par défaut de l'attribut

Propriétés : une collection de propriétés attachées aux attributs

Contraintes : une ou plusieurs restrictions imposées à un attribut

58

DIAGRAMME DE CLASSES (4)

- Documentation d'une classe : les attributs

Les attributs utilisent la notation suivante

Visibilité / nom : type multiplicité = défaut
{propriétés et contraintes}

Visibilité ::= {+|-|# |~}

Multiplicité ::= [min.. Max]

Visibilité : public, private, protected, package

/ : indique que l'attribut est dérivé

Nom : nom permettant de désigner l'attribut

Type : type de l'attribut

Multiplicité : spécifie le nombre de valeurs stockées dans l'attribut

Défaut : valeur par défaut de l'attribut

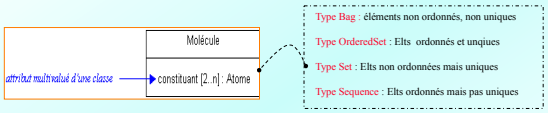
Propriétés : une collection de propriétés attachées aux attributs

Contraintes : une ou plusieurs restrictions imposées à un attribut

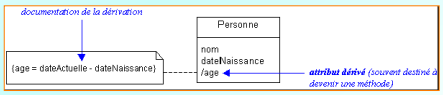
59

DIAGRAMME DE CLASSES (5)

- Attributs multivalués (attributs de type collection)



- Attributs dérivés

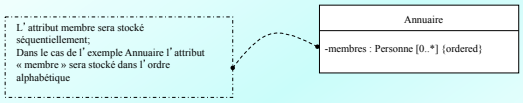


- Typiquement accessible en lecture seule (readOnly)

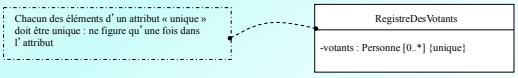
60

DIAGRAMME DE CLASSES (6)

- Attributs ordonnés



- Attributs uniques

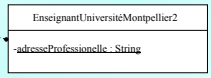


61

DIAGRAMME DE CLASSES (7)

- Attributs statiques

L'attribut est un attribut de classe; attribut partagés par l'ensemble des instances de la classe



- Propriétés des attributs
  - *readOnly*
    - spécifie que l'attribut ne peut pas être modifié, une fois celui-ci initialisé
  - *Union*
    - spécifie que le type attribut est une union des valeurs possibles pour celui-ci.
  - *subsets <nom-attribut>*
    - spécifie que le type d'attribut est un sous-ensemble de toutes les valeurs possibles d'un attribut donné.
  - *redefines <nom-attribut>*
    - spécifie que l'attribut joue le rôle d'alias pour un attribut donné
  - *Composite*
    - spécifie que l'attribut est impliqué dans une relation « tout-partie »

62

DIAGRAMME DE CLASSES (8)

- Documentation d'une classe : les opérations

Les opérations utilisent la notation suivante

Visibilité nom (paramètres) : type-retourné {propriétés}

Les paramètres sont écrits sous la forme suivante :

direction nom_paramètre : type [multiplicité]
= valeur_par_défaut {propriétés}

Visibilité : public, private, protected, package

Nom : nom permettant de désigner l'opération (généralement un verbe représentant une action)

Type-retourné : type de l'information retournée par l'opération, s'il y a une.

propriétés : spécifie les contraintes et les propriétés associées à l'opération.

Les éléments syntaxiques relatifs aux paramètres sont :

direction : indique la façon dont un paramètre est utilisé par une opération. Prends une des valeurs : IN, INOUT, OUT, RETURN

nom_paramètre : nom du paramètre.

type : type du paramètre

Multiplicité : nombre d'instances (du type spécifié) fournies par le paramètre

Valeur_par_défaut : spécifie la valeur par défaut de ce paramètre

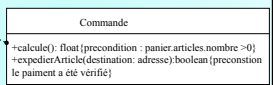
Propriétés : spécifie les propriétés liées au paramètre : ORDERED, READONLY, UNIQUE

63

DIAGRAMME DE CLASSES (9)

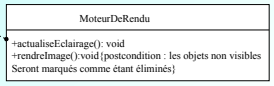
- Contraintes sur une opération
  - Permettent de définir la façon dont celle-ci interagit avec le reste du système
  - Condition a priori (precondition)

Les conditions a priori permettent d'identifier L'état du système requis pour invoquer une opération



  - Condition a posteriori (postcondition)

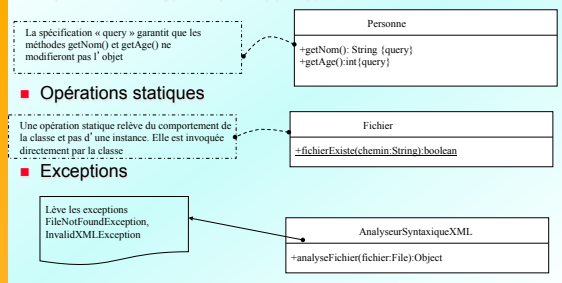
Les conditions a posteriori ont pour but de définir l'état d'un système, une fois l'opération exécutée.



64

DIAGRAMME DE CLASSES (10)

- Opérations de type requête (query)



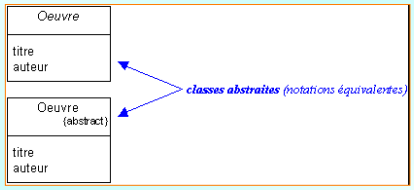
- Opérations statiques

- Exceptions

65

DIAGRAMME DE CLASSES (11)

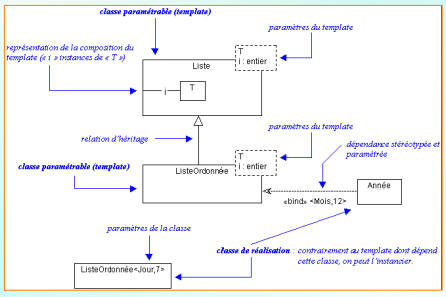
- Classe abstraite



66

DIAGRAMME DE CLASSES (12)

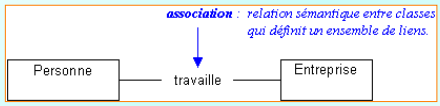
- Template (classe paramétrable):



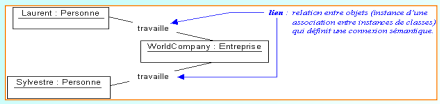
67

DIAGRAMME DE CLASSES (13)

- Associations entre classes
  - Une association exprime une connexion sémantique bidirectionnelle entre deux classes.



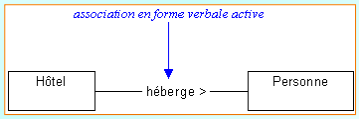
  - L'association est instanciable dans un diagramme d'objets ou de collaboration, sous forme de liens entre objets issus de classes associées.



68

DIAGRAMME DE CLASSES (14)

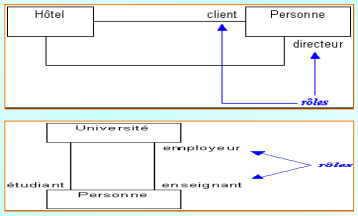
- Associations entre classes
  - Documentation d'une association et types d'associations
    - Association en forme verbale active : précise le sens de lecture principal d'une association.



69

DIAGRAMME DE CLASSES (15)

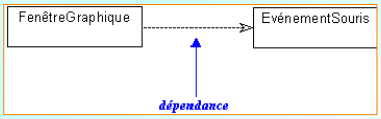
- Associations entre classes
  - Rôles
    - Spécifie la fonction d'une classe pour une association donnée



70

DIAGRAMME DE CLASSES (16)

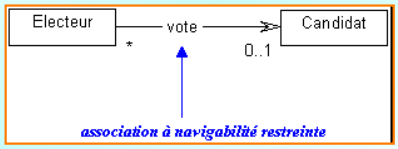
- Associations entre classes
  - Relation de dépendance
    - Relation d'utilisation unidirectionnelle et d'obsolescence
      - Une modification de l'élément dont on dépend, peut nécessiter une mise à jour de l'élément dépendant.



71

DIAGRAMME DE CLASSES (17)

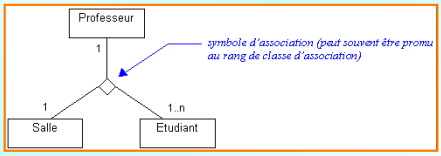
- Associations entre classes
  - Association à navigabilité restreinte
    - Par défaut, une association est navigable dans les deux sens.
    - La réduction de la portée peut aussi être exprimée dans un modèle pour indiquer que les instances d'une classe ne "connaissent" pas les instances d'une autre.



72

DIAGRAMME DE CLASSES (18)

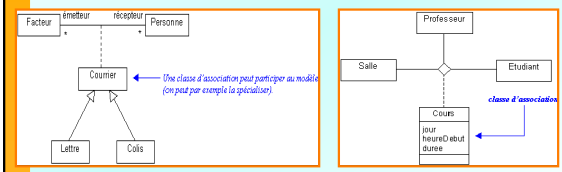
- Associations entre classes
  - Association n-aire
    - Il s'agit d'une association qui relie plus de deux classes



73

DIAGRAMME DE CLASSES (19)

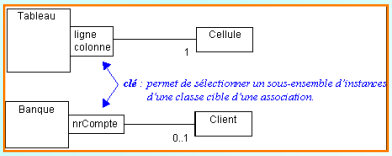
- Associations entre classes
  - Classe d'association
    - Il s'agit d'une classe qui réalise la navigation entre les instances d'autres classes



74

DIAGRAMME DE CLASSES (20)

- Associations entre classes
  - Qualification
    - Permet de sélectionner un sous-ensemble d'objets, parmi l'ensemble des objets qui participent à une association.
    - La restriction de l'association est définie par une clé, qui permet de sélectionner les objets ciblés.



75

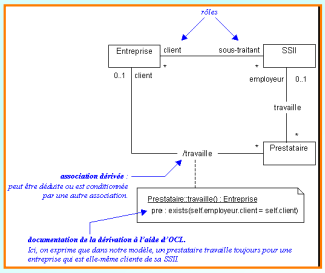
DIAGRAMME DE CLASSES (21)

- Associations entre classes
  - Association dérivée
    - Les associations dérivées sont des associations redondantes, qu'on peut déduire d'une autre association ou d'un ensemble d'autres associations.
    - Elles permettent d'indiquer des chemins de navigation "calculés", sur un diagramme de classes
    - Elles servent beaucoup à la compréhension de la navigation (comment joindre telles instances d'une classe à partir d'une autre).

76

DIAGRAMME DE CLASSES (22)

- Associations entre classes
  - Association dérivée
    - Exemple



77

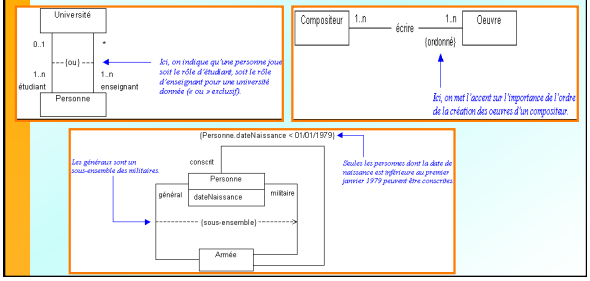
DIAGRAMME DE CLASSES (23)

- Associations entre classes
  - Contrainte sur une association
    - Les contraintes sont des expressions qui précisent le rôle ou la portée d'un élément de modélisation
      - Elles permettent d'étendre ou préciser sa sémantique
    - Sur une association, elles peuvent par exemple restreindre le nombre d'instances visées
    - Les contraintes peuvent s'exprimer en langage naturel
      - Graphiquement, il s'agit d'un texte encadré d'accolades.

78

DIAGRAMME DE CLASSES (24)

- Associations entre classes
  - Contrainte sur une association



79

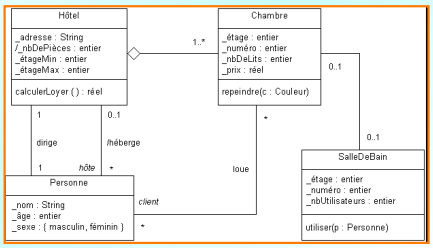
DIAGRAMME DE CLASSES (25)

- Associations entre classes
  - Contrainte sur une association : OCL
    - UML formalise l'expression des contraintes avec OCL (Object Constraint Language).
    - OCL est une contribution d'IBM à UML 1.1.
    - Ce langage formel est volontairement simple d'accès et possède une grammaire élémentaire (OCL peut être interprété par des outils).
    - Il représente un juste milieu, entre langage naturel et langage mathématique. OCL permet ainsi de limiter les ambiguïtés, tout en restant accessible.

80

DIAGRAMME DE CLASSES (26)

- Associations entre classes
  - Contrainte sur une association
    - Exemple



81

DIAGRAMME DE CLASSES (27)

- Associations entre classes
  - Contrainte sur une association
    - Exemple
      - *Le nombre de personnes par chambre doit être inférieur ou égal à* 3.
      - *l'étage de chaque chambre est compris entre le premier et le dernier étage de l'hôtel.*

```
– context Chambre inv: self.client <= 3
– context Hôtel inv:
        self.chambre→forAll(c : Chambre | c._étage<= self._étageMax and
        c._étage >= self._étageMin)
```

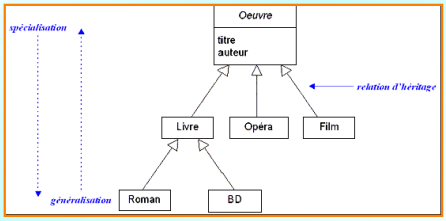
      - *Un hôtel Formule 1 ne contient jamais d'étage numéro 13*

```
context Chambre inv: self._étage <> 13
context SalleDeBain inv: self._étage <> 13
```

82

DIAGRAMME DE CLASSES(28)

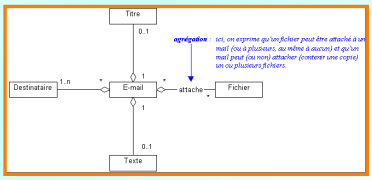
- Classification, spécialisation, généralisation et héritage
  - Exemple (voir transparents partie I)



83

DIAGRAMME DE CLASSES(29)

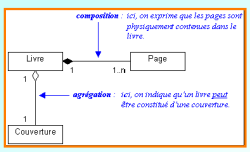
- Agrégation
  - L'agrégation est une association non symétrique, qui exprime un couplage fort et une relation de subordination.
  - Elle représente une relation de type "ensemble / élément".
  - UML ne définit pas ce qu'est une relation de type "ensemble / élément", mais il permet cependant d'exprimer cette vue subjective de manière explicite.



84

DIAGRAMME DE CLASSES (30)

- Composition
  - La composition est une agrégation forte (agrégation par valeur).
  - Les cycles de vies des éléments (les "composants") et de l'agrégat sont liés : si l'agrégat est détruit (ou copié), ses composants le sont aussi.
  - 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.



85

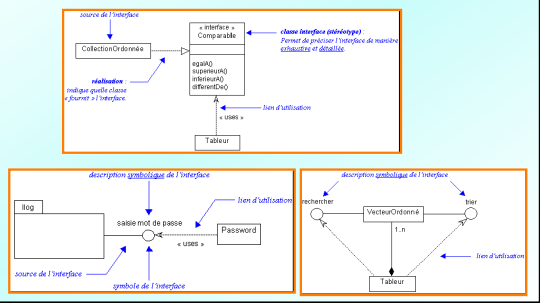
DIAGRAMME DE CLASSES(31)

- Interfaces
  - Une interface fournit une vue totale ou partielle d'un ensemble de services offerts par une classe, un paquetage ou un composant. Les éléments qui utilisent l'interface peuvent exploiter tout ou partie de l'interface
  - Dans un modèle UML, le symbole "interface" sert à identifier de manière explicite et symbolique les services offerts par un élément et l'utilisation qui en est faite par les autres éléments.

86

DIAGRAMME DE CLASSES(32)

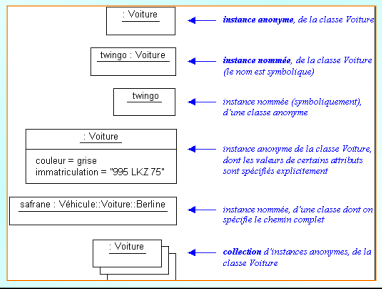
- Interfaces



87

DIAGRAMME D'OBJETS (1)

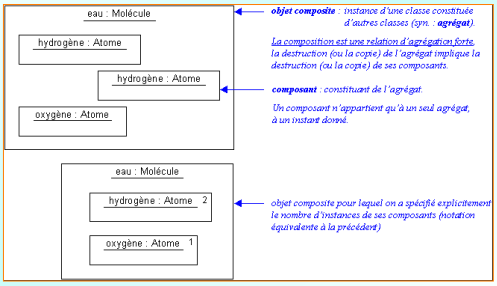
- Représentation des instances **(des objets)**
  - Un objet est une instance (avec un état précis) d'une classe



88

DIAGRAMME D'OBJETS (2)

- Objets composites



89

DIAGRAMME D'OBJETS (3)

- Sémantique
  - Un diagramme d'objets montre des objets et des liens entre ces objets
  - Les diagrammes d'objets s'utilisent pour montrer un contexte (avant ou après une interaction entre objets par exemple)