UNIVERSITÉ PARIS DESCARTES

BCO (M2104) :
Bases de la conception
« orientée objet »

DUT Informatique
Semestre 2

Mourad Ouziri
mourad.ouziri@parisdescartes.fr

IUT de Paris Descartes

Bases de la conception objet
Quelques informations générales

- Contrôles des connaissances
  - 2 DST : 1er DST en mi-semestre et 2ème DST en fin de semestre
  - TP noté en fin de semestre
  - Coefficients : DST1 x 2, DST2 x 2, TP noté x 1
- Bibliographie
  - Object-oriented Modeling and Design. *J. Rumbaugh Prentice Hall, 1991*
  - UML Resource Page. http://www.uml.org
  - UML 2 : Guide de référence. *Booch, Jacobson, Rumbaugh, ampusPress, 2004*
  - UML 2. *P. Roques, Eyrolles, 2008*

Bases de la conception objet

Aperçu de la matière

- Objectif : développer des logiciels de qualité
- Introduction à la conception d'applications à objets
- Analyse et conception d'application à objets
  - Le langage UML : cas d'utilisation, classes, séquence, états-transitions
  - Le langage de contraintes OCL – *Object Constraint Language*
- Implémentation des modèles de conception en Java : développement dirigé par les modèles
- Tests et validation : tests dirigés par les modèles

Introduction au

« développement logiciel »

Développement logiciel

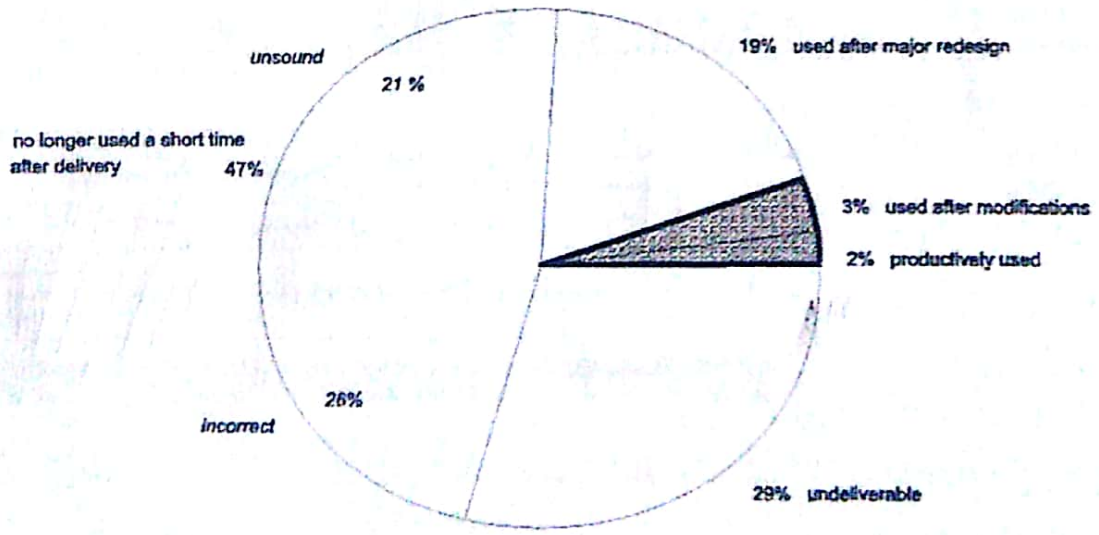
Motivation (1) – de la programmation au développement

- Développer un petit programme
  - Écrire un algorithme/programme : expression du *COMMENT*
  - Travail à l'échelle de l'individu : *Programming-in-the-small*
- Ne convient pas au développement logiciel !
  - Le *QUOI* n'est pas indiqué : que font les résultats ?
  - Pose problème pour le travail en équipe : faible lisibilité et peu transmissible !
  - Résultats produits qu'à la fin du développement !
  - Maintenance difficile et coûteuse !
- Production du logiciel : *Programming-in-the-large*

Développement logiciel

Motivation (2) – crise de l'industrie du logiciel

THE SOFTWARE CRISIS



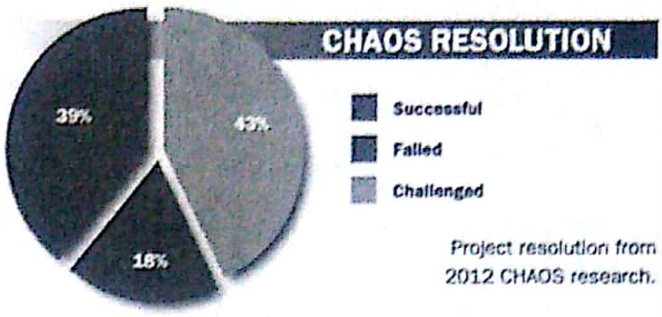
Source: Sage A. & Palmer, J., Software Systems Engineering. John Wiley & Sons, New York, 1990.

Développement logiciel

Motivation (3) – crise de l'industrie du logiciel

☞ Taux d'échec des projets de développement logiciel

The 2012 CHAOS results show another increase in project success rates, with 39% of all projects succeeding (delivered on time, on budget, with required features and functions); 43% were challenged (late, over budget, and/or with less than the required features and functions); and 18% failed (cancelled prior to completion or delivered and never used). These numbers represent an uptick in the success rates from the previous study, as well as a decrease in the number of failures. The low point in the last five study periods was 2004, in which only 29% of the projects were successful. This year's results represent a high watermark for success rates in the history of CHAOS research.



Project resolution from 2012 CHAOS research.

RESOLUTION

	2004	2006	2008	2010	2012
Successful	29%	35%	32%	37%	39%
Failed	18%	19%	24%	21%	18%
Challenged	53%	46%	44%	42%	43%

Project resolution results from CHAOS research for years 2004 to 2012.

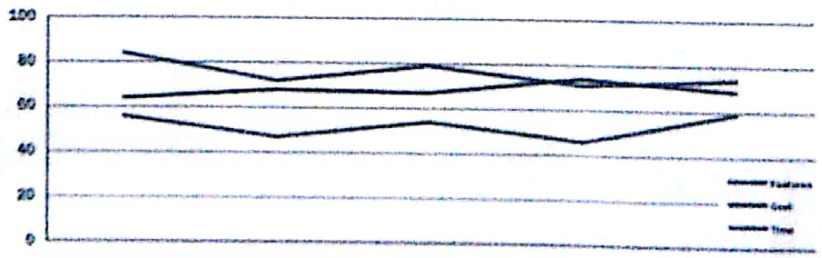
Développement logiciel

Motivation (4) – crise de l'industrie du logiciel

☞ Dépassement de temps et budget et non complétude fonctionnelle

OVERRUNS AND FEATURES

Time and cost overruns, plus percentage of features delivered from CHAOS research for the years 2004 to 2012.



	2004	2006	2008	2010	2012
TIME	84%	72%	79%	71%	74%
COST	56%	47%	54%	46%	59%
FEATURES	64%	68%	67%	74%	69%

Développement logiciel

Motivation (5) – crise de l' industrie du logiciel

- Quelques raisons d' échec des logiciels
  - **Mauvaise compréhension des besoins des utilisateurs**
  - **Découverte tardive des erreurs/problèmes**
  - Logiciel difficile à maintenir ou à faire évoluer
  - Des modules qui ne fonctionnent pas ensemble
  - Problème de coordination dans les équipes
  - Procédures de tests coûteuses
  - ...

Développement logiciel

Définitions

- Nécessité de surmonter la crise de l' industrie du logiciel
- Naissance du domaine du Génie Logiciel (1968)
- Génie logiciel – *Software Engineering*

  Le génie logiciel désigne l'ensemble des méthodes, des techniques et outils concourant à la **production de logiciels de qualité**
- Logiciel – *Software*

  Ensemble des programmes, procédés et règles (et éventuellement de la documentation) relatifs au fonctionnement d'un ensemble de traitements de l'information *(IEEE Std 729)*

Développement logiciel

Qualité du logiciel (1)

- Quelques facteurs de qualité *(McCall – US Air Force, FURPS – HP)*
  - **Conformité fonctionnelle** : satisfaire les besoins fonctionnels des utilisateurs
  - **Maintenabilité** : minimiser l'effort pour localiser et corriger les erreurs
  - **Evolutivité** : minimiser l'effort nécessaire pour le modifier par suite d'évolution des spécifications
  - **Maniabilité** : minimiser l'effort nécessaire pour son apprentissage et son utilisation
  - **Portabilité** : facilité de le transférer d'une plateforme/environnement à un autre
  - **Efficacité** : se limiter à l'utilisation des ressources strictement nécessaires à l'accomplissement de ses fonctions
- CISQ – Consortium for IT Software Quality (créé en 2009 aux USA) pour établir un standard mondial de la qualité du logiciel

Développement logiciel

Qualité du logiciel (2)

- Que faire pour obtenir cette qualité du logiciel ?
  - **Analyser la situation réelle :** étudier la situation et les spécifications fonctionnelles
  - **Modéliser :** représenter le résultat d'analyse par des schémas formels (et visuels)
  - **Valider avant de programmer :** valider des modèles coûte moins cher que de valider du code
  - **Structurer** : définir une architecture de maintenabilité pour le logiciel
  - Suivre **une démarche** de développement : définir les étapes à suivre

Développement logiciel
Modélisation

- Modèle
  - Représentation abstraite et facilement compréhensible de la réalité
  - Schématisation de la solution avec des symboles (graphiques) intuitifs
  - Vue subjective mais pertinente de la réalité
- Pourquoi modéliser ?
  - Comprendre les besoins ainsi que le monde réel avant de réaliser le logiciel
  - Concevoir le logiciel indépendamment des langages de programmation objets
  - Faciliter la communication entre les contractants du projet
  - Répartir efficacement les tâches de développement
  - Réduction des coûts et des délais
  - Préparer la documentation

- Quel langage ? UML : langage de modélisation d'applications objets

Développement logiciel
Étapes de développement et diagrammes UML (1)

- Modélisation des besoins fonctionnels des utilisateurs
  - Les modèles exprimant les besoins des utilisateurs sont une description précise de *ce que le client demande*, et non de comment on peut les réaliser
  - Compréhensible par les experts du métier qui ne sont pas informaticiens
  - Ne comporte aucune décision d'implantation
- Diagramme UML correspondant
  - **Diagramme de cas d'utilisation – DCU**

    + Fiches descriptives des CU

} 1 cours

Développement logiciel

Étapes de développement et diagrammes UML (2)

- Modéliser l'application en *Objet*
  - Identifier les traitements et les données nécessaires au fonctionnement de l'application
  - Analyser le comportement du logiciel face à un besoin
  - Comprendre et tester le comportement des objets constituant le logiciel
- Diagrammes UML correspondants
  - **Diagramme de classes – DC** 3 cours
  - **Diagramme de séquences – DS** 2 cours
  - **Diagramme d'états-transitions – DE** 1 cours

En conclusion

- Modéliser (schématiser) avant de réaliser (programmer)
- Le modèle objet est intuitif
- A noter que la programmation ne représente qu'une infime part dans le développement logiciel !

Le langage UML

Langage de modélisation d'applications à objets

UML

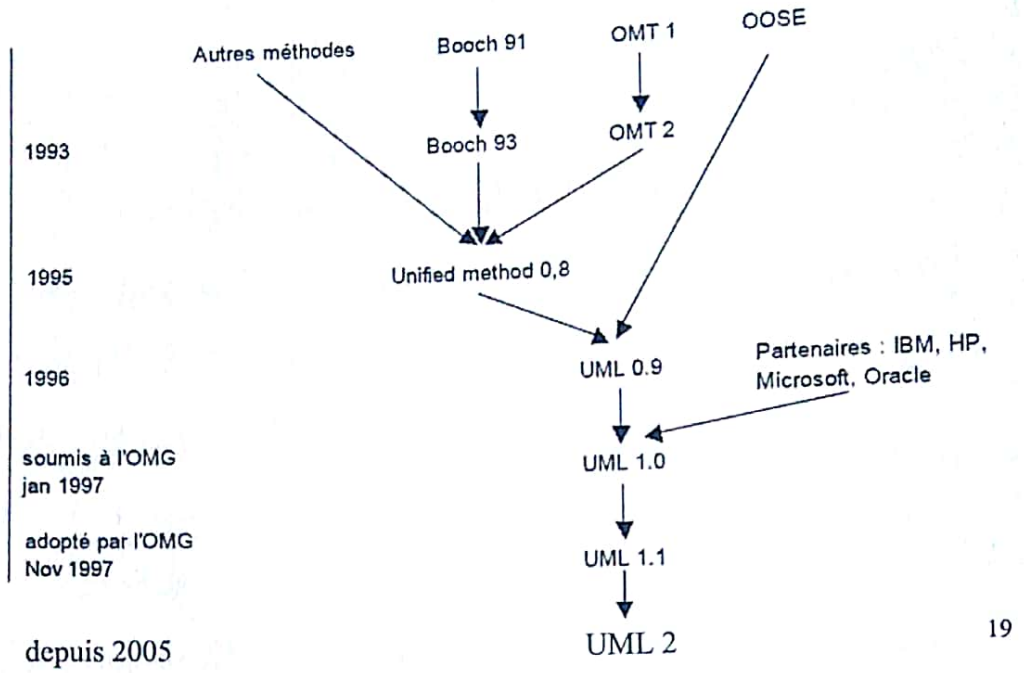
Quelques caractéristiques

☞ UML : *Unified Modeling Language*

- Norme de l' OMG (Object Managment Group) depuis 1996
- Langage de modélisation objet et pas une méthode
- Universel : indépendant des méthodes et langages de programmation
- Support de discussion, de communication et de répartition des tâches
- Intervient à toutes les étapes du développement logiciel : les diagrammes UML
- Diagramme UML : notation graphique représentant un aspect précis du logiciel

UML

Genèse



UML

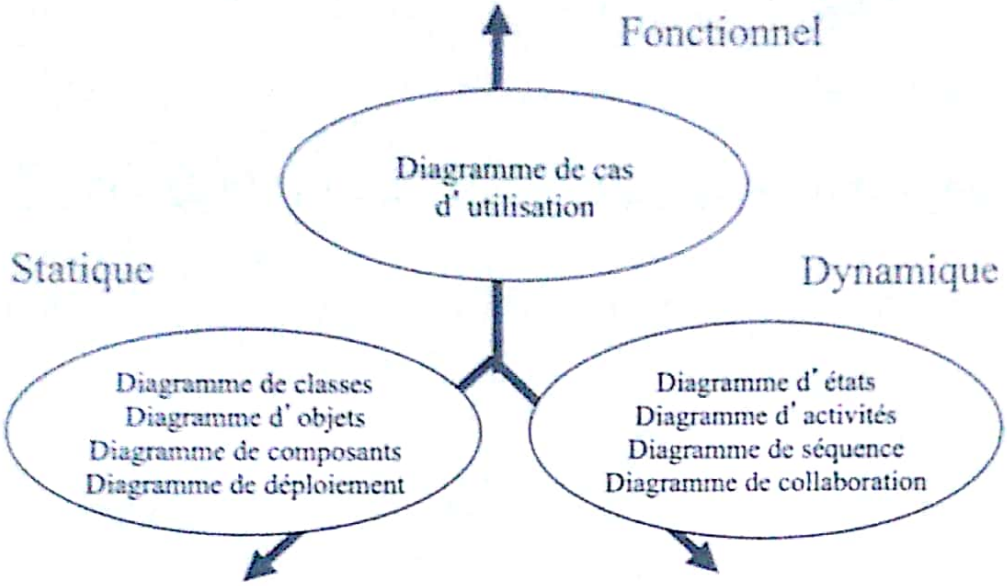
Objectif

- Objectifs de UML
  - Bien formaliser les besoins utilisateurs et montrer les frontières du projet de développement logiciel
  - Illustrer les réalisations (déroulements) des fonctions principales du logiciel
  - Représenter la structure statique (squelette) du logiciel « orienté objet »
  - Modéliser la dynamique et le comportement des objets
  - Révéler l'implantation physique de l'architecture
- Un langage compréhensible par l'homme et la machine
  - Conception (intellectuelle) humaine et manipulation par la machine
  - Atelier de génie logiciel – AGL
  - Ce qui permet la génération automatique du code

UML

Les diagrammes UML

☞ UML : les 9 diagrammes



UML

Quelques diagrammes

☞ UML : les quatre diagrammes que nous étudierons

- Diagrammes des cas d'utilisation (DCU) : fonctions de l'application du point de vue des utilisateurs
- Diagrammes de classes (DC) : structure statique d'une application objet
- Diagrammes de séquence (DS) : modèle dynamique représentant les interactions entre les objets d'une application
- Diagrammes d'états-transitions (DE) : comportement des objets d'une classe en terme d'états et de transitions valides

Modélisation fonctionnelle :

« Diagramme de Cas d' Utilisation UML »

UML

DCU – Diagramme de Cas d' Utilisation

☞ Diagramme de cas d' utilisation

- But : recueillir, comprendre et structurer les besoins utilisateurs
- DCU : Représentation graphique, simple et compréhensible des besoins utilisateurs
- Définir les frontières du système à modéliser (préciser le but à atteindre)
- Identifier les besoins fonctionnels requis par les utilisateurs potentiels du logiciel
- Fournir un document de discussion entre la maîtrise d' ouvrage et la maîtrise d' œuvre

UML

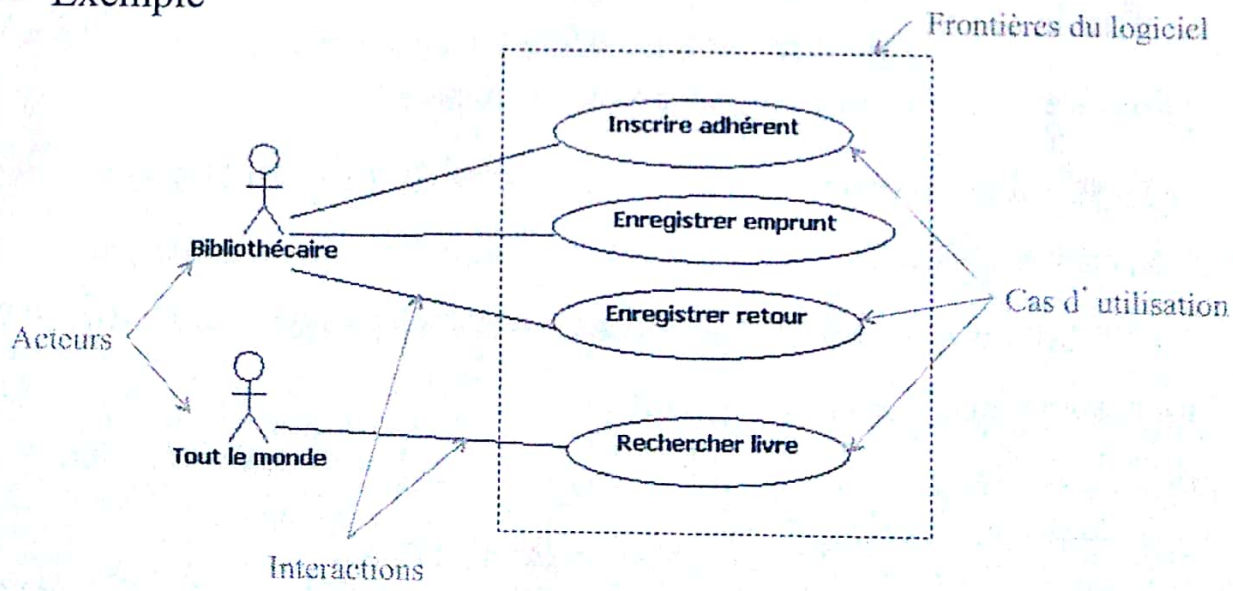
DCU – Diagramme de Cas d' Utilisation

- DCU : **Qui** utilisera le logiciel et **Quels** sont ses besoins fonctionnels ?
  - **Qui** : utilisateurs du logiciel analysé ayant une interaction direct avec lui
  - **Quoi** : fonctions du logiciel analysé
- Éléments de modélisation du diagramme de cas d' utilisation
  - **Acteur** (Qui?) : utilisateur du logiciel
  - **Cas d' utilisation** (Quoi?) : fonctionnalité utile (requise) pour au moins un acteur

UML

DCU – Diagramme de Cas d' Utilisation

- Exemple



UML

DCU – Diagramme de Cas d'Utilisation

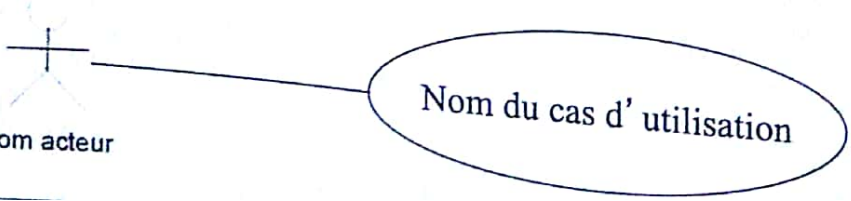
- Acteur
  - Toute entité externe au logiciel et qui interagit directement avec lui (qui l'utilise)
  - Représente un rôle joué par plusieurs personnes (ou systèmes)
  - La même personne/entité physique peut être représentée par plusieurs acteurs en fonction des rôles qu'elle joue
  - Un acteur n'est pas nécessairement une personne physique : il peut être un service administratif, une société, un système informatique, …
- Représentation de l' acteur

Nom acteur

UML

DCU – Diagramme de Cas d'Utilisation

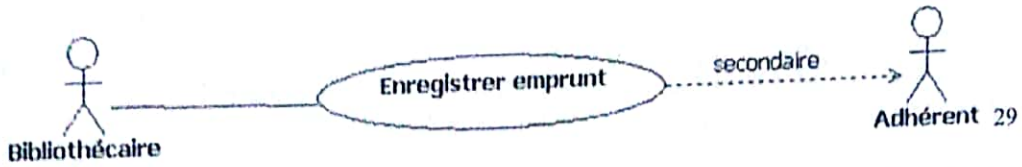
- Cas d' utilisation
  - Comportement du logiciel du point de vue des Acteurs
  - Fonction métier (besoin fonctionnel) requise par au moins un acteur
  - Une fonction métier déclenchée par un acteur (directement ou indirectement)
  - Il modélise à la fois des activités (fonctions) et des communications
- Représentation d' un cas d' utilisation



UML

DCU – Diagramme de Cas d'Utilisation

- Deux catégories d'acteurs pour un CU
  - Acteur principal : toute entité qui utilisent directement un cas d' utilisation
  - Acteur secondaire : une entité qui bénéficie d' un résultat produit par le CU ou celle sollicitée par le logiciel
- Exemple
  - Bibliothécaire : voudrait enregistrer les emprunts pour les adhérents
  - Adhérent : informé de la date limite de remise



29

UML

DCU – Identifier les acteurs

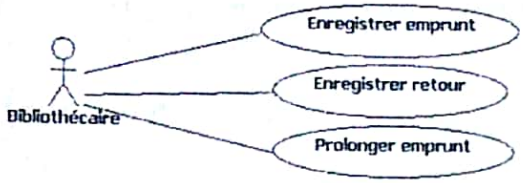
- Qui utilisera directement les fonctionnalités du logiciel ?
- Qui a besoin du support du logiciel pour effectuer ses tâches quotidiennes ?
- Avec quels autres systèmes le logiciel interagit-il ?
- Qui a un intérêt pour les résultats produits ?
- …

30

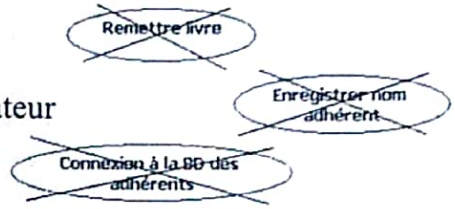
UML

DCU – Cas d' utilisation

- Rappel : un modèle (DCU) est un schéma subjectif !
- Les CU peuvent être conçus à différents niveaux de détail !



- Quelques caractéristiques des CU:
  - Informatisable
  - Une finalité pour l' utilisateur
  - Service/fonction métier



DCU

Fiches descriptives de Cas d' Utilisation

UML

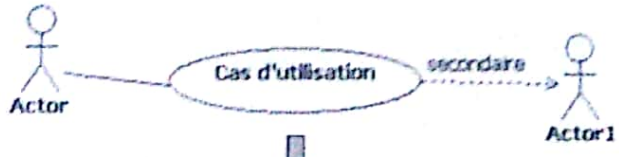
DCU – Fiche descriptive des CU

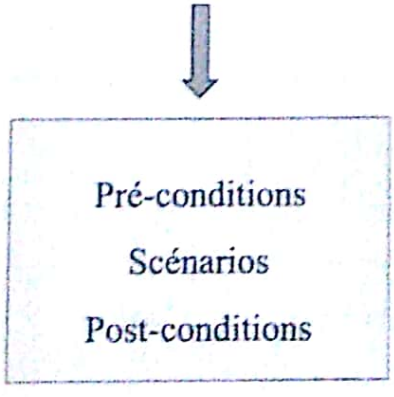
- Limites du DCU
  - Le DCU n'est qu'un sommaire des besoins fonctionnels
  - Nom des CU peu explicatif de la fonction → fonctions pas assez précises
- D'où la nécessité de détailler les CU par une description textuelle
- Objectifs des fiches descriptives des CU
  - Décrire comment se déroule le CU/fonction sur le plan métier
  - Donner les scénarios principaux des CU
  - Définir les informations échangées entre l'application et les utilisateurs
  - Préparer les cas de tests

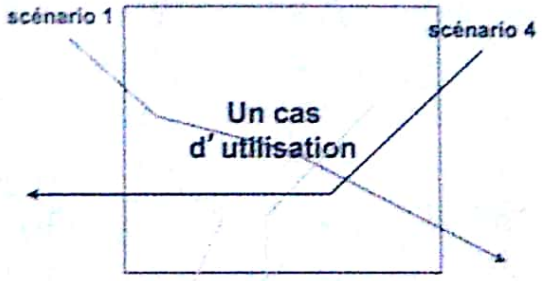
UML

DCU – Fiche descriptive des CU

- Contenu d'une fiche descriptive d' un CU







UML

DCU – Fiche descriptive des CU

☞ Contenu d'une fiche descriptive d' un CU

1. Identification du CU

Nom du CU, date de création, résumé , version, acteurs impliqués

2. Séquences : scénarios possibles du CU

2.1 Pré-conditions : conditions devant être satisfaites pour que le CU puisse se dérouler

2.2 **Enchaînements** : scénarios du CU

Nominal (obligatoire) : scénario le plus probable (favorable)

Alternatifs (optionnel) : scénarios alternatifs au scénario nominal

Exceptionnels (optionnel) : traitement des exceptions (erreurs)

2.3 Post-conditions : conditions devant être satisfaites à la fin du CU et ce quelque soit le scénario

UML

Fiche descriptive – Exemple

☞ Fiche descriptive du CU « *Valider la création d'un compte bancaire*»

Nom du cas : Valider la création d' un compte bancaire

Acteur principal : Directeur d' agence

Date : 15/01/2014 **Version** : 1.0

Pré-conditions : le directeur est authentifié

Enchaînement nominal

Directeur d' agence	CU – Valider des comptes
1. Demander à valider des comptes	2. Afficher la liste des comptes en attente de validation
3. Choisir un compte	4. Afficher les informations du compte choisi (propriétaire, justif. de ressources, date de création)
5. Valider le compte	6. Enregistrer la validation + date de validation
	7. Retour au point 2

Post-conditions : Les comptes validés (ou annulés) auparavant ne changent pas d' état

UML

Fiche descriptive – Exemple

Enchaînement alternatif 1 – Non validation d'un compte

Le cas continue au point 5

5. Annuler la création du compte et saisir le motif
6. Enregistrer la non validation du compte et le motif

Enchaînement alternatif 2 – Aucun compte en attente de validation

Le cas continue au point 2

2. Afficher le message « Pas de comptes à valider »

Enchaînement d'exception – Motif de non validation de compte non renseigné

Le cas continue au point 6 du cas alternatif 1

6. Afficher le message « Motif de non validation obligatoire ! »
7. Retour au point 4

UML

Fiche descriptive – Exemple

- Un deuxième scénario peu pertinent !

Directeur d'agence	CU – Valider des comptes
1. Demander à valider des comptes	2. Demander à saisir le numéro du compte à valider
3. Saisir un numéro de compte	4. Afficher les informations du compte (propriétaire, justif. de ressources, date de création)
5. Valider le compte	6. Enregistrer la validation + date de validation
	7. Retour au point 2

- Scénario alternatif : « Le compte saisi n'existe pas ou a déjà été validé »

4. Afficher le message « Compte inconnu ou déjà validé ! »
5. Retour au point 2

UML

Fiche descriptive – Exemple

☞ Un scénario peu pertinent !

Directeur d' agence	CU – Valider des comptes
1. Demander à valider des comptes	2. Afficher la liste des (tous !) comptes bancaires
3. Choisir un compte	4. Afficher les informations du compte (propriétaire, justif. de ressources, date de création)
5. Valider le compte	6. Enregistrer la validation + date de validation
	7. Retour au point 2

☞ Scénario alternatif : « Le compte choisi a déjà été validé »

4. Afficher le message « Compte déjà validé ! »

5. Retour au point 2

UML

DCU – Fiche descriptive des CU

☞ Avantages

– Décrire les CU : meilleure compréhension des fonctions du logiciel

– Faciliter la conception des diagrammes UML (notamment les diagrammes de séquence et de collaboration)

– Préparer les cas de tests

– Préparer la documentation finale du logiciel

DCU

Relations entre Cas d' Utilisation

UML

DCU – Relations dans le DCU

- Modulariser pour mieux **maintenir**
- Modulariser pour **réutiliser**
  - Constat : comportements (traitements) communs à plusieurs CU
  - Conséquence : redondance du travail de développement
  - Modulariser pour éviter les développements redondants
- Types de relations dans un DCU
  - 3 types de relation en CU : Inclusion, Extension et Héritage
  - 1 type de relation entre acteurs : Héritage

UML

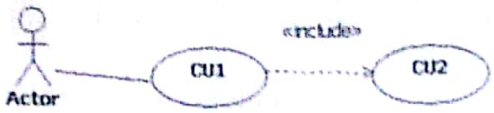
DCU – Relations dans le DCU

☞ **Inclusion de CU**

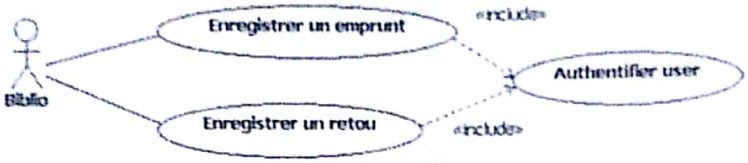
– Sémantique : CU1 *inclut* CU2 ssi CU1 fait appel à CU2 dans tous les scénarios possibles

– CU1 ne peut se terminer avec succès si CU2 ne s'est pas terminé avec succès !

– Représentation graphique



– Exemple : la bibliothécaire doit être authentifiée pour enregistrer un emprunt ou un retour



UML

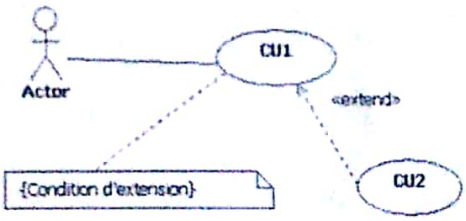
DCU – Relations dans le DCU

☞ **Extension de CU**

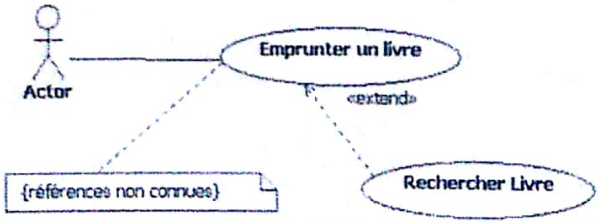
– Sémantique : CU1 *étend* CU2 ssi CU1 fait appel à CU2 dans certains scénarios

– L'extension est conditionnelle

– Représentation graphique



– Exemple : lorsqu'on emprunte un livre, on peut le rechercher si on ne connait pas ses références exactes



UML

DCU – Relations dans le DCU

☞ **Héritage entre CU**

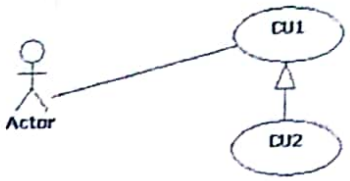
– Sémantique : CU2 *hérite de* CU1 signifie que :

CU2 est une spécialisation de CU1 (CU1 et une généralisation de CU2)

CU1 spécialise CU2 en y rajoutant des spécificités

– Traitements de même nature et substituables

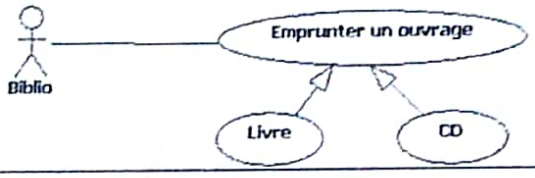
– Représentation graphique



– Exemple

On peut emprunter soit

des livres soit des CD



UML

DCU – Relations dans le DCU

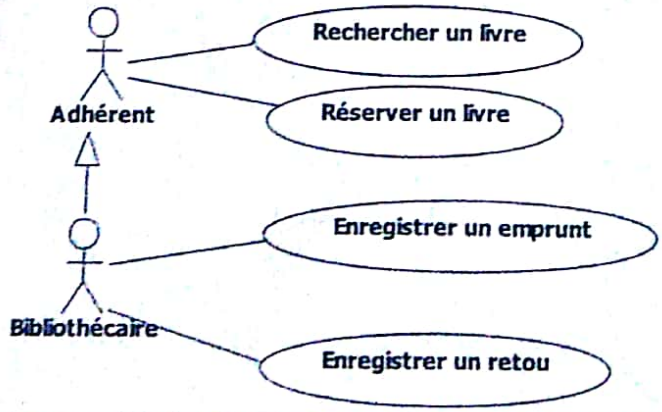
☞ **Héritage entre acteurs**

– Sémantique :

Acteur1 *hérite de* Acteur2 si Acteur1 utilise tous les CU de Acteur2

– Exemple

La bibliothécaire utilise (a besoin de) tous les CU de l'adhérent



UML

DCU – Résumé

- Le DCU exprime les attentes des utilisateurs
  - Fonctions métiers exprimées du point de vue de l'utilisateur
- Il doit être lisible
  - C'est un support de discussion permettant de valider les besoins fonctionnels
- Il peut inclure des relations entre CUs
  - Ne pas trop détailler les CU en voulant utiliser systématiquement ces relations
  - Attention ! Ne pas confondre les relations d'héritage et d'extension
- Le DCU est indépendant du modèle *Objet*
  - Peut être utilisé dans n'importe quel processus de développement

UNIVERSITÉ PARIS DESCARTES

Bases de la conception « orientée objet »

DUT Informatique
Semestre 2

Mourad Ouziri
mourad.ouziri@parisdescartes.fr

IUT de Paris Descartes

Conception objet :

« Modélisation de classes avec UML »

Objectif : savoir analyser une situation réelle simple et identifier de manière pertinente les classes de l'application objet

Conception objet
Qualité

- Rappel sur la qualité
  - **Fonctionnalité** : réponse aux spécifications fonctionnelles
  - **Maintenabilité** : coût de corrections et d'évolutions
  - Performance : ressources requises pour fonctionner
  - Robustesse : capacité à fonctionner en cas d'anomalies (manque de ressources)
  - …
- Phases de développement
  - Phase d'analyse : analyse **fonctionnelle** de l'application
  - Phase de conception : mise en place de la structure (architecture) de l'application (pour une meilleure **maintenabilité** *approfondie en semestre 3*)

Conception objet
Modèle de classes

- DCU
  - Exprime les services fonctionnels que doit offrir le logiciel à ses utilisateurs
  - Ne donne aucune indication du comment les implémenter
- Modèle de classes – définition
  - Exprime la structure des éléments (objets) constituant l'application
  - Exprime les classes (ou types) d'objets de l'application
  - Indique les classes, attributs et (signature des) méthodes constituant le logiciel
  - Structure statique : n'indique pas comment les méthodes sont implémentées

Conception objet
Définitions

- Application objet
  - Une application objet est une société d'objets (ensemble d'objets communicants)
  - Chaque objet réalise des tâches élémentaires qui relèvent de ses compétences (**rôle**
  - Les objets de l'application coopèrent (**interagissent**) pour réaliser les fonctions de l'application

Conception objet
Définitions

- Conception objet
  - Structurer/décomposer une situation réelle en objets
  - Modélisation (par des schémas visuels) d'une situation réelle avec des objets
  - Une formule simple, un objet du monde réel est représenté par un objet informatique (résidant dans la mémoire vivre de l'application)
- Abstraction : un élément fondamental dans la conception
  - Faire face à la complexité de la situation réelle
  - Identifier les **propriétés externes** des objets les distinguant les uns des autres
  - Regrouper les objets ayant les mêmes propriétés en classes

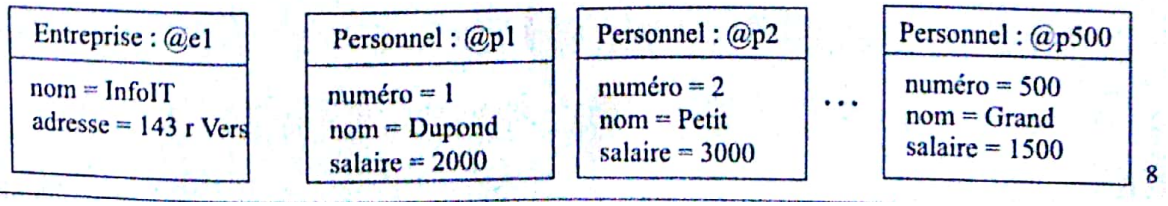
Conception objet
Notion d'objet

- Objet
  - Un objet peut être toute entité (matérielle ou immatérielle) ayant un rôle dans la situation modélisée
  - Objet = < Identité, attributs, comportement >
  - Un objet forme un tout
- Identité d'un objet
  - Identifiant de l'objet (peut être son adresse mémoire !)
  - Permet de s'adresser à un objet
  - Elle n'est pas définie par le concepteur
  - Différente de l'identifiant (clé) de gestion, définie par le concepteur

Conception objet
Notion d'objet

- Attributs d'objet
  - Caractéristiques (non visibles) décrivant les objets
  - Possèdent des valeurs propres (privées) à chaque objet
  - Déterminent l'état d'un objet
  - Peuvent évoluer dans le temps (ou pas)
- Exemple :
  - L'entreprise *InfoIT* sise au *143 r Versailles* emploie 500 personnels décrits par leurs numéro, nom et salaire

@x : adresse de l'objet



Conception objet

Notion d'objet

- Comportement d'un objet
  - Opérations (traitements) que peut accomplir un objet
  - Rôles (responsabilités) attribués à l'objet : actions qu'il peut accomplir
  - Part de contribution de l'objet dans l'accomplissement des fonctions de l'application
  - Agit, généralement, sur les attributs de l'objet et il en dépend
- Exemple (suite) : l'entreprise *IfoITSA*
  - Rôles attribués à un objet Personnel :

    Conserver le numéro, nom et salaire du personnel qu'il représente

    Fournir le numéro, nom et salaire du personnel qu'il représente

    Modifier le salaire du personnel

Conception objet

Notion d'objet

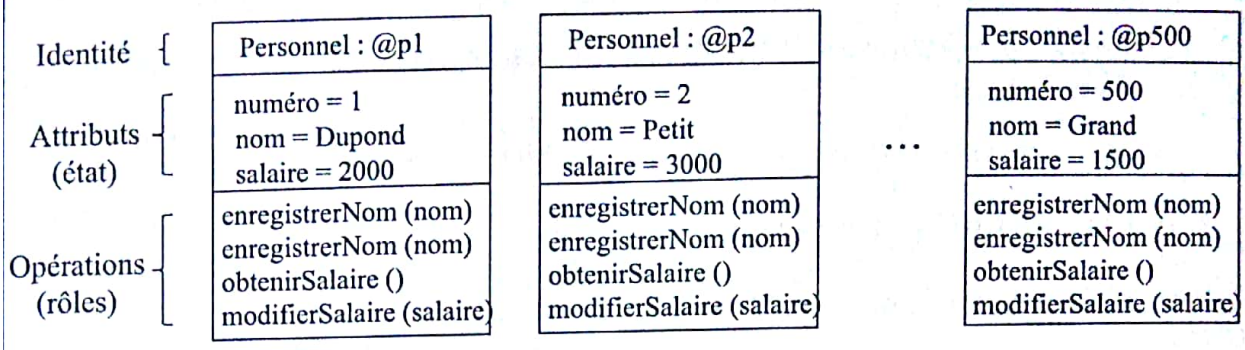
- Exemple : comportement d'un objet *Personnel*
  - Quel est le salaire de *Dupond* ?
  - A quel(s) objet(s) s'adresser ? A qui a-t-on attribué cette responsabilité (rôle) !
  - A celui qui détient l'information !
  - C'est l'objet *Personnel* identifié par *@p1*
- Exemple : comportement des objets *Personnel*
  - Quel est le salaire d'un personnel ?
  - S'adresser à l'objet *Personnel* qui représente le personnel en question
  - C'est l'objet *Personnel* identifié par *@p1*
- Tous les objets *Personnel* possèdent les mêmes responsabilités

Conception objet

Notion d'objet

☞ Exemple (suite) :

– Tous les objets Personnel possèdent les mêmes opérations (rôles) :

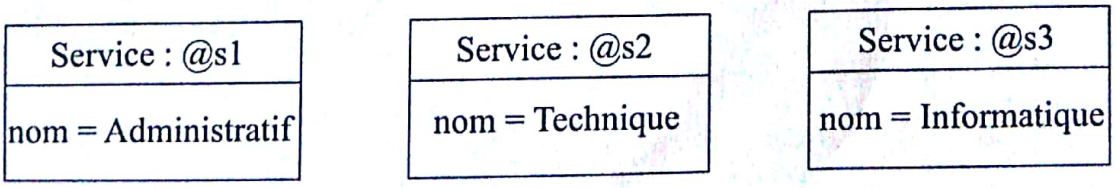


Conception objet

Notion d'objet

☞ Exemple (suite) : l'entreprise *IfoITSA* …

– L'entreprise est organisée en 3 services, chaque service est décrit par son nom



– Rôles attribués à un objet Service :

Conserver le nom du service qu'il représente

Fournir le nom du service information à la demande

Modifier le nom du service représenté

Conception objet

Notion d'objet

- Exemple : quels attributs et comportements ?!





Conception objet

Notion de classe

- Une classe : plusieurs objets d'une même famille, ayant des propriétés similaires (attributs et comportement)

  Les personnels Dupond, Petit, … sont décrits par leurs numéro, nom et salaire

  Les livres :

La Gloire De Mon Père de M. Pagnol paru en 1958

Le seigneur des Anneaux de JRR Tolkien paru en 1954

  sont caractérisés par les mêmes attributs : titre, auteur et date de parution

  Le résultat de conception sera le même pour l'ensemble de ces objets

- Conception : étude des **propriétés** des objets en faisant **abstraction** des valeurs de leurs attributs les différenciant

Conception objet

Notion de classe

- Classe
  - Structure (abstrait) décrivant les propriétés des objets d'une même famille
  - Type d'objets composé d'attributs et d'opérations
  - Permet de réduire la complexité de la situation étudiée
  - Moule d'objets : permet de créer des objets similaires (instance de classe)
- Attributs
  - Caractéristiques décrivant les objets de la classe
  - Définissent l'état d'un objet de la classe
- Opérations : compétences/rôles
  - Fonctions (traitements) pouvant être réalisées par les objets de la classe
  - Tous les objets de la classes fournissent (peuvent exécuter) ces opérations
- Propriétés d'une classe d'objet : définies par attributs et opérations

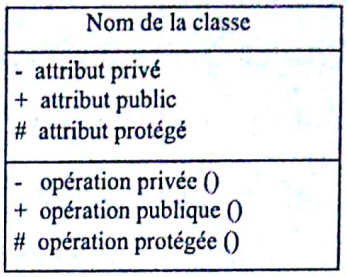
Conception objet

Modélisation de classe avec UML

- Eléments de modélisation en UML
  - Classes : attributs, méthodes (opérations)
  - Relations : association, agrégation, composition, multiplicités et rôles
  - Héritage
  - Utilisation (dépendance)
  - Classes abstraites et Interface
  - Réalisation/implémentation

Conception objet

Modélisation de classe avec UML

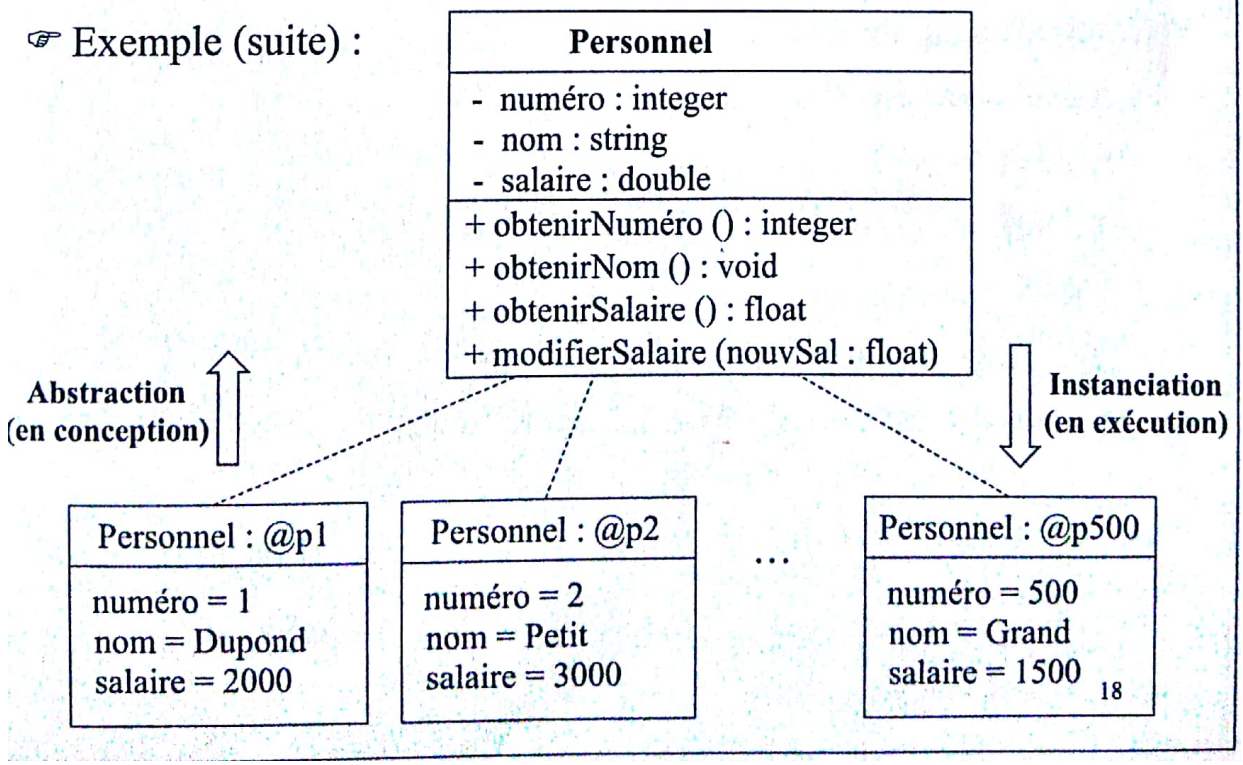


- Classe
  - Structure commune à plusieurs objets
- Encapsulation
  - Les attributs définissent l'état interne (non visible) des objets de la classe
  - Accessibles uniquement par les opérations de l'objet, pas d'ailleurs !
- Objet : instance d'une classe
  - Classe (moule d'objets) : peut servir à créer des objets
  - Objet : obtenu par instanciation (attribuer des valeurs aux attributs) de sa classe
  - Identifié par un *oid* (Object Identifier)

Conception objet

Modélisation de classe avec UML

- Exemple (suite) :



Conception objet

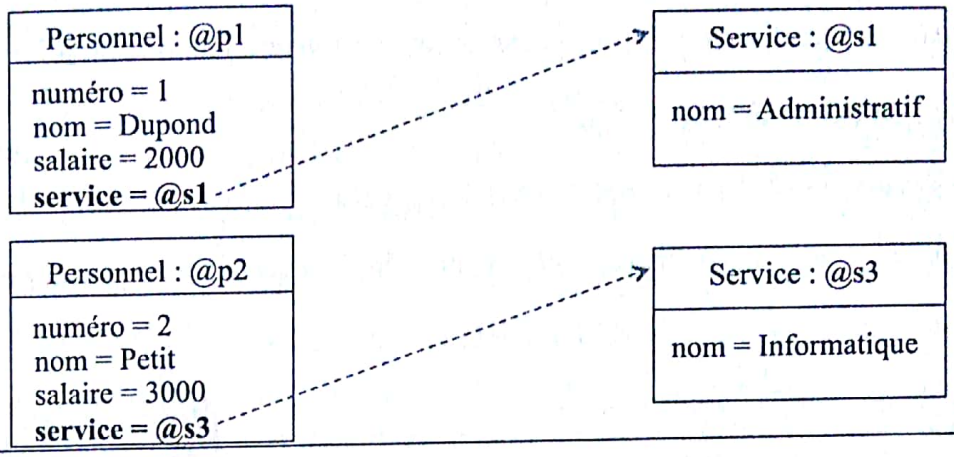
Relations entre objets

☞ Exemple (suite) : un personnel est affecté à un service

– On voudrait savoir à quel service est affecté un personne

– Un objet personnel garde (stocke) un lien vers le service auquel il appartient

– Exemple : le personnel *Dupond* est affecté au service *Administratif* et le personnel *Petit* au service *Informatique*



Conception objet

Relations entre objets

☞ Comportement lié aux relations

– A quel service est affecté le personnel *Dupont* ?

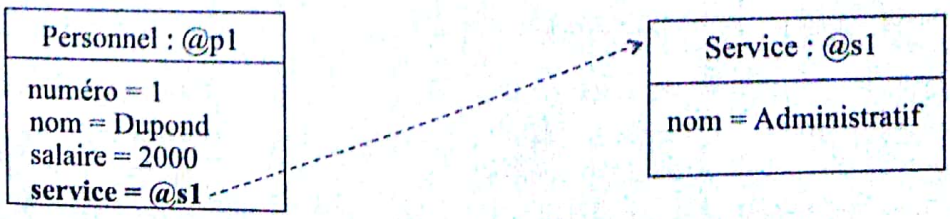
– A qui s'adresser ?

– A l'objet qui détient l'information, l'objet Personnel *Dupond* !

– L'objet *Dupond* répond : c'est le service *@s1*

– Mais on voulait savoir le nom du service !

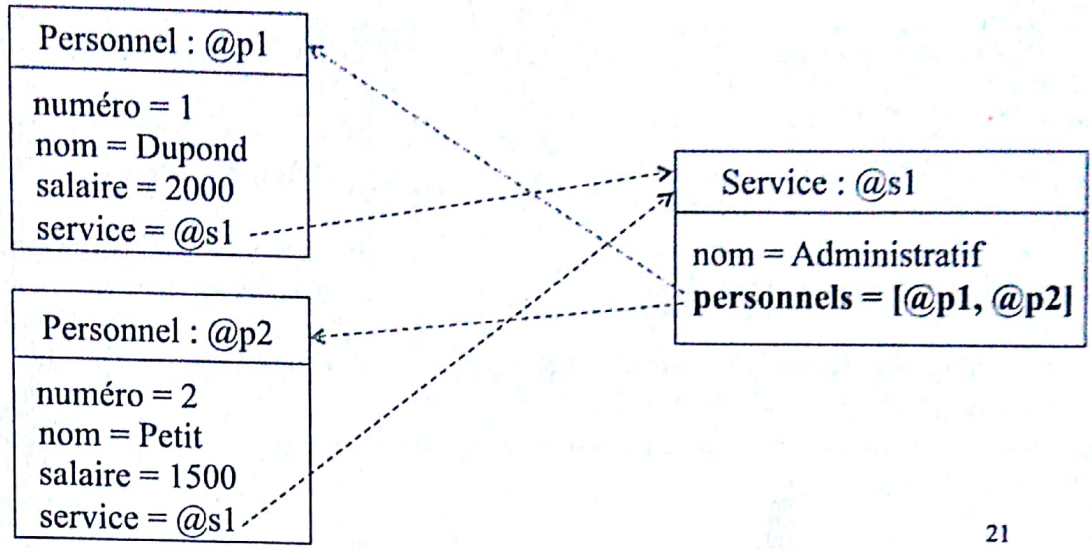
– Il suffit donc de demander au service *@s1* son nom !



Conception objet

Relations entre objets

- Exemple (suite) : un service comprend plusieurs personnels
  - Le service *Administratif* comprend les personnels *Dupond* et *Petit*



Conception objet

Relations entre objets

- Comportement lié aux relations
  - Quels sont les personnels du service *Administratif* ?

    Demander à l'objet Service *@s1* la liste de ses personnels

    Il renvoie la liste [@p1, @p2]
  - Embaucher un nouveau personnel (3, Dédé, 2200) au service *Administratif*

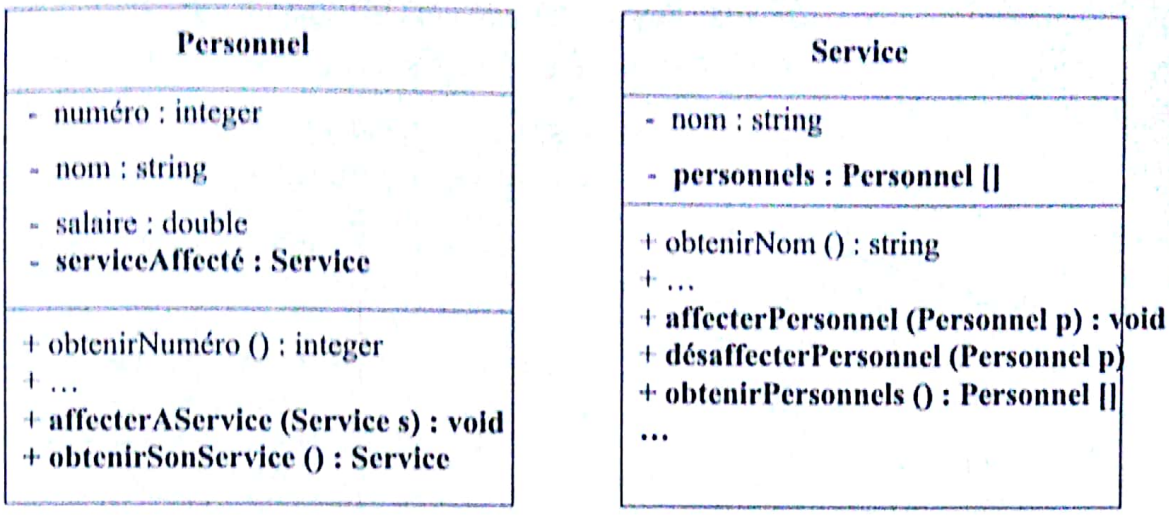
    Créer un nouveau objet Personnel (son identité est *@p3*)

    Demander à l'objet @s1 de l'ajouter à la liste de ses personnels

Conception objet

Modélisation de classe avec UML

☞ Exemple : Comportement lié aux relations



Conception objet

Modélisation de classe avec UML

☞ Associations entre classes

- Exprime une relation sémantique bidirectionnelle entre classes
- Abstraction des liens entre objets de l'application
- Permettent les interaction entre objets
- Deux catégories de relations : structurelles et non structurelles

☞ Association structurelle

- Exprime une relation sémantique uni ou bidirectionnelle entre classes
- Lien informatif, permettant de stocker une information d'un objet
- Exemple : à quel service est affecté un personnel ?

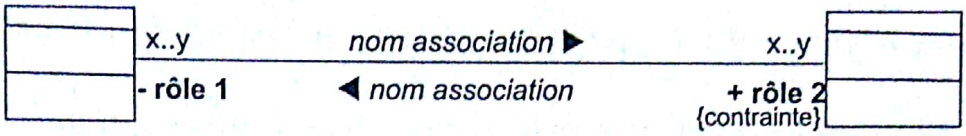
☞ Association non structurelle (dépendance entre classes)

- Tout lien ne servant pas à stocker un lien d'objet
- Exemple : un objet appelle une méthode d'un autre objet passé en paramètre

Diagramme de classes

Relations entre classes

- Association structurelle
  - Exprime une relation sémantique (uni ou bidirectionnelle) entre les objets des classes associées
  - Possède un nom, deux rôles, deux multiplicités, des contraintes, …

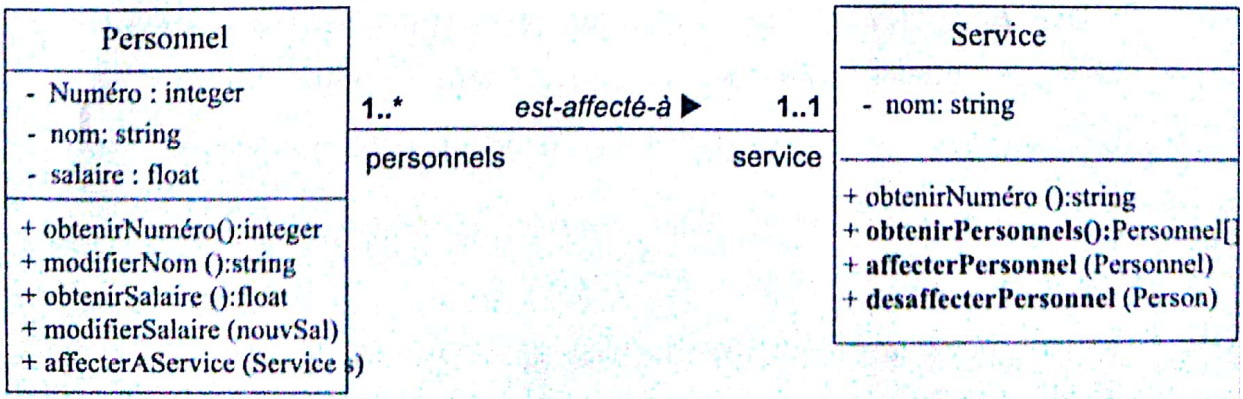


  - *Nom de l'association* : forme verbale, sens de lecture avec flèche
  - *Rôles* : forme nominale, identification d'une extrémité de l'association
  - *Multiplicités (x..y)* : 0..1, 1..1 (ou 1), 0..* (ou *), 1..*, M..N

Diagramme de classes

Relations entre classes

- Exemple :
  - Un personnel est affecté à un seul et unique service
  - Un service comprend un ou plusieurs personnels



Conception objet

Analyse des fonctions de l'application

☞ Une application à objets est une société d'objets

– Les objets interagissent pour réaliser les fonctions de l'application (*cf.* DCU)

– Chaque objet contribue en réalisant une partie d'une fonction de l'application

☞ Conception d'une fonction de l'application

– Déterminer les entrées et sortie de la fonction → opération principale

– Décrire les macro-étapes de la fonction

– Trouver l'objet (classe) à qui attribuer la responsabilité de la fonction (celui à qui confier l'opération)

– Déterminer les objets (classes) participants à la réalisation de la fonction et le rôle de chacun → opérations intermédiaires

---

Conception objet

Analyse des fonctions de l'application

☞ Exemple (suite) : embauche d'un personnel (1)

– Embaucher le nouveau personnel ***(501, Dupuis, 2300 eur)*** au service *Administratif* (géré par l'objet ***@s1***)

1. Demander à l'objet *@s1* d'embaucher le nouveau personnel (*501, Dupuis, 2300 eur)*. opération *embaucherPersonnel (numéro, nom, adresse)*

2. L'objet *@s1* crée (instancie) un nouvel objet Personnel *@p501*

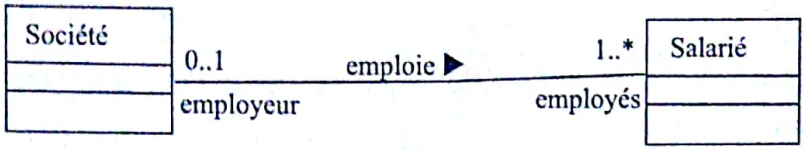
3. L'objet *@s1* demande à l'objet personnel *@p501* de stocker les renseignements (*501, Dupuis, 2300 eur)*

4. L'objet *@s1* ajoute l'objet Personnel *@p501* à sa liste de personnels

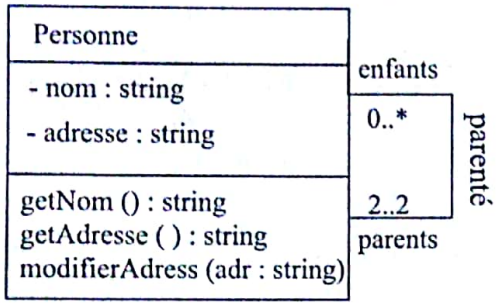
Diagramme de classes

Relations entre classes

☞ Rôle



– Les rôles sont nécessaires pour la compréhension des associations réflexives



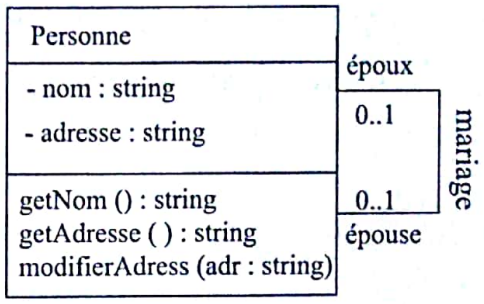
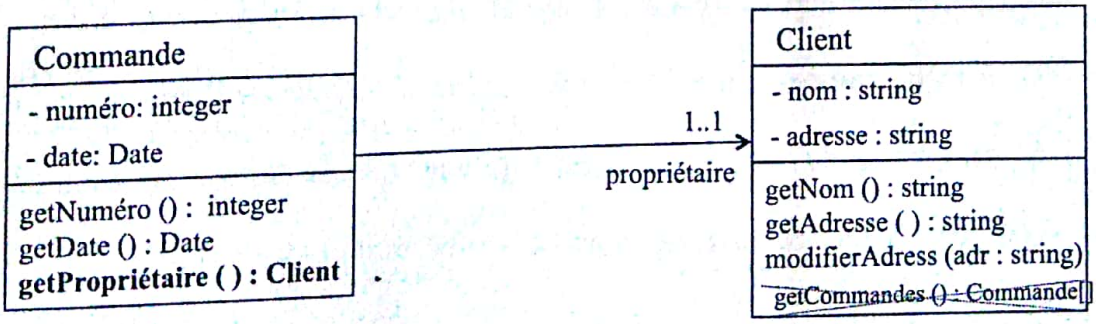


Diagramme de classes

Relations entre classes

☞ Association unidirectionnelle

– Le lien n'est stocké que dans une seule classe !



– Objets de l'application

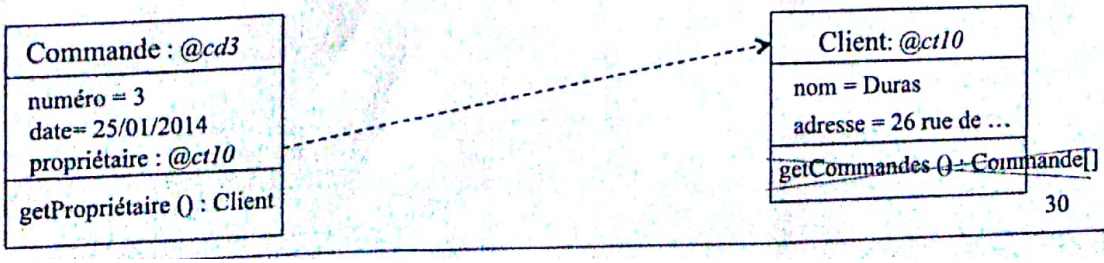
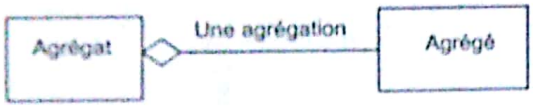


Diagramme de classes

Relations entre classes

- Agrégation
  - Association exprimant le lien de composition d'objets : *composite (agrégat)* et *composant (agrégé)*
  - Non symétrique : une des extrémités joue un rôle prédominant par rapport à l'autre
  - Association simple dont le nom pourrait être *est-partie-de, fait-partie-de, est-composé-de, est-constitué-de, ...*
  - Un objet *composant* peut être utilisé pour composer un ou plusieurs objets *composites*



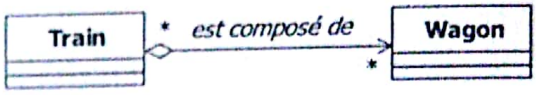
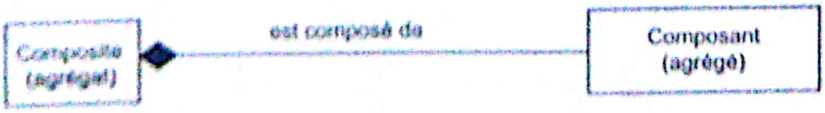


Diagramme de classes

Relations entre classes

- Composition
  - Cas particulier de l'agrégation. C'est une agrégation forte
  - La durée de vie de l'objet *composant* est limitée à celle de l'objet *composite*
  - La destruction de l'objet *composite* implique celle de tous ses *composants*
  - Un objet *composant* ne peut composer qu'un seul objet *composite*



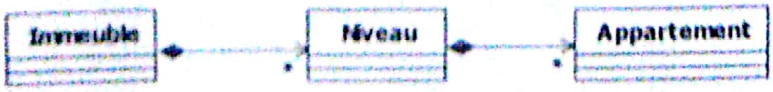
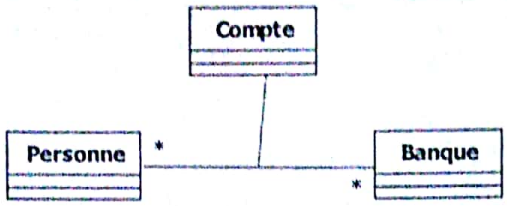


Diagramme de classes

Relations entre classes

- Classe d'association
  - Une association peut être amenée à porter des données
  - Une donnée portée par une association dépend (fonctionnellement) à la fois de tous les objets participants à cette association
- Exemple

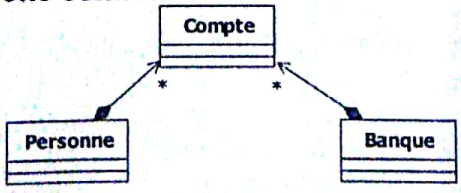


- Contraintes implicites
  - L'existence d'un objet de la classe d'association dépend de l'existence des objets associés
  - Une association ne peut lier qu'une seule fois les mêmes objets

Diagramme de classes

Relations entre classes

- Transformation de la classe d'association
  - Contrainte implicite : l'association ne peut lier qu'une seule fois les mêmes objets
  - Comment se passer de cette contrainte ?
- 1ère possibilité



- 2ème possibilité : contrainte {bag}

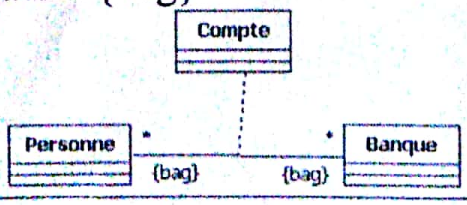
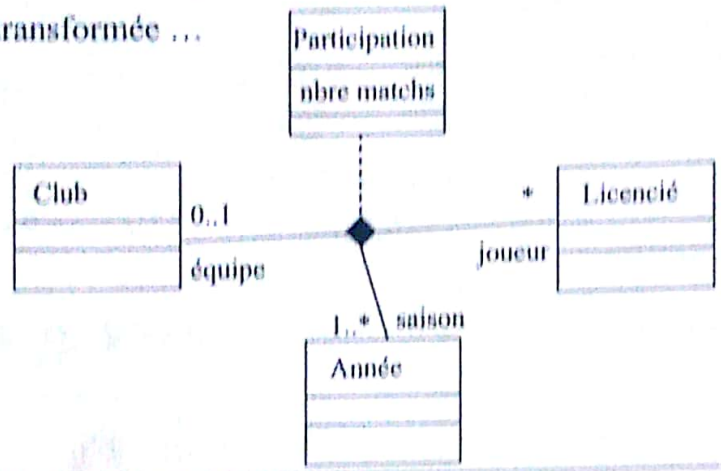


Diagramme de classes

Relations entre classes

- Association *n-aire*
  - Association de trois objets ou plus
  - Sémantique ambigüe
  - Souvent transformée …
- Exemple



Conception objet

Modélisation de classes

- Reste à voir
  - Héritage et polymorphisme
  - Interface, implémentation et polymorphisme d'interface
- A suivre …

Bases de la conception « orientée objet »

DUT Informatique
Semestre 2

Mourad Ouziri
mourad.ouziri@parisdescartes.fr

IUT de Paris Descartes

PARIS DESCARTES

PARIS DESCARTES

Conception objet :

« Modélisation des interactions avec le diagramme de séquence UML »

Objectif : savoir analyser la structure de l'application et identifier les objets participant à l'accomplissement d'une tâche et leurs interactions

Introduction

- Modèle fonctionnel – Diagramme de Cas d'Utilisation UML
  - QUOI/QUELLES fonctionnel ?
  - Besoins fonctionnels du point de vue des utilisateurs (acteurs)
- Modèle objet – Diagramme de Classes UML
  - QUOI/QUELLES données ?
  - Structure des objets (données) de l'application
- Modèle d'interactions – **Diagramme de séquence** UML
  - COMMENT ?
  - Comment les objets collaborent pour réaliser une fonction donnée du DCU
  - Deux types de diagrammes d'interactions : diagramme de séquence et diagramme de collaboration

Modélisation des interactions

Analyse des fonctions de l'application

- Rappel : application à objets = société d'objets en interaction
- Conception d'une fonction de l'application
  - L'objet « Responsable » de la fonction (expert d'information)
  - L'objet « Porte-parole » de la fonction
  - Les objets « participants » à la séquence d'interactions

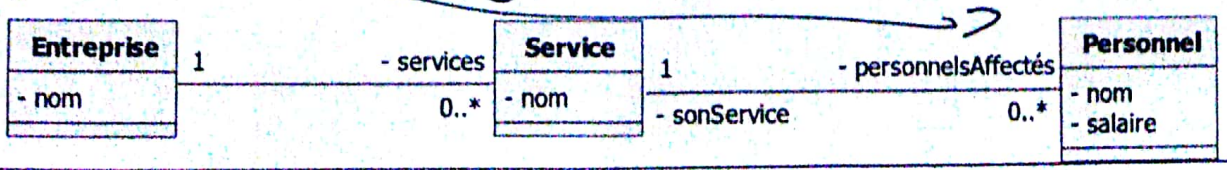


Diagramme de séquence

Définition et syntaxe

- Définition
  - Représente les interactions entre objets de manière chronologique
  - Montre comment les objets échangent des messages pour réaliser un scénario (ou plusieurs) d'un CU
  - Formalise les fiches descriptives de CU
- Syntaxe graphique
  - En colonnes :
    - Objets (instances de classes) participant au scénario
    - Ligne de vie et périodes d'activité des objets
  - En lignes : messages échangés par les objets dans l'ordre chronologique
  - Axe du temps : axe vertical allant du haut vers le bas

Diagramme de séquence

Syntaxe graphique

- Exemple

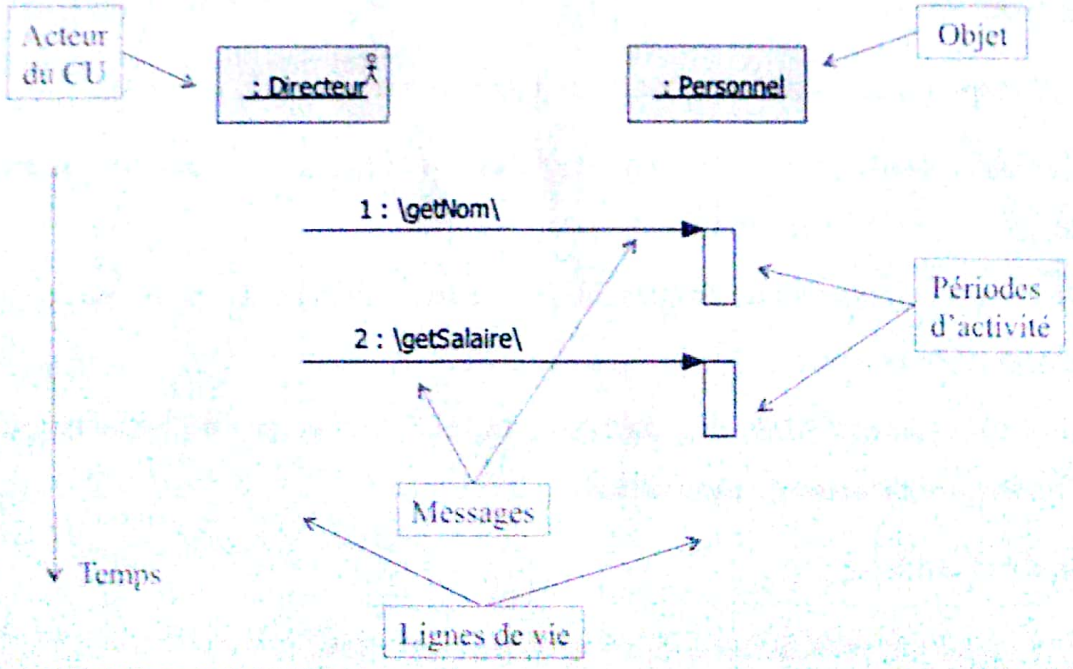


Diagramme de séquence
Eléments de modélisation

☞ Message

- Moyen unique de communication entre objets (encapsulation)
- Objet O1 envoie un message à Objet O2. O2 exécute le message et renvoie, le cas échéant, le résultat à O1
- Concrètement, un message correspond à l'appel d'une méthode d'un objet
- Lorsque le message comporte des informations, il convient de les indiquer en paramètre du message (exemple : *modifierSalaire (**Marie, 3000**)*)

Diagramme de séquence
Eléments de modélisation

☞ Ligne de vie

- Elle représente l'existence d'un objet pendant l'exécution du scénario.
- Si l'objet existe pendant toute cette période, sa ligne de vie couvre l'ensemble de la hauteur du diagramme.
- Si l'objet est créé lors de l'exécution du scénario, sa ligne de vie débute après le message de création.
- Si l'objet est détruit lors de l'exécution du scénario, sa ligne de vie se termine à l'arrivée du message de destruction.

☞ Période d'activité

- Elle montre la période pendant laquelle un objet accomplit une action

Diagramme de séquence

Exemple

- Exemple
  - Consulter le nom du service d'affectation d'un personnel

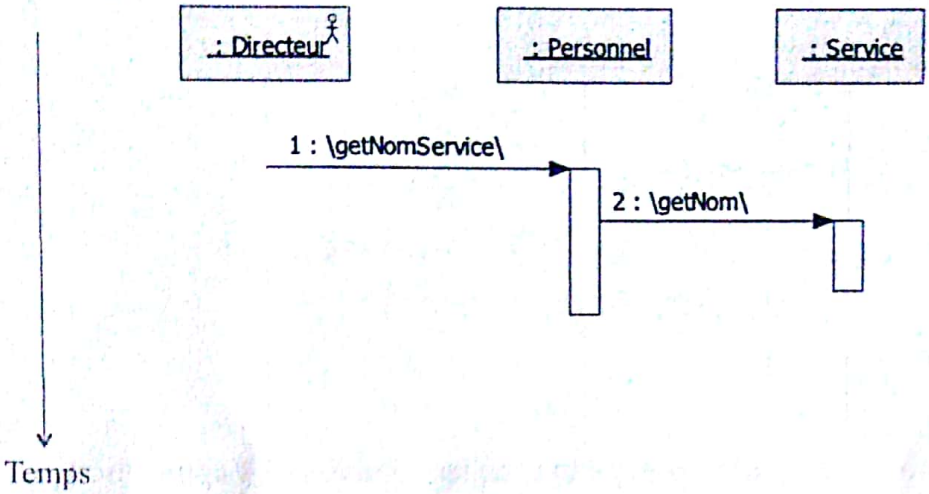
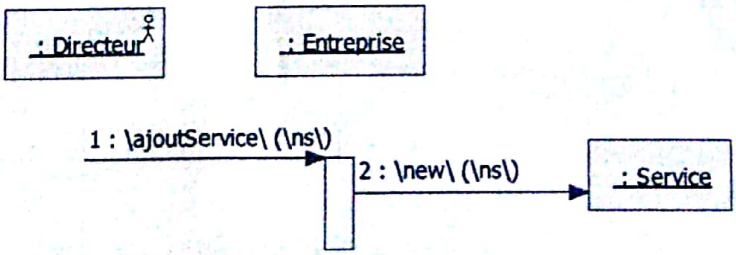


Diagramme de séquence

Eléments de modélisation

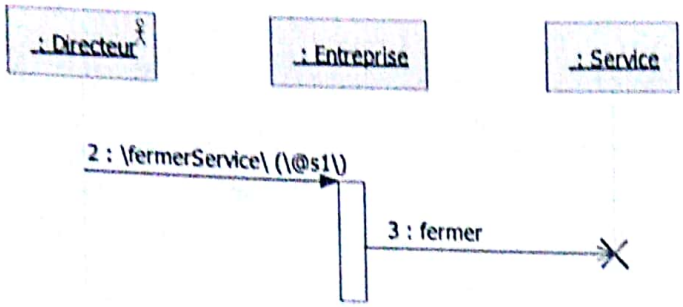
- Création d'objets
  - Ajout d'un nouveau service



  - Exercice : faire l'embauche d'un nouveau personnel à un service

Diagramme de séquence
Eléments de modélisation

- Destruction d'objets
  - Fermeture d'un service et sa suppression de l'application



  - Exercice : faire le départ d'un personnel et sa suppression de l'application

Diagramme de séquence
Conditions et gardes

- Messages conditionnés
  - Augmentation de salaires : 10% si inférieur à 2000 euro et 5% sinon

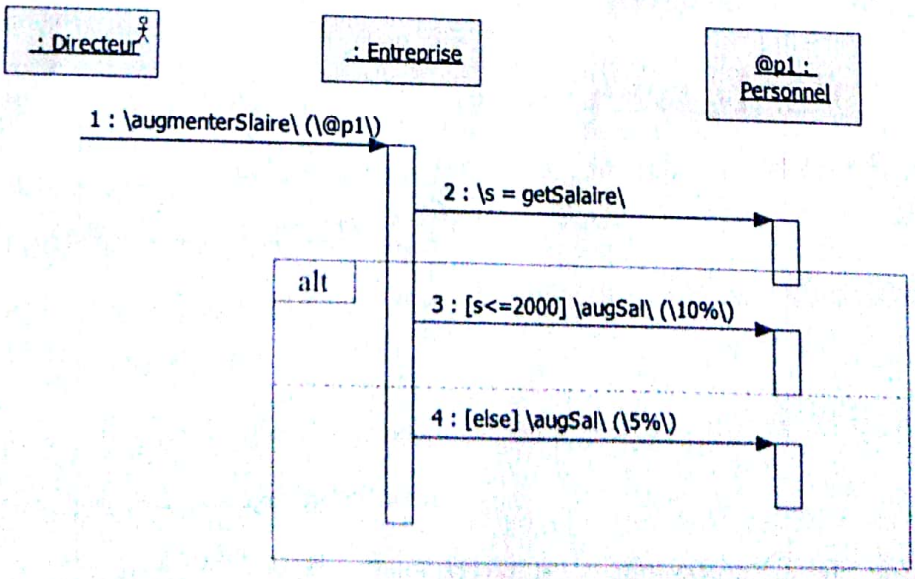


Diagramme de séquence

Itérations

- Messages itérés sur plusieurs objets
  - Calculer le coût salarial du service *@s1* (somme des salaires de ses personnels)

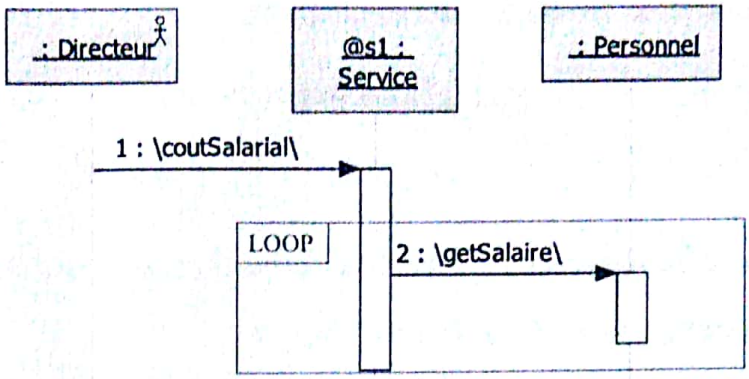


Diagramme de séquence

Itérations

- Messages itérés sur plusieurs objets
  - Rechercher le salaire de *Dupond* du service *@s1*

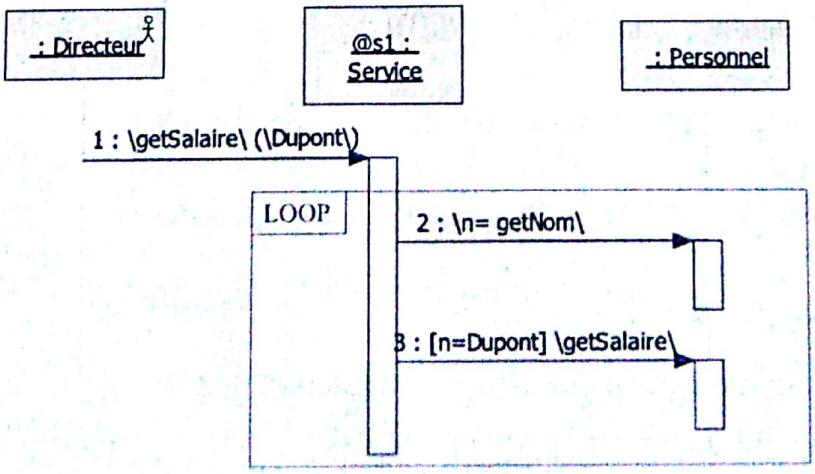


Diagramme de séquence

Point d'entrée d'une application objet

- Point d'entrée de l'application objet
  - Objet unique (Singleton) représentant la totalité de l'application
  - Il permet d'atteindre tous les objets de l'application
  - Un diagramme de séquence complet commence souvent par cet objet
- Exemple
  - Faire le diagramme de séquence permettant de consulter le salaire de « Dupont »
    - Cas 1 : on connait l'objet *@p2* représentant *Dupont*
    - Cas 2 : on ne connait que le nom du personnel, l'objet correspondant doit d'abord être trouvé en passant l'objet unique (point d'entrée) de l'application

Principe de délégation

- Flot de contrôle centralisé
  - Un traitement est réalisé par un seul objet (une de ses méthodes)
  - Non recommandée : maintenance difficile
- Délégation : flot de contrôle décentralisé
  - Un traitement est réalisé par plusieurs objets différents
  - Délégation du traitement effectif à l'objet « Responsable »
  - Avantage : faible dépendance, bonne maintenabilité

Diagramme de séquence

Flots de contrôle

- Flot de contrôle centralisé
  - Consulter le coût salarial de l'entreprise

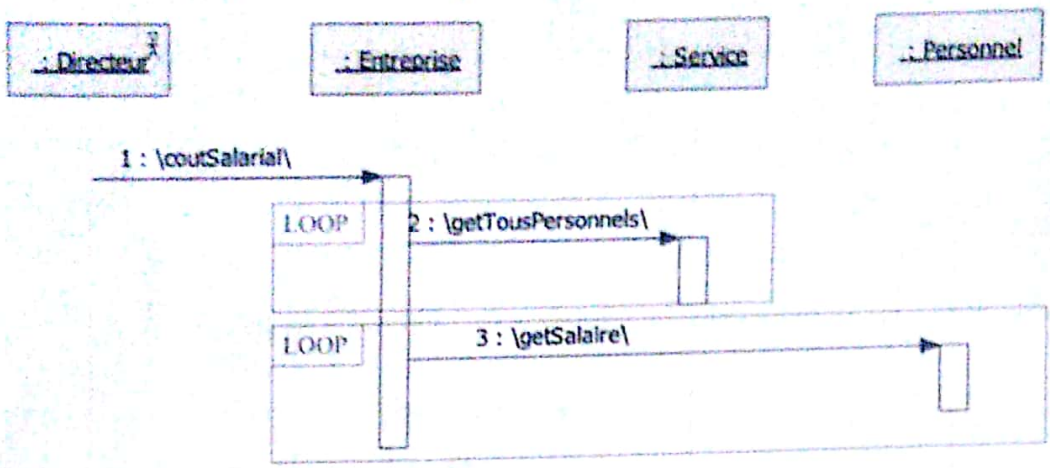


Diagramme de séquence

Flots de contrôle

- Flot de contrôle décentralisé : la délégation de traitements
  - L'entreprise Délègue le calcul à ses services

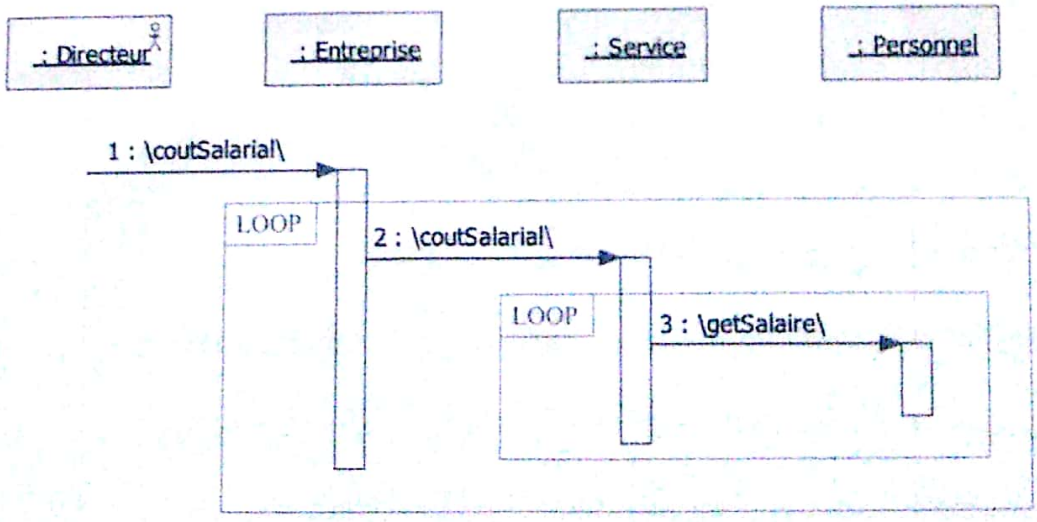
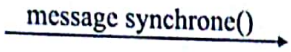


Diagramme de séquence

Types de message

- Synchrone
  - Message bloquant son émetteur jusqu'à la fin de son exécution



- Asynchrone
  - L'émetteur poursuit son exécution après l'envoie du message
  - Nécessité de message de retour si résultat requis

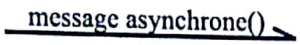
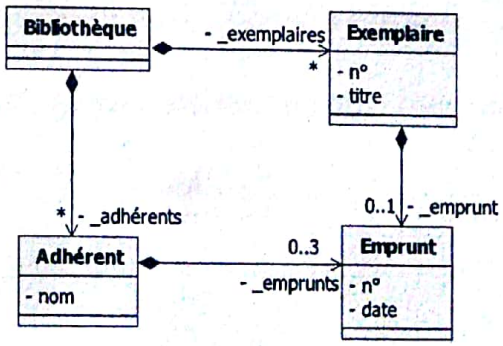


Diagramme de séquence

Exemple de synthèse

- Exemple : gestion simplifiée d'une bibliothèque



  - Inscrire un nouvel adhérent (nom de l'adhérent)
  - Vérifier la disponibilité d'un exemplaire (n° de l'exemplaire)
  - Enregistrer un emprunt (nom de l'adhérent, n° de l'exemplaire)
  - Enregistrer le retour d'un emprunt (n° de l'exemplaire)

Diagramme de séquence

Conclusion

- Deux recommandations :
  - Vérifier la cohérence des DS par rapport au diagramme de classes
  - Concevoir par délégation

Bases de la conception « orientée objet »

DUT Informatique
Semestre 2

Mourad Ouziri
mourad.ouziri@parisdescartes.fr

IUT de Paris Descartes

PARIS DESCARTES IUT

PARIS DESCARTES

Conception objet :

« Diagramme d'états-transitions UML »

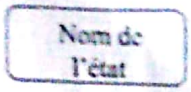
Objectifs :

- Schématiser le comportement attendu des objets de l'application
- Etablir un modèle de tests unitaires du comportement des objets

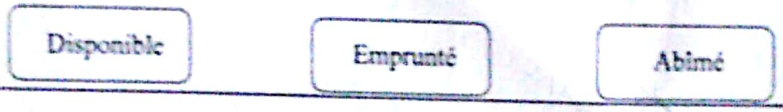
UML

DE – Diagramme d'états-transitions

- Objectif
  - Décrire le comportement des objets d'une classe
- Définition
  - Automate déterministe à états finis représentant les états pertinents d'un objet et les transitions faisant évoluer cet objet d'un état à un autre
- État
  - Situation durable d'un objet
  - Conjonction instantanée des valeurs des attributs et associations de l'objet
- Syntaxe graphique



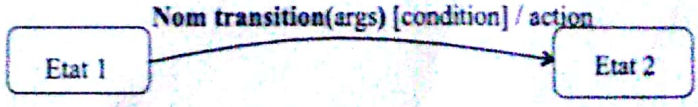
- Exemple : quelques états d'un exemplaire de livre



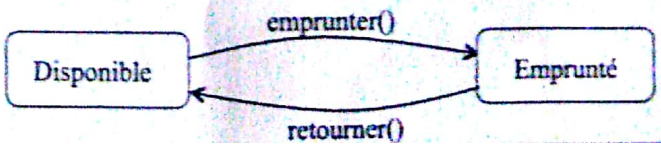
UML

DE – Diagramme d'états-transitions

- Transition
  - Passage instantané d'un état à un autre (ou dans le même état)
  - Elle est déclenchée par un événement
  - Deux types d'événements :
    - Message : réception d'un message par l'objet concerné (méthode dans la classe correspondante)
    - Temporel : correspond à un instant donné : *when(attr=10h)*, après un certain délai : *after(3jours)*
  - Elle peut être conditionnée : la transition n'est déclenchée que si la condition est vérifiée
- Syntaxe graphique



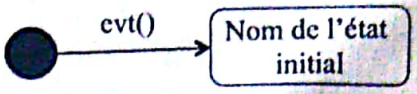
- Exemple



UML

DE – Diagramme d'états-transitions

- États particuliers
  - État initial

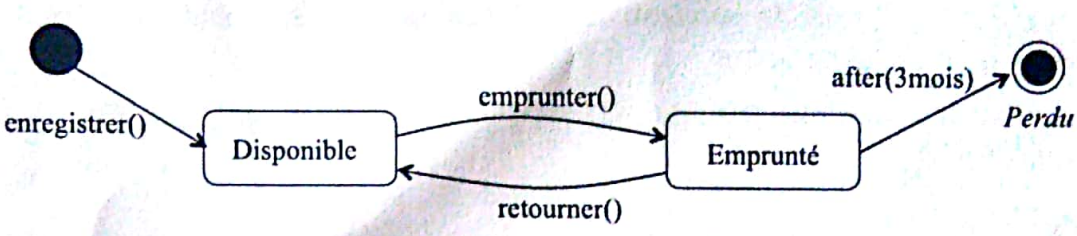


Premier état dès création d'un objet. Il est obligatoire et unique.

  - État final

État représentant la destruction de l'objet. Il est facultatif et peut être multiple.

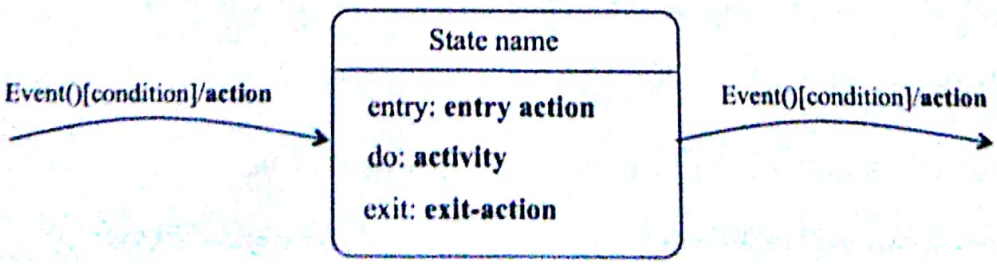
- Exemple



UML

DE – Diagramme d'états-transitions

- Action
  - Traitement instantané accompagnant une transition
  - Il peut être exécuté avec la transition, à l'entrée ou à la sortie d'un état
- Activité
  - Traitement nécessitant un certain temps d'exécution
  - Il est exécuté lorsque l'objet rentre dans l'état et interrompu à la sortie de l'état
- Syntaxe graphique



UML

DE – Diagramme d'états-transitions

- État composite ou généralisation d'états
  - Élément de structuration des diagrammes d'états-transitions
  - C'est un pseudo-état qui regroupe plusieurs états
  - Il permet de simplifier les diagrammes d'états
- Un état composite est créé à partir des :
  - États partageant les mêmes propriétés
  - États partageant les mêmes transitions d'entrée et de sortie
- Historique dans un état composite
  - Lorsqu'un objet rentre de nouveau dans un état composite, il peut retrouver son dernier (sous-)état de sortie
  - Le dernier état de sortie peut donc être mémorisé (symbole H dans l'état composite)
- Exemple

DE

Tests unitaires

- DE : modèle de comportement des objets d'une classe
- Tests unitaires : tests à l'échelle d'une classe
- Etat d'un objet
  - Situation durable de l'objet, obtenue à partir des valeurs de ses attributs
  - Valeur subjective et dépend du contexte de l'objet
- Indépendance de la classe testée du diagramme d'état
  - Possibilité de tester la classe avec différents diagrammes d'états
- Un diagramme d'états : une classe de tests
  - Les états du DE sont définis par la classe de tests associée
  - Les transitions sont testées dans cette même classe de tests associée

DE

Tests unitaires

- Classe de tests : tester deux types de transitions
  - Transitions possibles : définies dans le DE
    - Transition initiale : instanciation de l'objet
    - Transitions non initiales : toute autre transition définie dans le DE
  - Transition non possibles : celles non définies dans le DE
- Structure d'une classe de test DE

```
class TestClassseAvecDE1 {
    private Etat getEtat (Classe o) {
    }
    @Test
    public void testTransition1 () {
    }
}
```

```
enum Etat {
        Etat1,
        Etat2,
        …
        Etatn
}
```


DE

Tests unitaires

- Structure de test d'une transition définie

```
class TestClassseAvecDE1 {
    @Test
    public void testTransitionDéfinie1 () {
            // GIVEN : état de départ
            // WHEN : transition
            // THEN : état final
            // CLOSE : remise à zéro {optionnel}
    }
}
```

DE

Tests unitaires

- Structure de test d'une transition non définie
  - L'objet se trouve forcément dans un état (de départ)
  - La transition non définie passera l'objet dans un état indéfini

```
class TestClassseAvecDE1 {
    @Test (expected=UneException.class)
    public void testTransitionNonDéfinie1 () throws UneException {
                    // GIVEN : état de départ
                    // WHEN : transition
    }
}
```