Programmation Orientée Objet

by

Dr. Jordan F. Masakuna

Assistant

Contact

- Jordan F. Masakuna
  - masakunafelicien@gmail.com
  - jordan@aims.ac.za
  - +243976596074
  - +27743221221

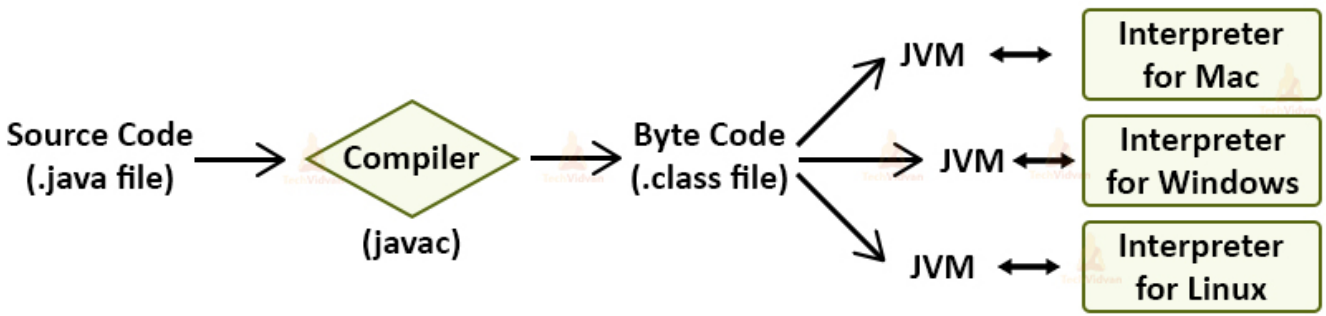
Introduction

- ensemble des activités qui permettent l'écriture des programmes informatiques
- programme informatique: un jeu d'instructions élémentaires pour résoudre un problème
  - programme source: code en langage proche de l'homme
    - langage évolué
  - programme binaire: code à exécuter par un microprocesseur
    - langage machine
  - traducteur?

Traducteur

- traducteur: traduit programmes écrits en langage évolué en langage machine
  - éditeur de textes: e.g. NetBeans, IntelliJ
- trois niveaux de traduction
  - niveau lexical : le vocabulaire du langage
  - niveau syntaxique : la grammaire du langage
  - niveau sémantique : la signification de la notation
- types:
  - compilateur: analyse et traduit l'ensemble du programme
    - e.g. C, C++, Cobol, Fortran, Pascal
  - interpréteur: traduit le programme instruction par instruction
    - e.g. Basic, Matlab, Perl
  - **Java?**

Java: approche hybride



- programme Java n'est analysé et compilé par une machine physique
- ... mais par une machine virtuelle (JVM)
- byte-code = pseudo-assembleur
- JVM = Java Virtual Machine
- JVM facilite la portabilité

Logiciel

- ensemble de programmes assurant une tâche spécifique (e.g. logiciel de gestion de la relation client)
- progiciel: paquet de logiciels ou multi-logiciels
- comment construire un logiciel?
  - jeu enfantin des instructions élémentaires
  - avec des exigences de qualité mesurables

Problème du logiciel

- taille
- coût de développement
- coût de maintenance
- fiabilité

Solutions voulues

- qualité d'un logiciel
  - exactitude: fournir des résultats voulus dans des conditions normales
  - robustesse: bien reagir lorsqu'on s'écarte des conditions normales
  - extensibilité: être adapté à une évolution des spécifications
  - réutilisabilité: être utilisé pour un autre problème
  - modularité: regroupement de fonctions pour faciliter la réutilisabilité
  - portabilité: être utilisé sous d'autres implémentations
  - efficience: fonctionné en temps et couts réduits
  - abstraction : ses entités proches de celles du monde réel
  - sûreté : encapsulation et le typage des entités

Paradigmes de programmation

- les principes fondamentaux du développement de logiciels
  - programmation impérative: déroulement des étapes à suivre
    - programmation structurée: structures de contrôles et boucles
    - programmation procédurale: subdivision en routines
    - programmation modulaire: subdivision en modules
    - **programmation orientée objet**?
    - e.g. C, C++, Java, Python, Pascal, Fortran, ALGOL et langages d'assemblage
  - programmation déclarative: description de l'objectif (ou résultat) à atteindre
    - programmation logique: bases de faits et de règles, et moteur d'inférence
    - programmation fonctionnelle: tous les éléments peuvent être compris comme des fonctions
    - e.g. Lisp, ML, Haskell, F#, Prolog et Oz

POO

- La programmation orientée objet (POO) est un paradigme de programmation informatique qui consiste à définir des composantes logicielles afin de produire des logiciels de qualité

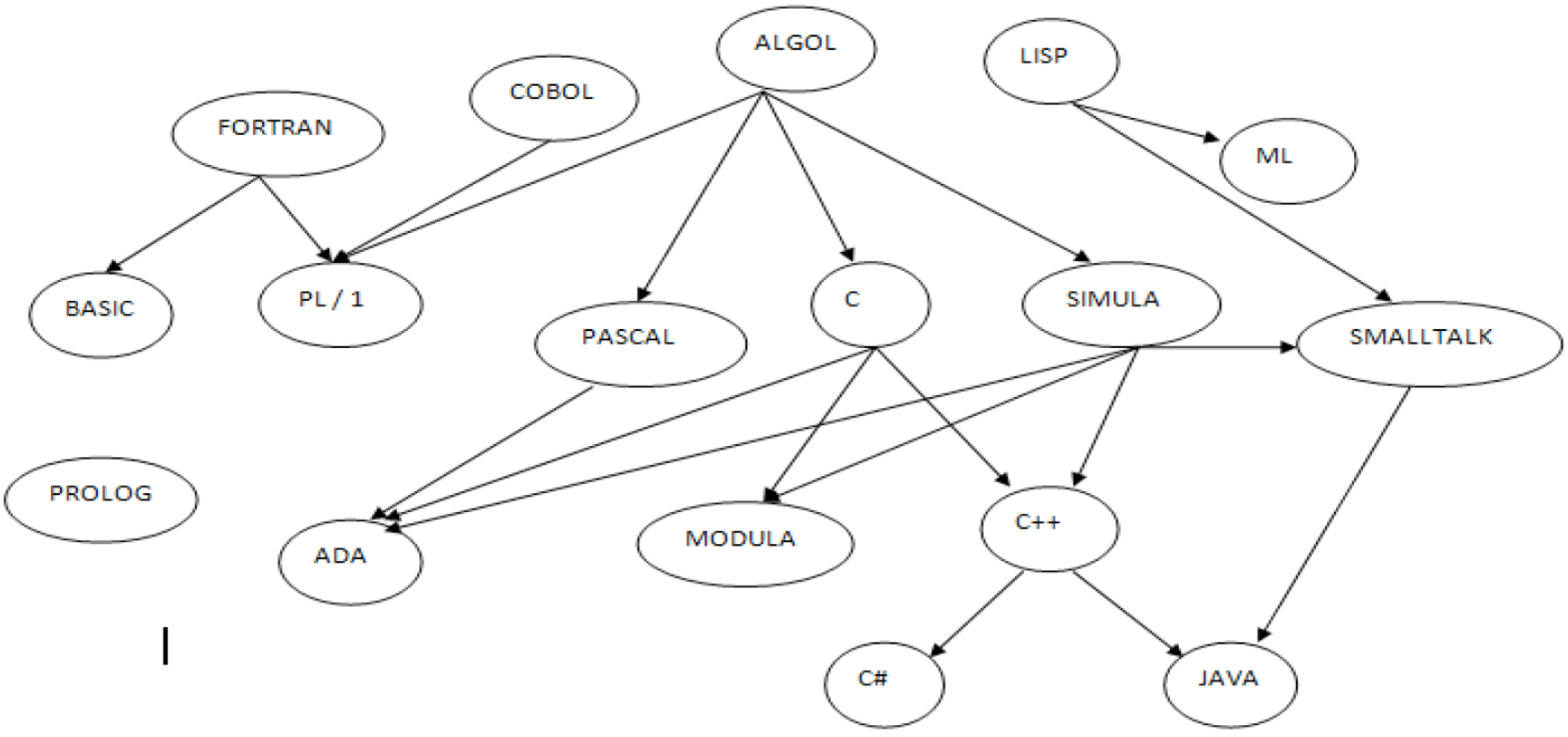
Histoire

- fonctions + procédures: (60, Fortran)
- typage des données: (70, Pascal et Algol)
- modules: données + fonctions regroupées (80, Ada)
- POO

Histoire

- **Alan Kay**: auteur de la POO en 1966-67
- **Simula**: un premier langage de POO

Généalogie des langages de programmation



Objectifs?

- comprendre la programmation comme un puissant outil informatique incontournable
- maîtriser un outil de modélisation de programmes informatiques
- maîtriser des concepts fondamentaux de la POO
- développer des compétences d'analyse, de conception, d'implémentation et d'optimisation des programmes informatiques
- maîtriser des concepts de la POO en Java
- connaître des méthodes de cycle de vie du développement logiciel
- développer des logiciels qui résolvent les problèmes des entreprises

Plateformes de Java

- Java Card
- Java ME (Micro)
- Java SE (Standard)
- Jakarta EE (Enterprise)
- JavaFX

Outils

- langage: Java
- plateforme: Java SE
- editeur: NetBeans

Évaluation

- plusieurs travaux dirigés (généralement à la fin de chaque séance)
- une interrogation orale sous forme d'exposé
- trois travaux pratiques (deux en groupe et un autre individuel)
- examen

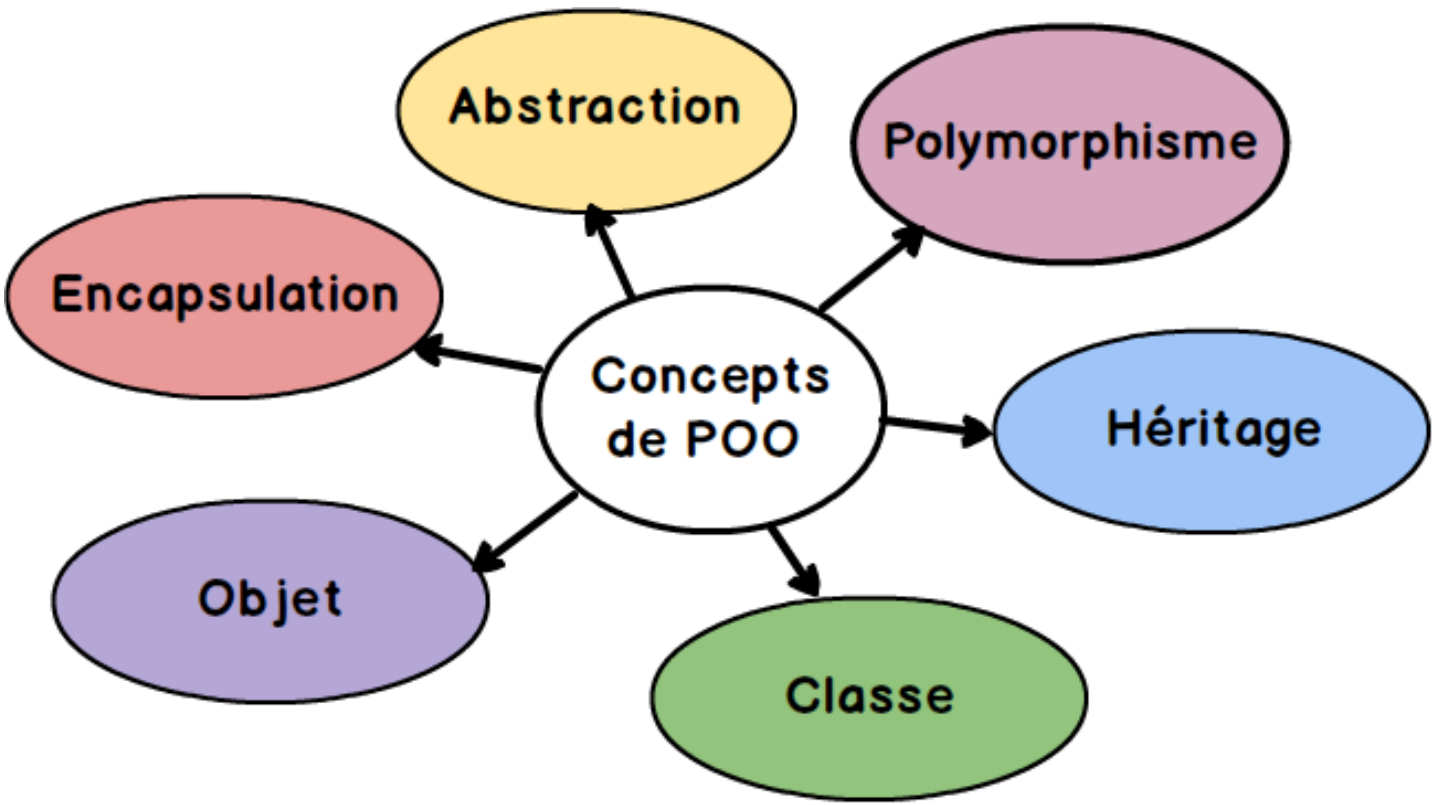
Content layout

- Paradigmes de la programmation orientée objet
- Unified Modelling Language (UML)
- Cycle de vie du développement logiciel
- Industrie logicielle

Content layout

- Paradigmes de la programmation orientée objet
- Unified Modelling Language (UML)
- Cycle de vie du développement logiciel
- Industrie logicielle

Concepts de POO



Documentation

(1) Gamma, Erich. *Design patterns: elements of reusable object-oriented software*. Pearson Education India, 1995.

(2) Freeman, Eric, et al. *Head first design patterns.* " O'Reilly Media, Inc.", 2008.

(3) Sierra, Kathy, and Bert Bates. *Head first java*. " O'Reilly Media, Inc.", 2003.

(4) http://julien.sopena.fr/enseignements/L3-PRO-JAVA/cours/03-Polymorphisme/03-Polymorphisme.pdf

(5) Horstmann, C. S., & Cornell, G. (2001). *Core Java 2: Fundamentals (Vol. 1)*. Prentice Hall Professional.

(6) Bloch, Joshua. Effective java. Pearson Education India, 2016.

(7) https://www.tutorialspoint.com/java/index.htm

Classe

- la description d'un ensemble d'entités ayant une structure de données commune et disposant des mêmes méthodes.
- exemple
  - classe: **université**
  - objet: **UNH, UNIKIN, UNILU** etc.

Classe en Java

```
public class Universite
{
        ...
}
```

Classe

- attributs: types élémentaires (e.g. int, float) et type composite (e.g. classe, String)
- il existe généralement cinq types de méthodes:
  - **accesseur (getter)**: accéder aux propriétés d'une classe
  - **modificateur (setter)**: changer le comportement d'une classe
  - **constructeur**: pour construire un objet
  - **destructeur**: pour détruire un objet
  - **méthode principale**
- interface

Interface

- ensemble des méthodes publiques à travers lesquelles on peut interagir avec un objet
- ensemble de services visibles depuis l'extérieur

Classe en Java

```
public interface Personne
{

   String getNom();
   void setNom(String nom);
   void finalize();

}
```

Classe en Java

```
public class Personne implements Personne
{
  // attribut
  string nom;
  // getter
  public String getNom(){
    return nom;
  }
  //setter
  public void setNom(String nom){
    this.nom = nom;
  }
  public void finalize(){
        ...
  }

}
```

Objet

- une association des données et des procédures (ou méthodes) agissant sur ces données
- donnée = identité
- méthode = comportement
- données + méthodes = objet
- les objets apparaissent alors comme des variables d'un tel type classe (en POO, on dit un objet est une *instance* d'une classe)

Objet en Java

```
public class Universite
{
...
}
private Universite UNH = new Universite();
```

Encapsulation

- parfois impossible d'agir directement sur les données d'un objet
- interroge les méthodes
- méthodes = interface
- on traduit parfois l'appel d'une méthode par l'envoi d'un *message* à l'objet

Encapsulation

- Il existe six types d'encapsulation sur une entité POO (classe, objet, méthode ou attribut)
  - **public** : accès libre
  - **protected** : accès qu'aux descendants de la classe de définition
  - **private** : pas d'accès de l'extérieur
  - **package** : accès à l'intérieur du package
  - **static**: attribut de classe
  - **abstract**

Classe abstraite

- **classe abstraite**: est une classe dont l'implémentation n'est pas complète et qui ne peut pas être instanciée. Elle sert de base à d'autres classes dérivées
- **méthode abstraite**: ne peut être utilisée que dans une classe abstraite, et elle n'a pas d'implémentation. Son implémentation est fournie par des classes filles (ou sous-classes).

Classe abstraite en Java

```
abstract class Personne
{
  public abstract void  speak();

  public void sleep() {
    System.out.println("sleeping");
  }

}
```

Attribut de classe

- Par défaut, chaque instance d'une classe possède sa propre copie des attributs de la classe (**attribut d'instance**)
- les valeurs des attributs peuvent donc différer d'un objet à un autre.
- cependant, il est parfois nécessaire de définir un attribut de classe (**static**) qui garde une valeur unique et partagée par toutes les instances de la classe
- les instances ont accès à cet attribut mais n'en possèdent pas une copie
- un attribut de classe n'est donc pas une propriété d'une instance mais une propriété de la classe et l'accès à cet attribut ne nécessite pas l'existence d'une instance.

Encapsulation en Java

```
public class Universite
{

   public String  name;
   private int ranking;
   protected float capital;
   // attribut de classe
   static String pays = "RD.Congo";

}
```

Abstraction

- une représentation informatique des objets du monde réel
- extraction des variables pertinentes attachées aux objets que l'on souhaite manipuler

Abstraction en Java

```
public class Universite
{

  public String  name;
  private int ranking;
  protected float capital;

}
```

Héritage

- définir une nouvelle classe (i.e. sous-classe) à partir d'une classe existante (i.e. classe de base) à laquelle on ajoute de nouvelles données et méthodes
- exemple
  - classe base: **personne**
  - sous-classe: **homme**

Héritage en Java

```
public class Personne
{

  private String  name;
  private int age;
  private String sexe;

}

public class Homme extends Personne
{

  private String  occupation;
  private float revenu;

}
```

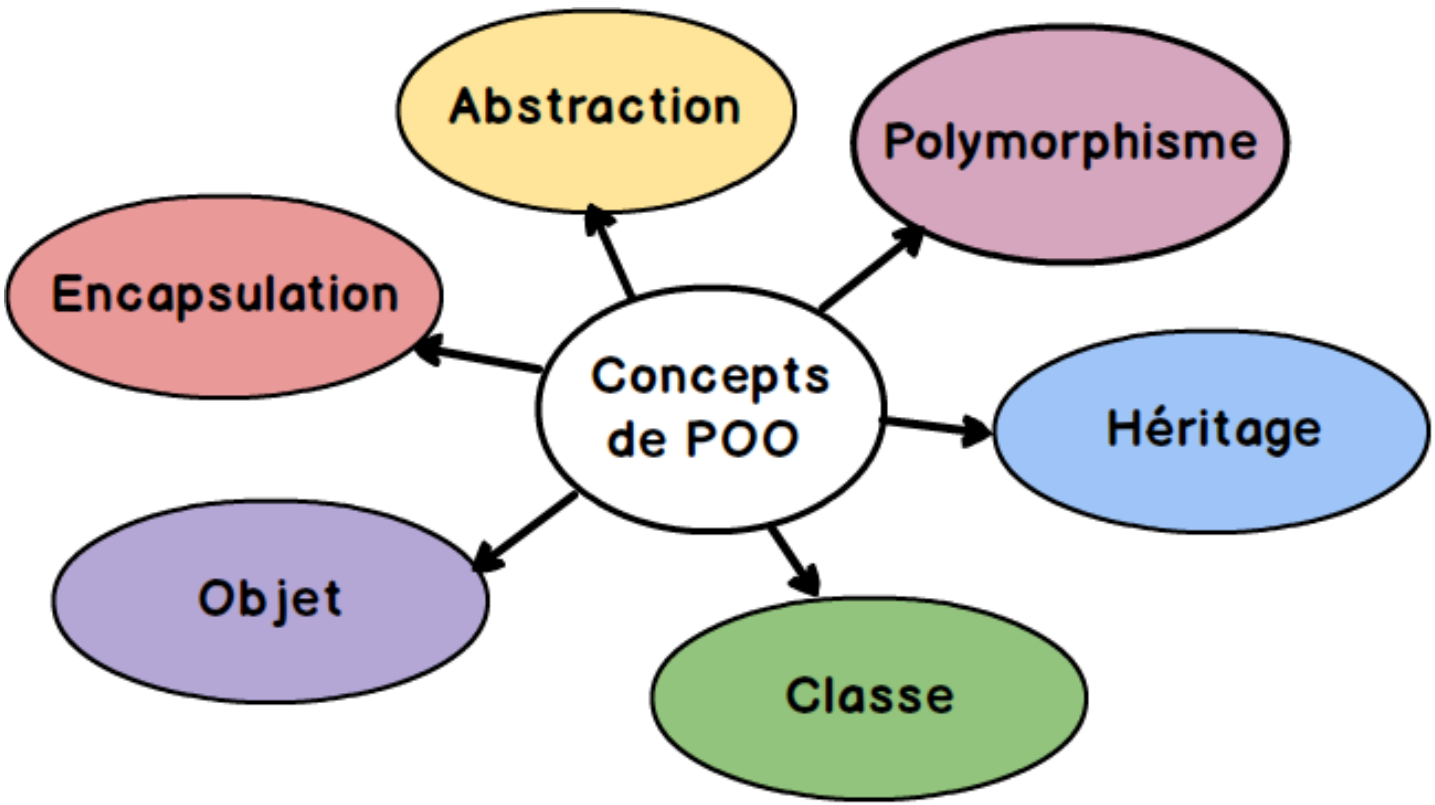
Polymorphisme

- une classe dérivée peut redéfinir ou modifier certaines méthodes héritées de sa classe de base
- la possibilité de traiter de la même manière des objets de types différents, pour peu qu'ils soient tous de classes dérivées de la même classe de base
- on utilise chaque objet comme s'il était de cette classe de base, mais son comportement effectif dépend de sa classe effective (dérivée de cette classe de base)

Polymorphisme en Java

```
public class Personne
{
  public Personne(){} //constructeur
  public void montreNationalite() {
    System.out.println("");
  }
}

public class Congolais extends Personne
{
  public void montreNationalite() {
    System.out.println("Congolaise");
  }
}
```

Résumé POO


Abstraction
Polymorphisme
Encapsulation
Concepts de POO
Héritage
Objet
Classe

Content layout

- Paradigmes de la programmation orientée objet
- Unified Modelling Language (UML)
- Cycle de vie du développement logiciel
- Industrie logicielle

Pourquoi UML?

- pour construire une **maison**, il faut établir un **plan**
- pour construire un **logiciel**, il faut établir un **modèle**
  - nécessité d'un outil de modélisation
  - le choix du modèle est important
- un modèle structure les informations
- un modèle est une représentation simplifiée d'une réalité

Pourquoi modéliser?

- la réalité a toujours été complexe
- un modèle est une représentation simplifiée de la réalité
- un modèle est une abstraction
  - modéliser revient à abstraire la réalité pour une compréhension approfondie du système
- un modèle sert à maîtriser la complexité de la réalité
- un modèle doit être relié au monde réel

Comment modéliser?

- choix du modèle à réaliser
- extraction des informations pertinentes de la réalité
- structuration des informations extraites
- corrélation entre modèle choisi et réalité
- raffinement du modèle

Étapes de construction d'un logiciel

Les étapes de réalisation d'un logiciel

- définition des besoins
  - besoins fonctionnels
    - les fonctionnalités du système
    - le comportement du système
  - besoins non fonctionnels
    - caractéristiques du système
    - nécessités en matière de qualité logiciel
    - contraintes d'implémentation (e.g. choix du langage de programmation et SGBD)

Étapes de construction d'un logiciel

- analyse
  - élaboration des spécifications de l'architecture générale du logiciel
- conception
  - définition précise de chaque sous-ensemble du logiciel
- développement
  - implémentation ou programmation
- test:
  - vérifie le bon fonctionnement d'une partie précise d'un logiciel
- validation:
  - démontre (y compris la documentation) qu' une activité conduit effectivement aux résultats escomptés

Étapes de construction d'un logiciel

- déploiement
  - mise en production
- maintenance
  - actions correctives (maintenance corrective)
  - actions évolutives (maintenance évolutive) sur le logiciel

Introduction

UML permet de modéliser toutes les étapes du développement d'un logiciel de l'analyse au déploiement

Introduction

UML est un langage de modélisation visuelle standard destiné pour

- modélisation métier et processus similaires
- analyse, conception et mise en œuvre de logiciels

Introduction

- UML est un langage commun pour les analystes métier, les architectes logiciels et les développeurs
- UML est utilisé pour décrire, spécifier, concevoir et documenter les processus métier existants ou nouveaux, la structure et le comportement des artefacts des systèmes logiciels

Applications

- UML peut être appliqué dans plusieurs domaines d'application (e.g. banque, finance, infrastructure, médecine ou télécommunication)
- UML peut être utilisé avec toutes les principales méthodes de développement de logiciels d'objets, et de composants et pour diverses plates-formes de mise en œuvre (e.g. J2EE, .NET)

Mauvaise définition

- UML est un langage de modélisation standard
- UML n'est pas un processus de développement logiciel

Processus de développement logiciel (PDL)

Le PDL consiste à:

- donner des conseils sur l'ordre des activités d'une équipe
- spécifier quels artefacts doivent être développés
- diriger les tâches des développeurs
- proposer des critères de suivi et de mesure des produits et activités d'un projet

UML et PDL

- UML est indépendant du PDL et pourrait être appliqué dans le contexte de différents processus
- UML est le plus adapté aux processus de développement itératifs et incrémentiels basés sur des cas d'utilisation

UML et OMG

- Object Management Group (OMG)
- OMG est un consortium de normes de l'industrie informatique
- les groupes de travail de l'OMG développent des normes d'intégration d'entreprise pour une gamme de technologies
- OMG: organisation mondiale créée en 1989 pour standardiser le modèle objet
- **UML est une norme maintenue par l'OMG**

Autres méthodes orientées objet précédant UML

- Booch: présentée par Grady Booch
- Object Modeling Technique (OMT): introduite par James Rumbaugh
- Object-Oriented Software Engineering (OOSE): proposee par Ivar Jacobson
- Object-Oriented Analysis (OOA)
- Object-Oriented Design (OOD)
- ...

Genèse de UML

- Booch, Rumbaugh et Jacobson commencent à travailler sur une méthode unifiée (1995)
- création d'un consortium de partenaires pour travailler sur la définition d'UML (1996)
- normalisation de la méthode UML 1.1 par l'OMG (1997)

Differente version d'UML ´

- UML 1.1 (1997)
- UML 1.2 (1998)
- UML 1.3 (1999)
- UML 1.4 (2001)
- UML 1.5 (2003)
- UML 2.0 (2005)
- UML 2.1 (2006)
- UML 2.2 (2009)
- UML 2.3 (2010)
- UML 2.4 (2011)
- UML 2.5 (2015)
- UML 2.5 (2017)

Documentation

(1) Priestley, Mark. *Practical object-oriented design with UML*. Mcgraw Hill Higher Education, 2003.

(2) Roques, Pascal. *UML in practice: the art of modeling software systems demonstrated through worked examples and solutions*. John Wiley & Sons, 2006.

(3) https://www.tutorialspoint.com/uml/index.htm

Diagrammes UML

UML définit deux grands types de diagramme:

- diagrammes de structure (diagrammes statiques)
- diagrammes de comportement (diagrammes dynamiques)

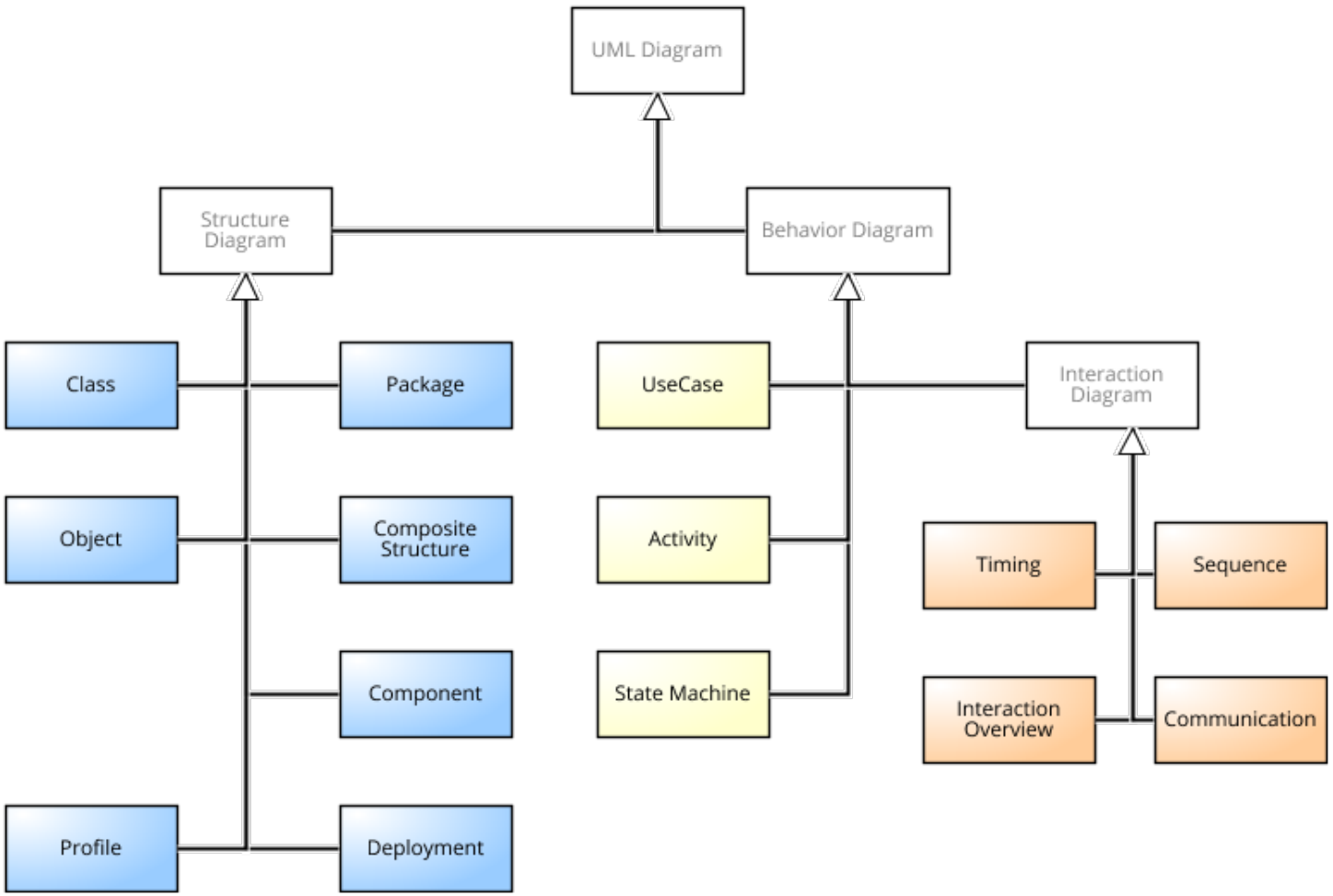
Diagrammes de structure

- la structure statique du système et de ses parties à différents niveaux d'abstraction, et d'implémentation et comment ils sont liés les uns aux autres
- les éléments d'un diagramme de structure représentent les concepts significatifs d'un système et peuvent inclure des concepts abstraits, réels et d'implémentation
- il existe au moins 7 diagrammes de structure (UML 2.5)

Diagrammes de comportement

- le comportement dynamique des objets d'un système, décrit comme une série de modifications du système au fil du temps
- il existe au moins 7 diagrammes de comportement (UML 2.5)

Diagrammes UML



Diagrammes de structure

(1) Diagramme de classe

- définit la structure du système conçu en tant que *classes* et *interfaces* associées, avec leurs caractéristiques, contraintes et relations (e.g. associations, généralisations et dépendances)
- exemple
  - classes: **étudiant**, **cours** et **département**
  - caractéristiques: étudiant (matricule, nom, promotion, etc.)
  - relations: **suivre** (étudiant suit un cours), **être du** (étudiant est d'un département)

Diagrammes de structure

(2) Diagramme d'objet

- définit les spécifications d'instance de classes et d'interfaces
- définit aussi des spécifications des associations
- exemple
  - objet: **étudiant Jonathan**, **cours de POO** et **département de réseaux**

Diagrammes de structure

(3) Diagramme de paquetage

- définit les paquetages du système ainsi que leurs relations
- paquetage fournit l'abstraction d'information par le groupement de fichiers du logiciel logiquement associés
- exemple
  - paquetage ressources humaines: étudiant, professeur, assistants, administratifs, etc.
  - paquetage équipements: ordinateur, véhicules, imprimantes, etc.

Diagrammes de structure

(4) Diagramme de composants

- définit la représentation des composants du système d'un point de vue physique tels qu'ils sont mis en œuvre
- exemple
  - fichiers
  - bibliothèques
  - bases de données

Diagrammes de structure

(5) Diagramme de déploiement

- définit la représentation des éléments matériels
- et l'architecture du système en tant que déploiement (distribution) d'artefacts logiciels vers des cibles de déploiement
- exemple
  - ordinateurs
  - périphériques
  - réseaux
  - systèmes de stockage

Diagrammes de structure

(6) Diagramme de profils

- permet de définir des stéréotypes personnalisés, des valeurs étiquetées et des contraintes en tant que mécanisme d'extension léger au standard UML
- les profils permettent d'adapter le méta-modèle UML à différentes plates-formes (telles que J2EE ou .NET), ou domaines (tels que la modélisation en temps réel ou des processus métier)

Diagrammes de structure

(7) Diagramme de structure composite

- définit la représentation sous forme de boîte blanche des relations entre composants d'une classe

Diagrammes de comportement

(1) Diagramme de cas d'utilisation

- définit la représentation des possibilités d'interaction entre le système et les acteurs (intervenants extérieurs au système)
- décrit un ensemble d'actions (cas d'utilisation) que certains système ou systèmes (sujet) peuvent effectuer en collaboration avec un ou plusieurs utilisateurs externes du système (acteurs) pour fournir des résultats observables
- exemple
  - enregistrer un nouveau étudiant

Diagrammes de comportement

(2) Diagramme d'activité

- définit la séquence et les conditions de coordination des comportements de niveau inférieur, plutôt que les classificateurs qui possèdent ces comportements
- ceux-ci sont communément appelés modèles de flux de contrôle et de flux d'objets
- exemple
  - paiement de frais par un étudiant: recherche de l'étudiant, vérification de son compte, validation du paiement et livraison de preuve de paiement

Diagrammes de comportement

(3) Diagramme états-transitions

- utilisé pour modéliser un comportement discret via des transitions d'états finis
- exemple
  - états étudiant: appliquant $\rightarrow$ admis $\rightarrow$ junior $\rightarrow$ senior $\rightarrow$ licencié

Diagrammes de comportement

(4) Diagramme d'interaction

- diagramme de séquence
- diagramme de communication
- diagramme global d'interaction
- diagramme de temps

Diagrammes de comportement

(4.1) Diagramme de séquence

- se concentre sur l'échange de messages entre les éléments du système ou de ses acteurs
- exemple: restaurant
  - Jordan commande la nourriture au serveur
  - le serveur à son tour commande la nourriture au cuisiner
  - le serveur sert de l'eau à Jordan
  - le cuisinier remet la nourriture au serveur
  - le serveur sert la nourriture à Jordan
  - Jordan paie au caissier

Diagrammes de comportement

(4.2) Diagramme de communication

- se concentre sur les échanges de messages entre les objets
- représentation simplifiée d'un diagramme de séquence
- exemple: impression d'un document
  - Jordan se connecte à un ordinateur comme utilisateur
  - il choisit le document à imprimer
  - l'ordinateur envoie le document choisi à l'imprimante

Diagrammes de comportement

(4.3) Diagramme global d'interaction

- définit les interactions via une variante de diagrammes d'activités d'une manière qui favorise la vue d'ensemble du flux de contrôle

Diagrammes de comportement

(4.4) Diagramme de temps

- définit les interactions lorsqu'un objectif principal du diagramme est de raisonner sur le temps
- montre les changements d'état d'un objet quand ceux-ci dépendent exclusivement du temps
- indique la durée minimale ou maximale de chaque état à l'aide de contraintes temporelles

Quelques logiciels gratuits pour faire UML

- **StarUML** (UML 2.x): à utiliser dans ce cours
- ArgoUML (UML 1.4)
- BoUML (UML 2.x)
- ...

Cas d'utilisation: Gestion d'une bibliothèque

Problématique

- la méthode de gestion de la bibliothèque par l'outil Microsoft Excel est moins efficace
- Microsoft Excel n'est pas un système de gestion de base de données, donc inapproprié pour une gestion efficace d'une bibliothèque
- par exemple
  - Microsoft Excel ne permet pas un système de rappel des dates de retour d'emprunts déjà à terme
  - inefficacité dans la collection des ouvrages perdus
  - manque de confidentialité des données
  - manque de pérennité
  - etc.

Cas d'utilisation: Gestion d'une bibliothèque

Les besoins fonctionnels

- Le système doit permettre au bibliothécaire de gérer :
  - les ouvrages (e.g. consulter, supprimer ou modifier, ou vérifier la disponibilité des exemplaires)
  - des membres (e.g. ajouter, supprimer ou modifier);
  - des emprunts (e.g. consulter, supprimer, modifier ou rappeler, ou définir la durée des emprunts, définir le max ouvrage à emprunter, consulter les ouvrages les plus sollicités, l'ouvrage le moins emprunté ou les nouveautés)
  - des pénalités (e.g. ajouter ou supprimer une pénalité, consulter la liste des pénalités ou définir la durée de la pénalité)
- Le système doit permettre au membre :
  - d'effectuer une recherche ou de réserver un ouvrage

Cas d'utilisation: Gestion d'une bibliothèque

Les besoins non-fonctionnels...

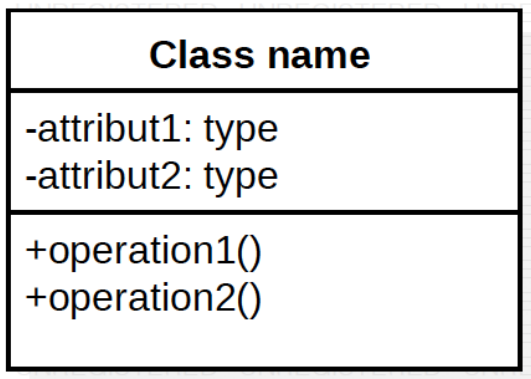
- Extensibilité
- Fonctionnel en réseau
- Portabilité
- Sécurité

TD 1: Gestion d'une bibliothèque

- énumérer les potentielles classes
- énumérer les potentiels attributs de chaque classe
- énumérer les potentielles opérations de chaque classe
- énumérer les potentielles relations entre classes
- **deadline: Mardi 16 Février 2021, 13h30**

(1) Diagramme de classe (Class diagram)

- définit la structure du système conçu en tant que *classes* et *interfaces* associées, avec leurs caractéristiques, contraintes et relations (i.e. associations, généralisations, dépendances, etc)



- **visibilité**: + (public), - (privé), # (protégé), ˜(package)
- **multiplicité**: 1, 1..*, 0..*, *
- **navigabilité**: indique le sens d'une relation (bidirectionnelle, unidirectionnelle et direction interdite)

(1) Diagramme de classe (Class diagram)

- classe ou méthode abstraite: nom de classe ou méthode en *italique*
- attribut ou méthode statique: nom souligné

Contraintes d'une association UML

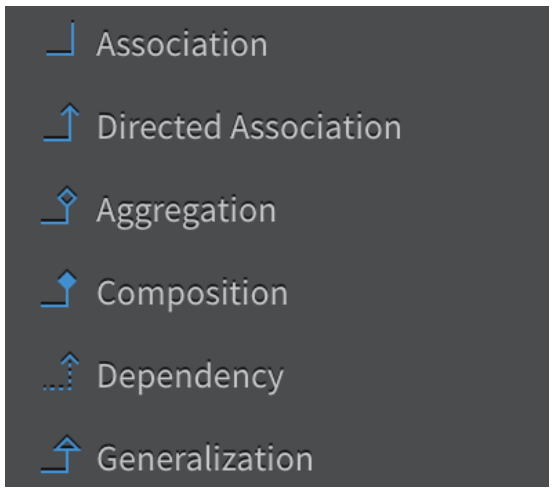
- {**implicit**}: indique que la relation n'est pas manifeste]
- {**ordered**}: indique que l'ensemble des objets à une extrémité d'une association sont d'une certaine manière spécifique
- {**changeable**}: indique que la connexion entre divers objets du système peut être ajoutée, supprimée et modifiée selon l'exigence

Contraintes d'une association UML

- {**addOnly**}: indique que les nouvelles connexions peuvent être ajoutées à partir d'un objet qui est situé à l'autre extrémité d'une association
- {**frozen**}: indique que lorsqu'un lien est inséré entre deux objets, il ne peut pas être modifié tant que la contrainte figée est active sur le lien donné
- {**readOnly**}: indique que qu'une valeur ne peut pas être modifiée directement mais peut changer en raison d'un changement d'une autre valeur
- exemple, si une personne a une date de naissance et un âge, l'âge peut être en lecture seule, mais il ne peut pas être figé

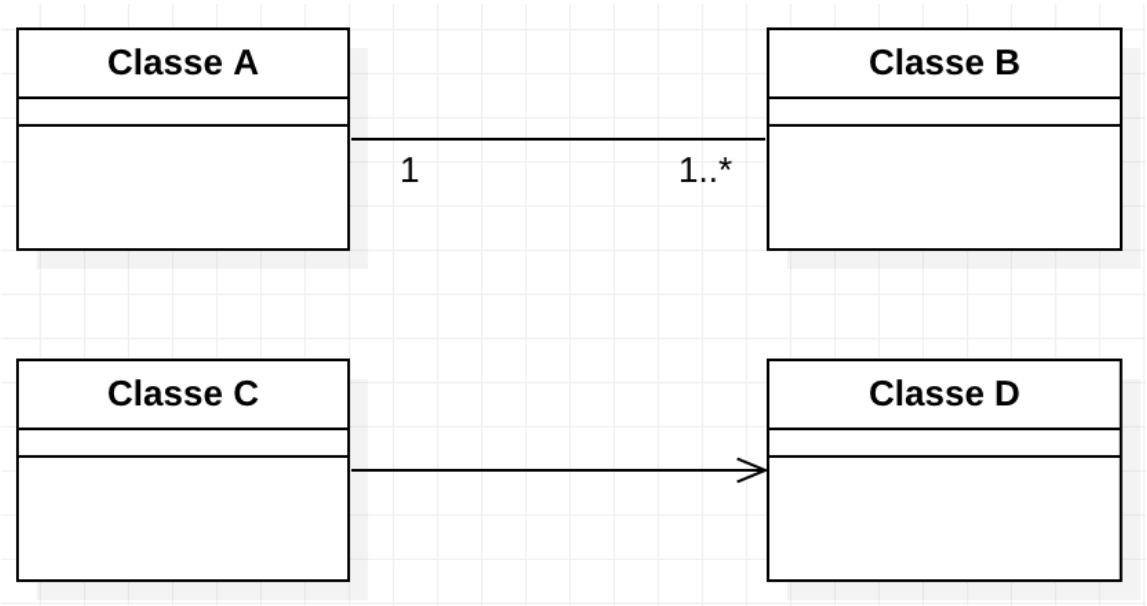
Relations

- association
- association directe
- agrégation
- composition
- dépendance
- généralisation



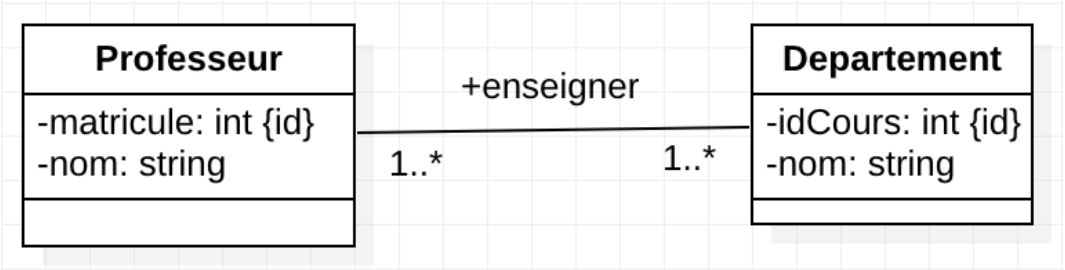
Relations: association et association directe

- une connexion sémantique entre deux classes (relation logique)
- association: relation bidirectionnelle
- association directe: association unidirectionnelle



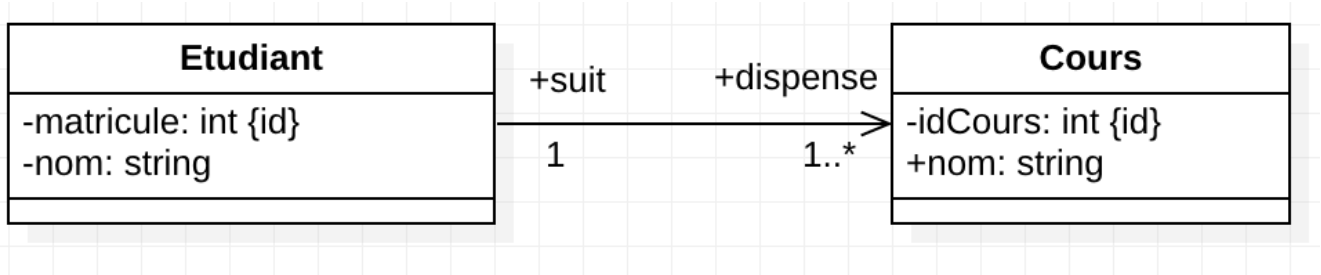
- exemple: étudiant et département

Association en Java




```
public class Professeur
{
  private int  matricule;
  private String nom;
  List<Departement> departements;
}
public class Departement
{
  private int  idDepartement;
  private String nom;
  private List<Professeur> professeurs;
}
```

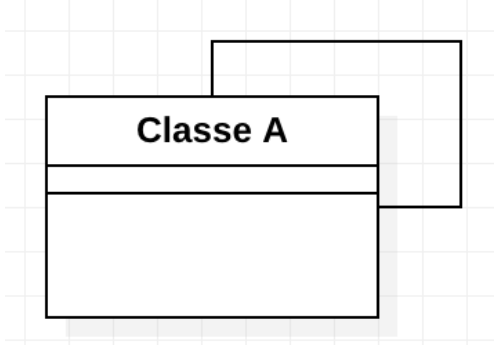
Association directe en Java




```
public class Etudiant
{
  private int  matricule;
  private String nom;
  private List<Cours> cours;
}
public class Cours
{
  private int  idCours;
  private String nom;
}
```

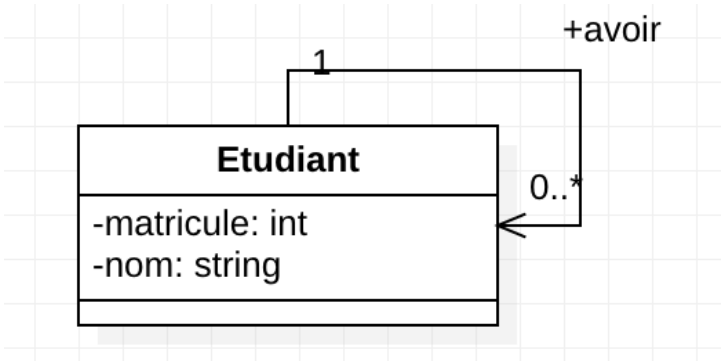
Relations: association réflexive

- est une association d'une classe à la même classe



- exemple: catégorie d'un produit

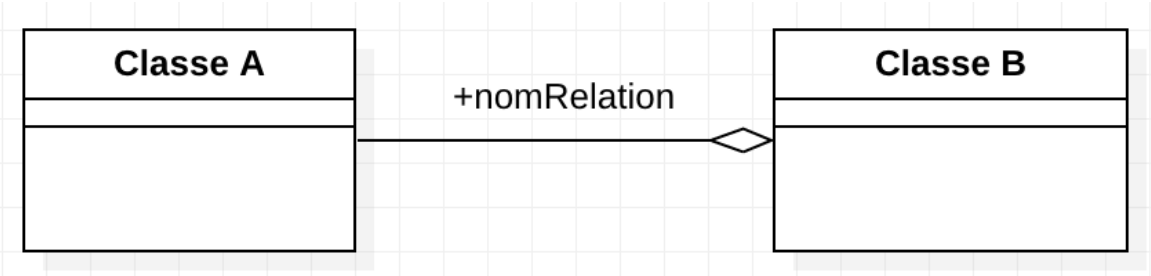
Association réflexive en Java




```
public class Etudiant
{
  private int  matricule;
  private String nom;
  private List<Etudiant> amis;
}
```

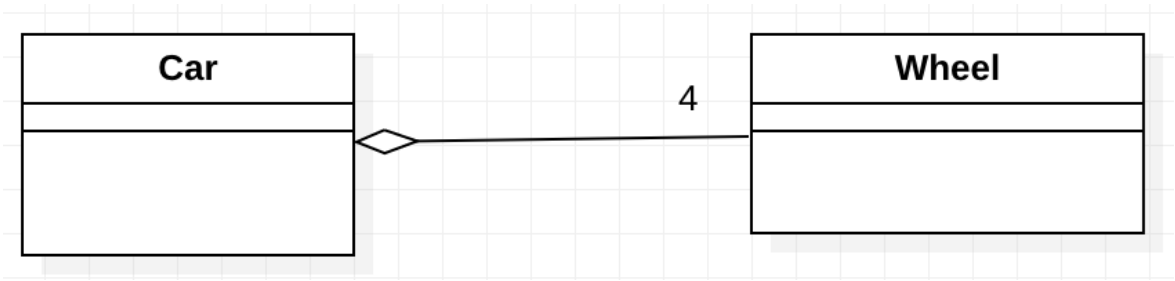
Relations: agrégation

- est une association avec relation de subordination entre deux classes
- classe A est une instance de la classe B
- destruction de la classe mère n'implique pas nécessairement la destruction de la classe composée
- classe composée: classe A
- classe mère: classe B



- exemple: maison et murs

Agrégation en Java



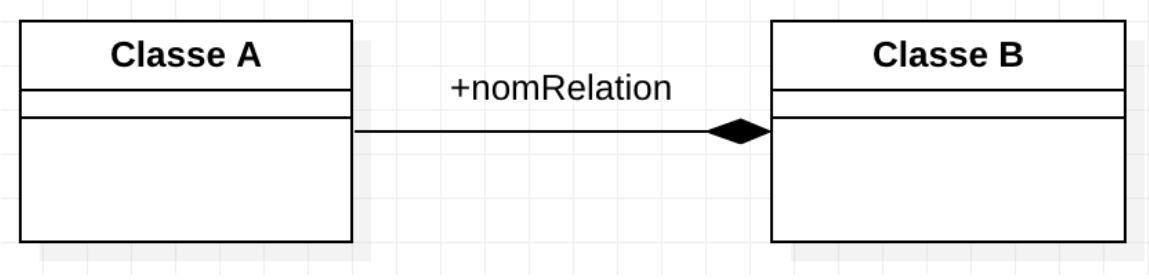

```
public class Car
{
  private List<Wheel> wheels;
}
public class Wheel
{

}
```

Relations: agrégation vs association

- il est parfois difficile de distinguer l'implémentation d'une association de celle d'une agrégation
- la distinction entre les deux associations est logique

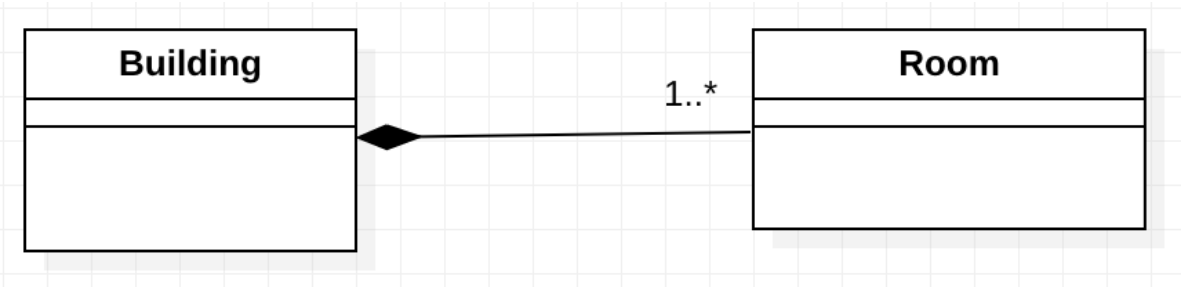
Relations: composition

- est une agrégation avec cycle de vie dépendant
- est une agrégation forte
- la classe composée est détruite lorsque la classe mère disparait
- classe composée: classe A
- classe mère: classe B



- exemple: un dossier et fichiers contenus dans le dossier

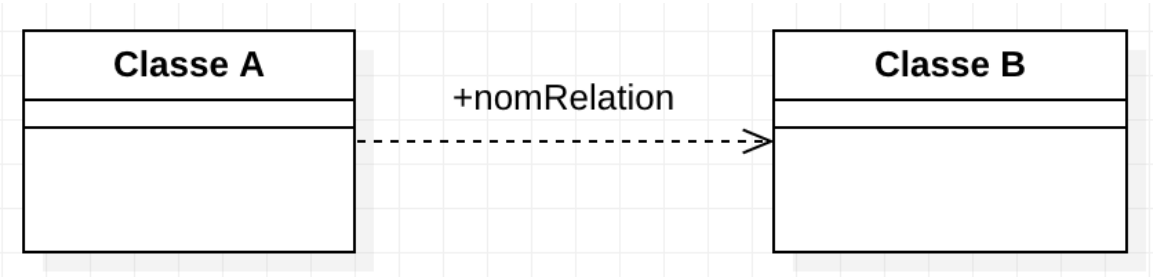
Composition en Java



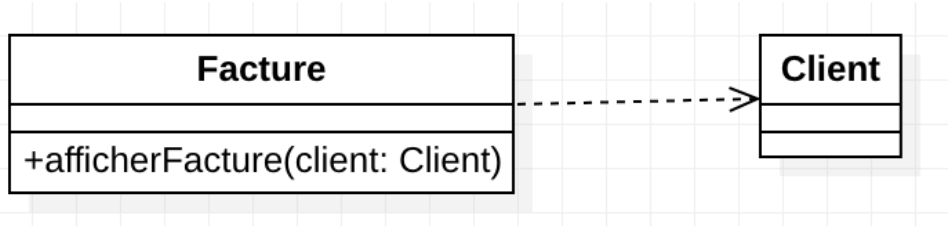

```
public class Building
{
  private List<Room> rooms;

  public class Room
  {

  }
}
```

Relations: dépendance

- implique qu'une ou plusieurs méthodes reçoivent un objet d'un type d'une autre classe



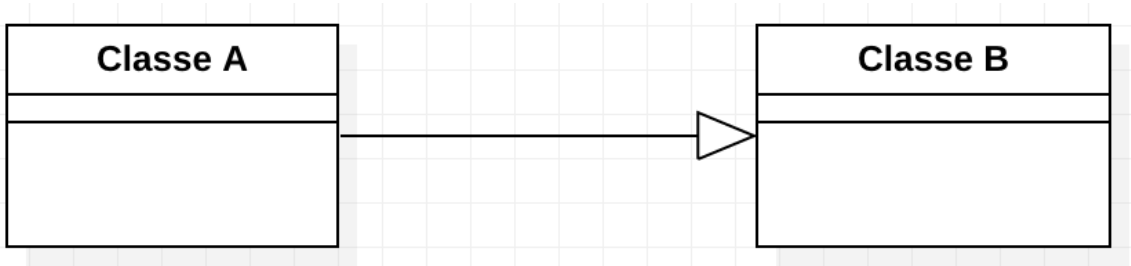
Dépendance en Java



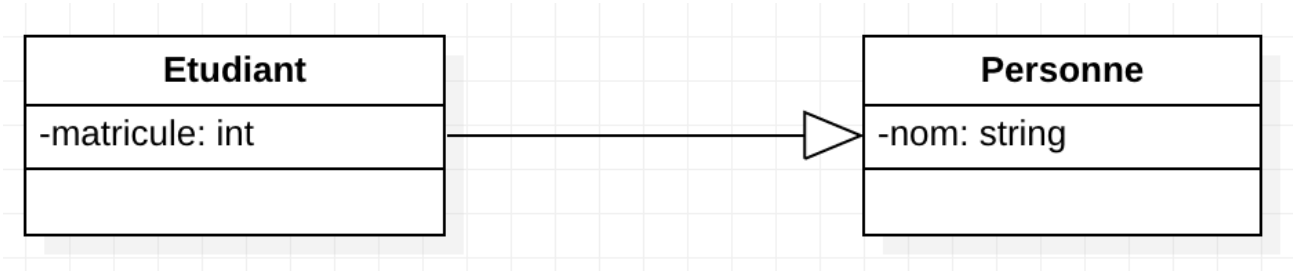

```
public class Facture
{
  public void afficherFacture(Client client){

  }

}
public class Client
{

}
```

Relations: généralisation

- représente le concept d'héritage entre deux classes
- classe de base: classe B
- sous classe: classe A
- classe de base généralise la sous classe
- la sous classe hérite la classe de base



Généralisation en Java




```
public class Personne
{
  private String nom;

}
public class Etudiant extends Personne
{
  private int matricule;
}
```

Bibliothèque: Potentielles classes et relations

Potentielles classes (en rouge) et relations (en bleu)

- la méthode de gestion de la bibliothèque par l'outil Microsoft Excel est moins efficace
- Microsoft Excel n'est pas un système de gestion de base de données, donc inapproprié pour une gestion efficace d'une bibliothèque
- par exemple
  - Microsoft Excel ne permet pas un système de rappel des dates de retour d'emprunts déjà à terme
  - inefficacité dans la collection des ouvrages perdus
  - manque de confidentialité des données
  - manque de pérennité
  - etc.

Bibliothèque: Potentielles classes et relations

Potentielles classes (en rouge) et relations (en bleu)

- Le système doit permettre au bibliothécaire de gérer :
  - les ouvrages (e.g. consulter, supprimer ou modifier, ou vérifier la disponibilité des exemplaires)
  - des membres (e.g. ajouter, supprimer ou modifier);
  - des emprunts (e.g. consulter, supprimer, modifier ou rappeler, ou définir la durée des emprunts, définir le max ouvrage à emprunter, consulter les ouvrages les plus sollicités, l'ouvrage le moins emprunté ou les nouveautés)
  - des pénalités (e.g. ajouter ou supprimer une pénalité, consulter la liste des pénalités ou définir la durée de la pénalité)
- Le système doit permettre au membre :
  - d'effectuer une recherche ou de réserver un ouvrage

Bibliothèque: Potentielles classes

- bibliotheque
- bibliothécaire
- ouvrage
- exemplaire
- membre
- emprunt
- nouveauté
- pénalité
- recherche

Bibliothèque: Sélection des classes finales

- ~~bibliotheque~~, ~~bibliothécaire~~
- ouvrage
- exemplaire
- membre
- ~~emprunt~~
- ~~nouveauté~~
- pénalité
- ~~recherche~~
- auteur
- livre
- disque
- article
- compteur

Bibliothèque: Potentielles relations

- emprunter (membre et ouvrage)
- réserver (membre et ouvrage)
- effectuer (membre et ouvrage)
- subir (membre et pénalité)

Bibliothèque: Sélection des relations finales

Relations finales

- emprunter (membre et ouvrage)
- réserver (membre et ouvrage)
- ~~effectuer~~
- subir (membre et pénalité)
- réaliser (auteur et ouvrage)
- contenir (ouvrage et versions)
- habiter (membre et province)
- être dans (province et adresse)

Bibliothèque: Relations finales

- emprunter (membre et ouvrage): association avec propriétés
- réserver (membre et ouvrage): association avec propriétés
- subir (membre et pénalité): association
- réaliser (auteur et ouvrage): association
- contenir (ouvrage et versions): composition
- habiter (membre et province): association directe
- être dans (province et adresse): agrégation

Bibliothèque: Classe-association

Classe-association

association avec attributs et méthodes

- emprunt
- réservation

Bibliothèque: Attributs et méthodes de classes

Ouvrage

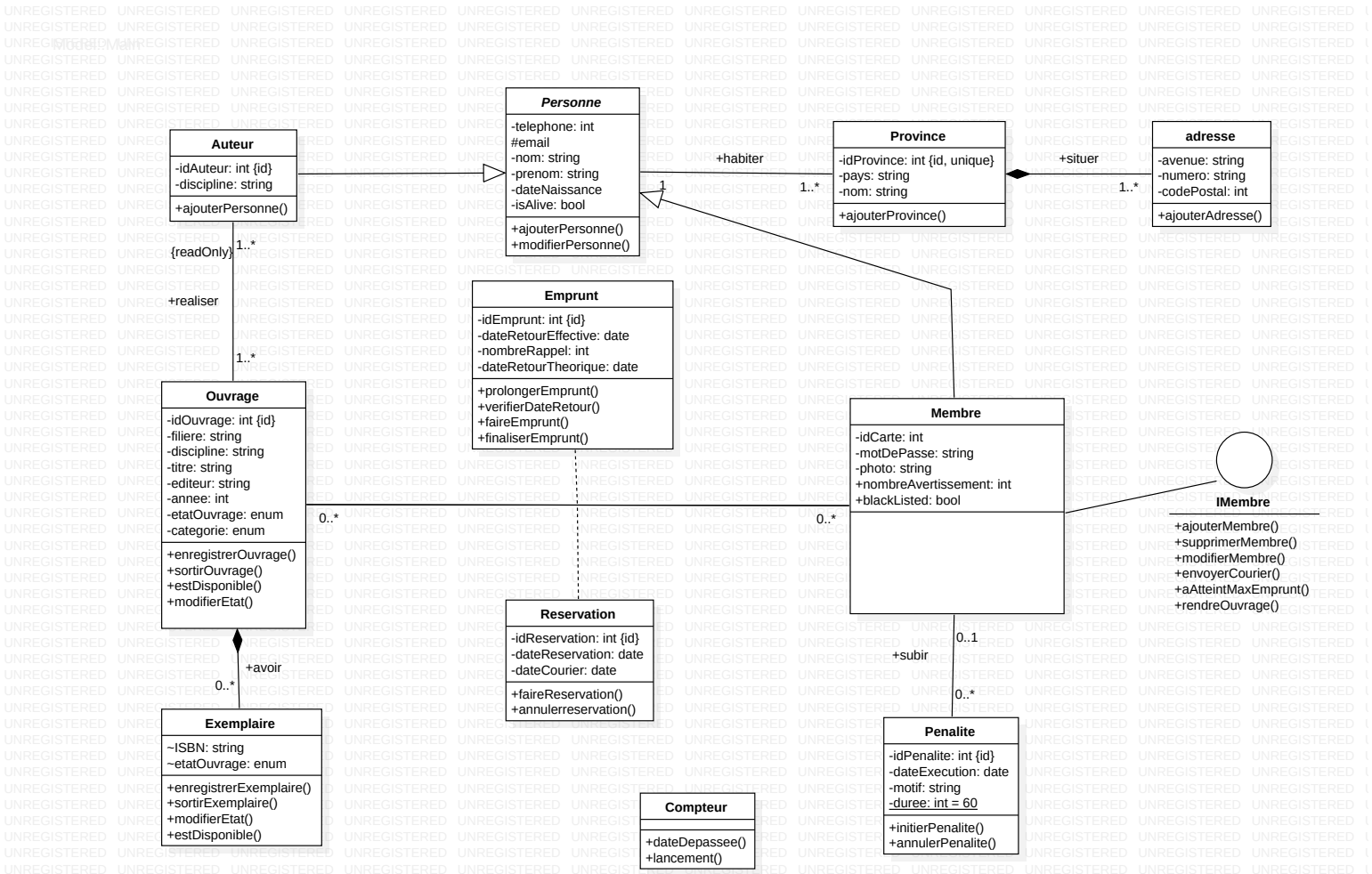
- attributs
  - idOuvrage
  - filière
  - discipline
  - etc.
- méthodes
  - enregistrerOuvrage()
  - sortirOuvrage()
  - estDisponible()
  - etc.

Bibliothèque: Attributs et méthodes de classes

Emprunt

- attributs
  - idEmprunt
  - dateRetourTheorique
  - dateRetourEffective
  - etc.
- méthodes
  - prolongerEmprunt()
  - faireEmprunt()
  - finaliserEmprunt()
  - etc.

Bibliothèque: Diagramme de classe



TD 2: Gestion d'une bibliothèque

- implémenter le diagramme de classe de la gestion d'une bibliotheque en Java
- **deadline: Mardi 17 Février 2021, 15h30**

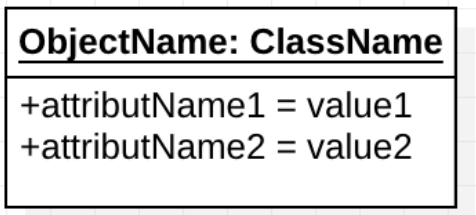
(2) Diagramme d'objet (Object diagram)

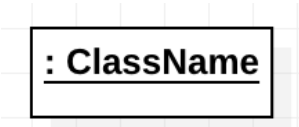
- un diagramme de classes représente un modèle abstrait composé de classes et de leurs relations
- un diagramme d'objets représente une instance à un moment particulier, qui est de nature concrète
  - le diagramme d'objets est plus proche du comportement réel du système
  - il capture la vue statique d'un système à un moment donné

(2) Diagramme d'objet (Object diagram)

- but du diagramme d'objets
  - relations d'objets d'un système
  - vue statique d'une interaction
  - comprendre le comportement des objets et leur relation d'un point de vue pratique

Notations

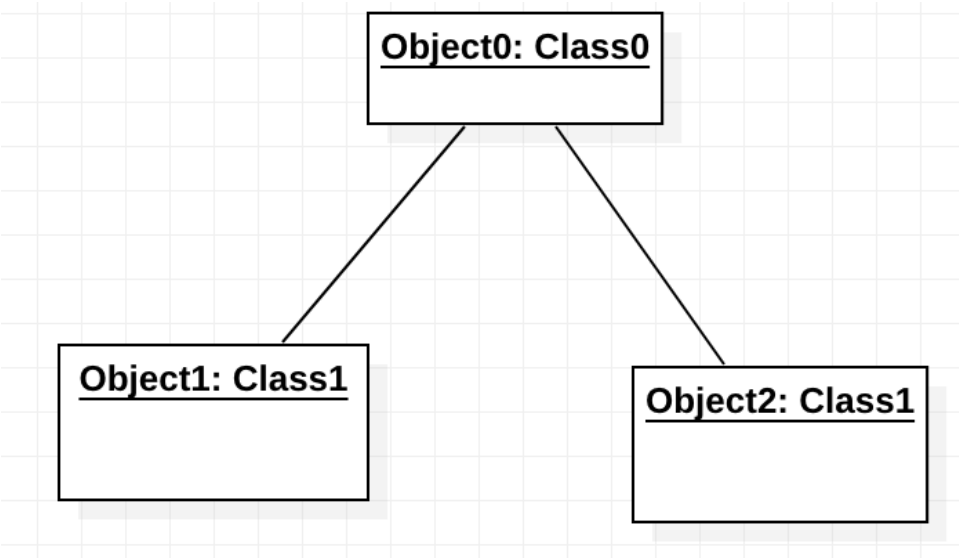

ObjectName: ClassName
+attributName1 = value1
+attributName2 = value2


: ClassName

Relations

- association bidirectionnelle
- association unidirectionnelle

Exemple



Exemple

Diagramme de classe

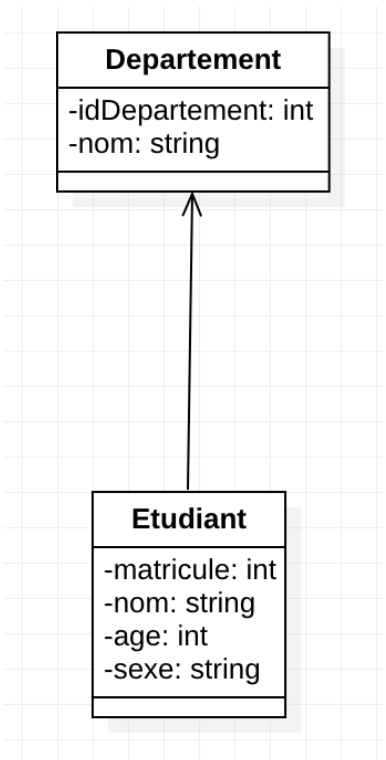


Diagramme d'objet

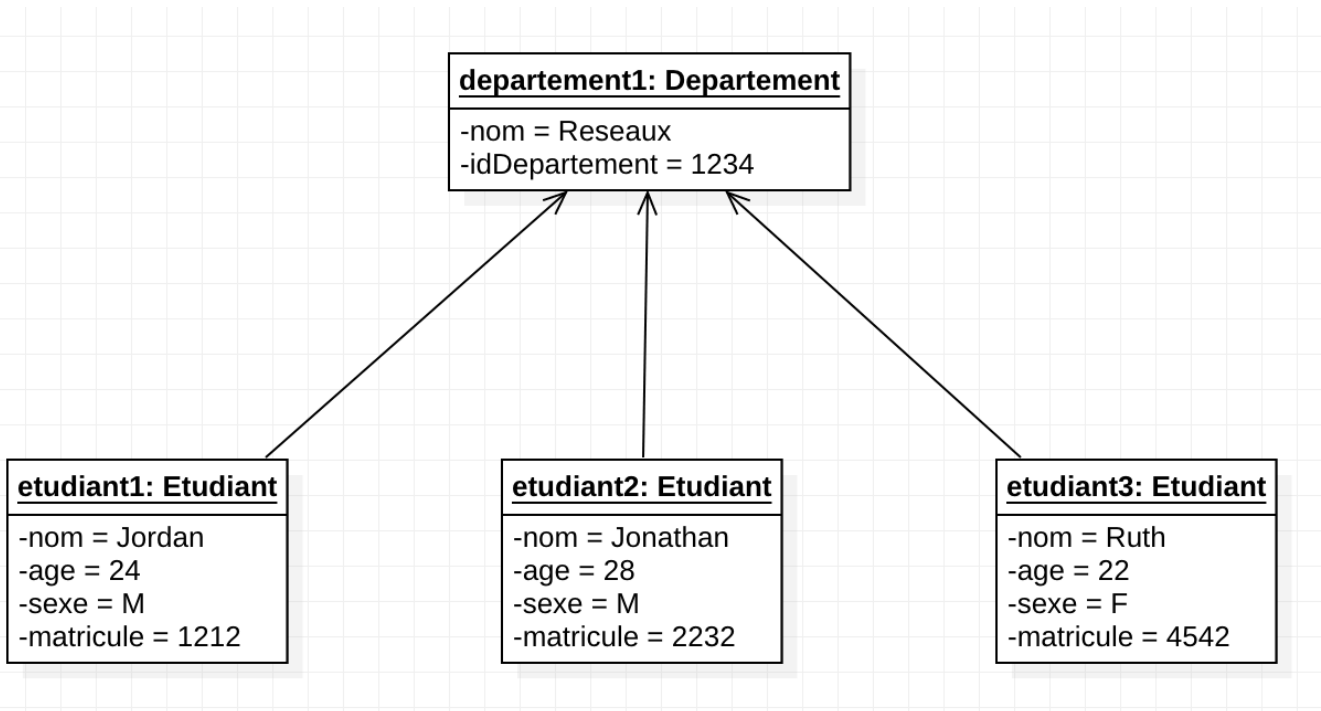
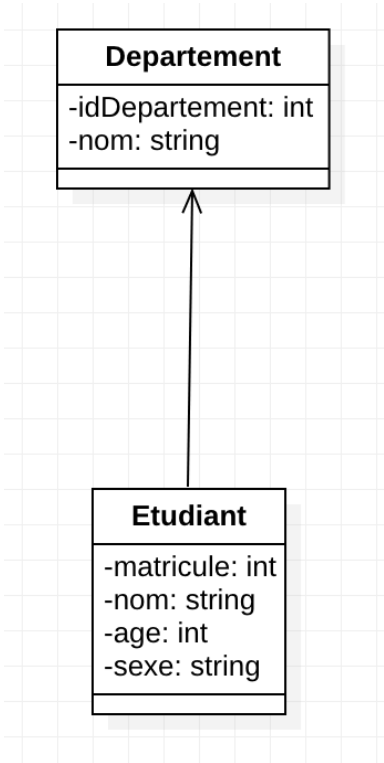
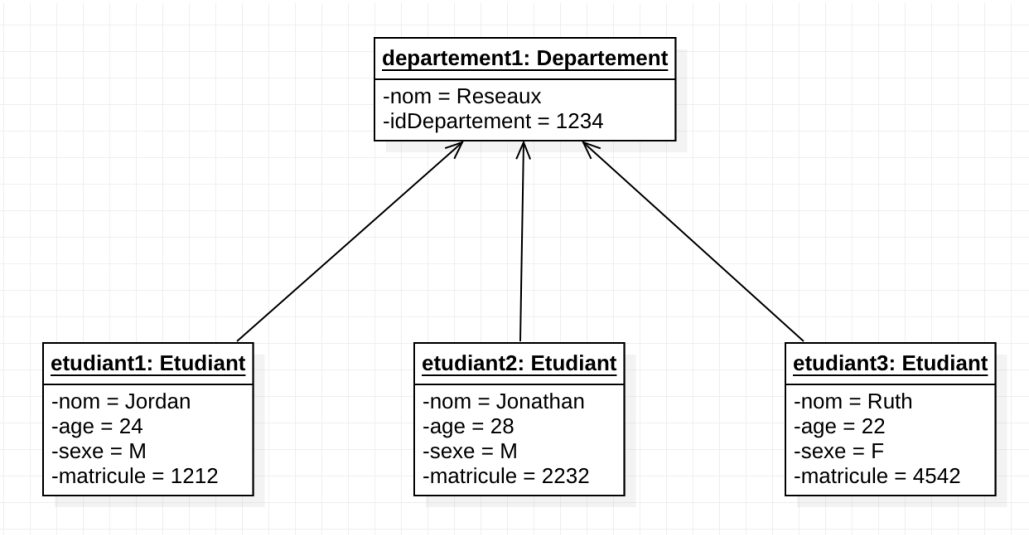


Diagramme d'objets en Java




```java
public class Departement
{
  private int idDepartement;
  private String nom;
  private List<Etudiant> etudiants;
}

public class Etudiant
{

}
```

Diagramme d'objets en Java




```
Departement departement1 = new Departement(1234, "reseaux");
Etudiant etudiant1 = new Etudiant(1212, "Jordan", "M", 24);
Etudiant etudiant2 = new Etudiant(2232, "Jonathn", "M", 28);
Etudiant etudiant3 = new Etudiant(4542, "Ruth", "F", 22);
```

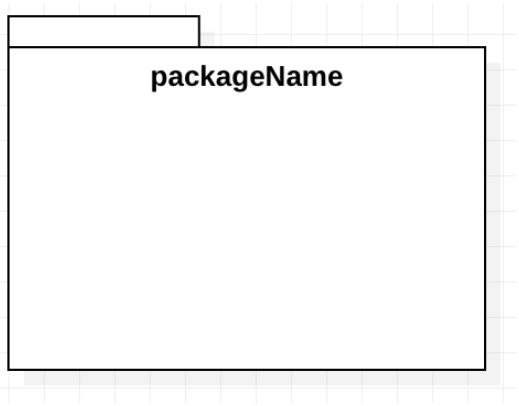
(3) Diagramme de paquetage (Package diagram)

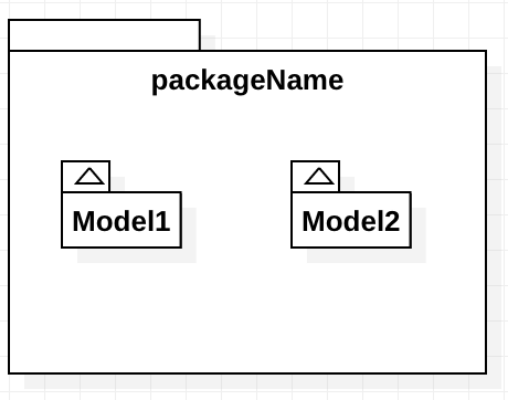
- définit les paquetages du système ainsi que leurs relations
- paquetage fournit l'abstraction d'information par le groupement de fichiers du logiciel logiquement associés
- exemple
  - paquetage ressources humaines: étudiant, professeur, assistants, administratifs, etc.
  - paquetage équipements: ordinateur, véhicules, imprimantes, etc.

(3) Diagramme de paquetage (Package diagram)

- utilisé dans les systèmes à grande échelle pour visualiser les dépendances entre les principaux éléments du système
- représente un mécanisme de regroupement au moment de la compilation
- organise un grand modèle
- regroupe des éléments liés
- sépare les espaces de noms
- un paquetage peut contenir de sous paquetages

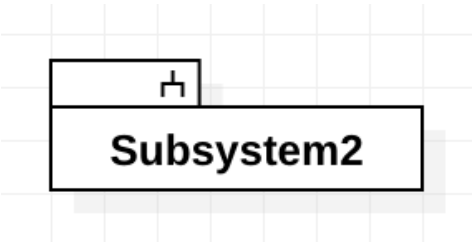
Notations





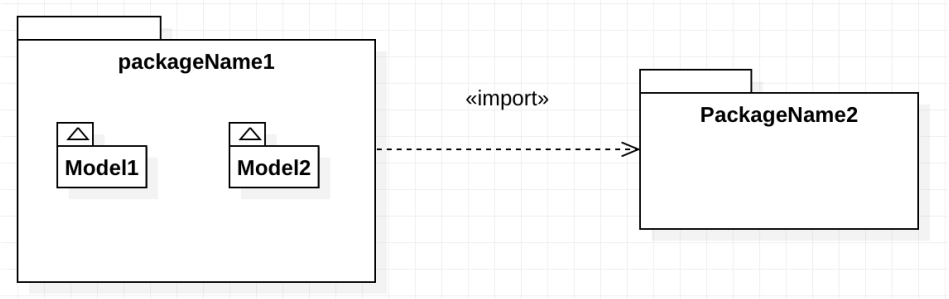
Sous systèmes

- utilisés pour la décomposition du système qui peut être représentée par les parties d'un système
- ont une spécification et une réalisation pour représenter deux vues
  - une vue externe: montrant les services fournis par le sous-système
  - une vue interne: montrant la réalisation du sous-système

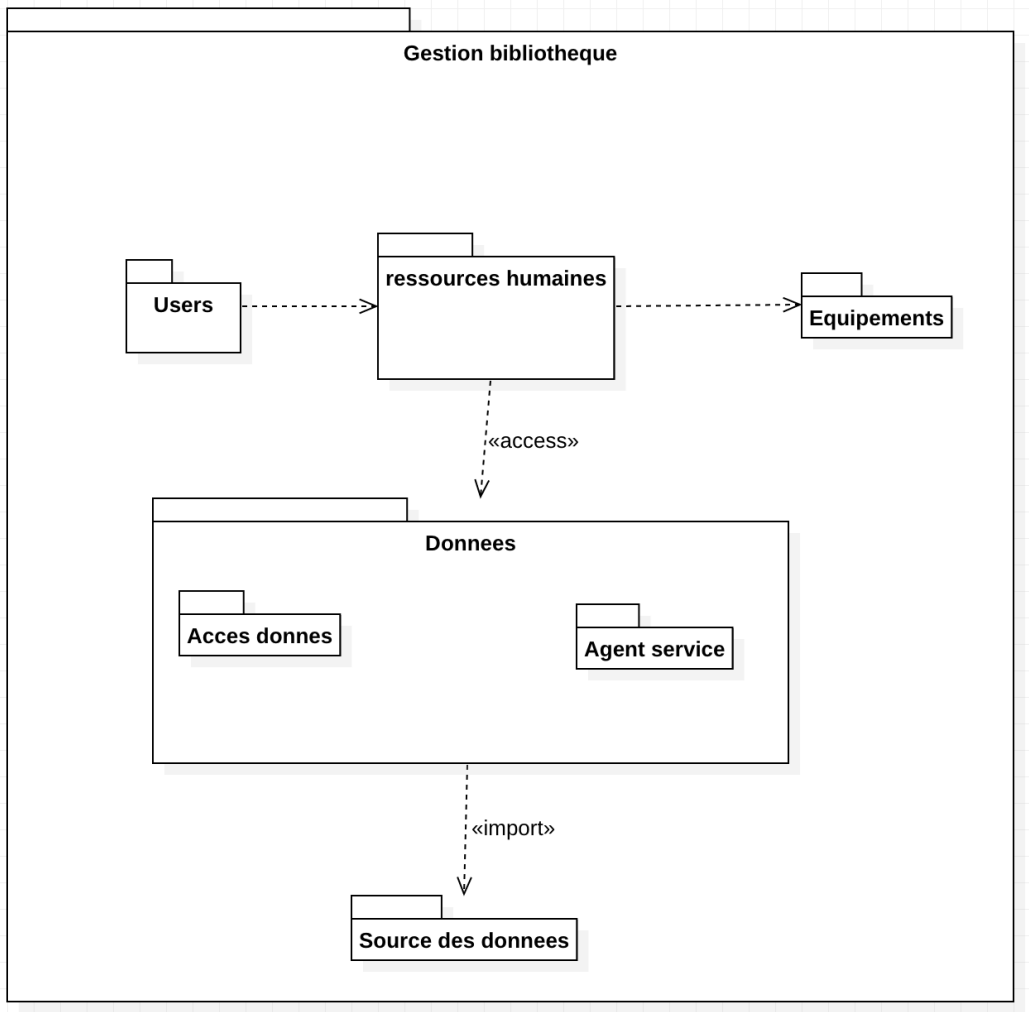


Relations

- *import*: paquetage cible est de l'extérieur
- *access*: paquetage cible est de l'intérieur
- *containment*: l'élément source contient l'élément cible
- *héritage*



Exemple


Gestion bibliotheque
Users
ressources humaines
Equipements
«access»
Donnees
Acces donnes
Agent service
«import»
Source des donnees

TD 3: Gestion d'une bibliothèque

- proposer le diagramme d'objet ainsi que le diagramme de paquetage pour la gestion de bibliotheque en StarUML
- implémenter les deux diagrammes en Java
- **deadline: Mardi 18 Février 2021, 13h30**

(4) Diagramme de composants (Component diagram)

- définit la représentation des composants du système d'un point de vue physique tels qu'ils sont mis en œuvre
- exemple
  - fichiers
  - bibliothèques
  - bases de données

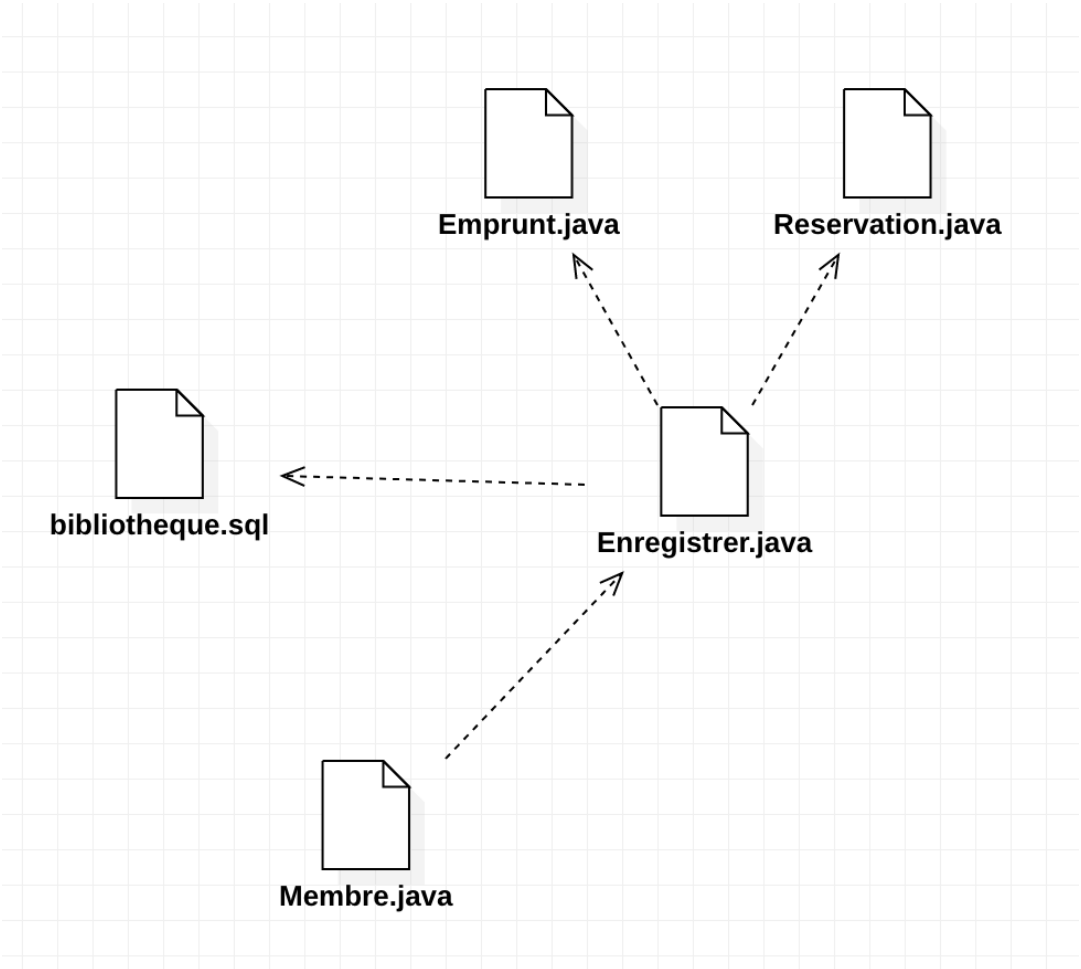
(4) Diagramme de composants

- est très importants du point de vue implémentation
- utilisé pour modéliser
  - les composants d'un système
  - le schéma de la base de données
  - les exécutables d'une application
  - le code source du système

(4) Diagramme de composants

- avant tout, les artefacts suivants doivent être clairement identifiés
  - fichiers utilisés dans le système
  - bibliothèques et autres artefacts pertinents pour l'application
  - relations entre les artefacts
- utiliser un nom significatif pour identifier les composants du système
- documenter les processus de mise en œuvre

Exemple


Emprunt.java
Reservation.java
bibliotheque.sql
Enregistrer.java
Membre.java

(5) Diagramme de déploiement (Deployment diagram)

- définit la représentation des éléments matériels
- et l'architecture du système en tant que déploiement (distribution) d'artefacts logiciels vers des cibles de déploiement
  - utilisé pour visualiser la topologie des composants physiques d'un système, où les composants logiciels sont déployés

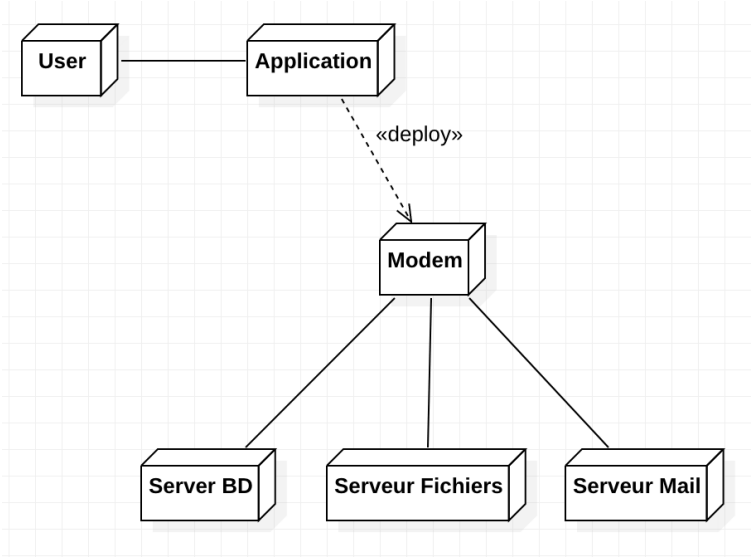
(5) Diagramme de déploiement

- constitué de nœuds et de leurs relations.
- exemple
  - ordinateurs
  - périphériques
  - réseaux
  - systèmes de stockage

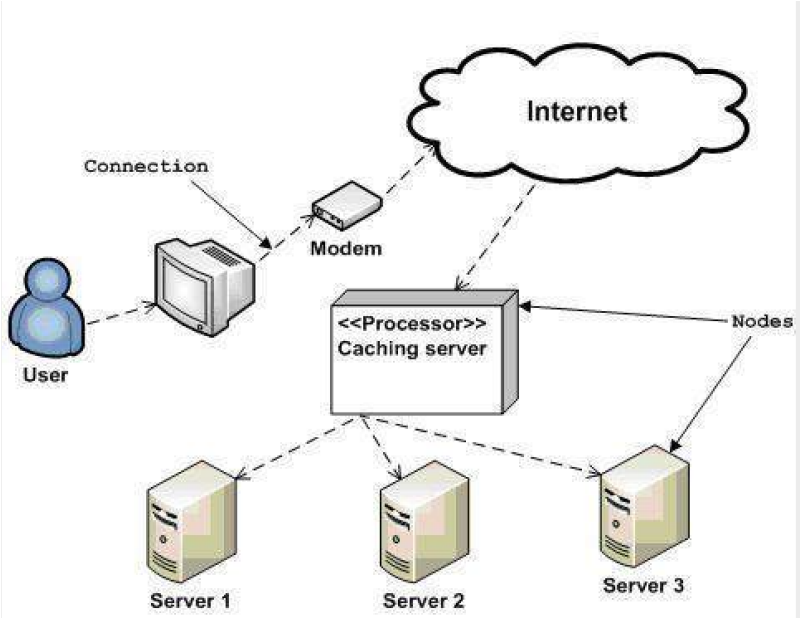
(5) Diagramme de déploiement

- objectifs
  - visualiser la topologie matérielle d'un système
  - décrire les composants matériels utilisés pour déployer les composants logiciels
  - décrire les nœuds de traitement d'exécution
- importance
  - performance
  - scalabilité
  - maintenance
  - portabilité

Exemple


User
Application
«deploy»
Modem
Server BD
Serveur Fichiers
Serveur Mail

Exemple


Internet
Connection
Modem
User
<<Processor>>
Caching server
Nodes
Server 1
Server 2
Server 3

TD 4: Gestion d'une bibliotheque

- décrire les diagrammes de structure composite et de profils avec détails
- proposer ces deux diagrammes pour la gestion de bibliotheque (en se servant de StarUML)
- **dealine: Lundi 22 Février 2021, 15h30**

Diagramme de comportement

(1) Diagramme de cas d'utilisation (Usecase diagram)

- définit la représentation des possibilités d'interaction entre le système et les acteurs (intervenants extérieurs au système)
- décrit un ensemble d'actions (cas d'utilisation) que certains système ou systèmes (sujet) peuvent effectuer en collaboration avec un ou plusieurs utilisateurs externes du système (acteurs) pour fournir des résultats observables
- exemple
  - enregistrer un nouveau étudiant

(1) Diagramme de cas d'utilisation

- utilisé pour rassembler les exigences d'un système
- utilisé pour obtenir une vue extérieure d'un système
- identifie les facteurs externes et internes qui influencent le système
- montre l'interaction entre les exigences des acteurs

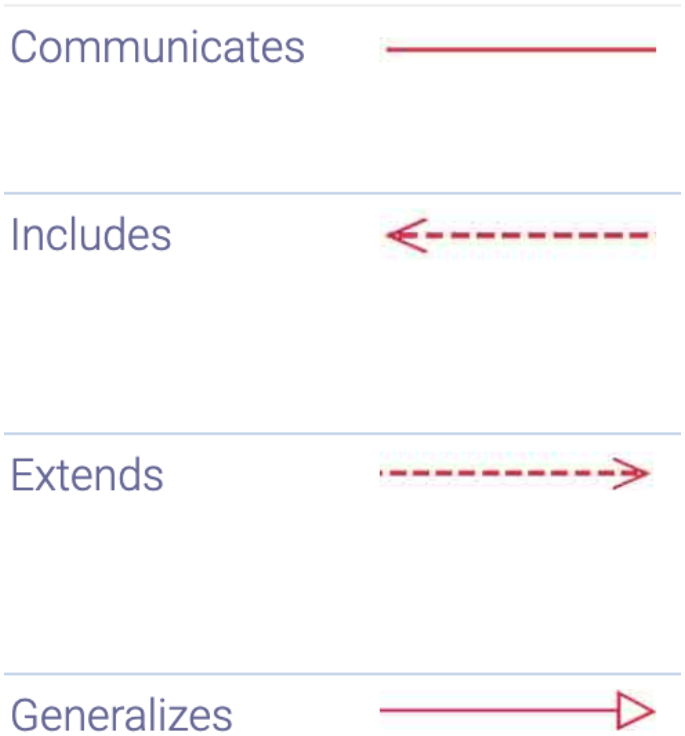
(1) Diagramme de cas d'utilisation

- les acteurs peuvent être un utilisateur humain, certaines applications internes ou certaines applications externes
- lorsque nous prévoyons de dessiner un diagramme de cas d'utilisation, nous devons identifier les éléments suivants
  - fonctionnalités à représenter comme cas d'utilisation
  - acteurs
  - relations entre cas d'utilisation et acteurs

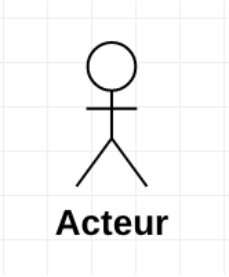
(1) Diagramme de cas d'utilisation

- relations
  - **extends**: une fonctionnalité gère les exceptions du cas d'utilisation de base
    - la flèche pointe du cas d'utilisation étendu vers le cas d'utilisation de base
  - **includes**: une fonctionnalité contient un comportement commun à plusieurs autres fonctionnalités
    - la flèche pointe vers la fonctionnalité courante
  - **generalises**: une fonctionnalité est plus générale qu'une autre fonctionnalité
    - la flèche pointe vers la fonctionnalité générale
  - **association**: un acteur est connecté à une fonctionnalité

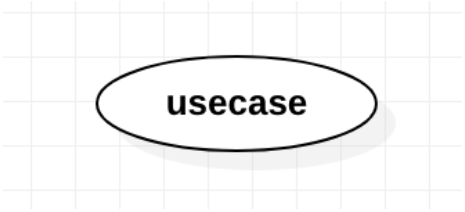
Relations



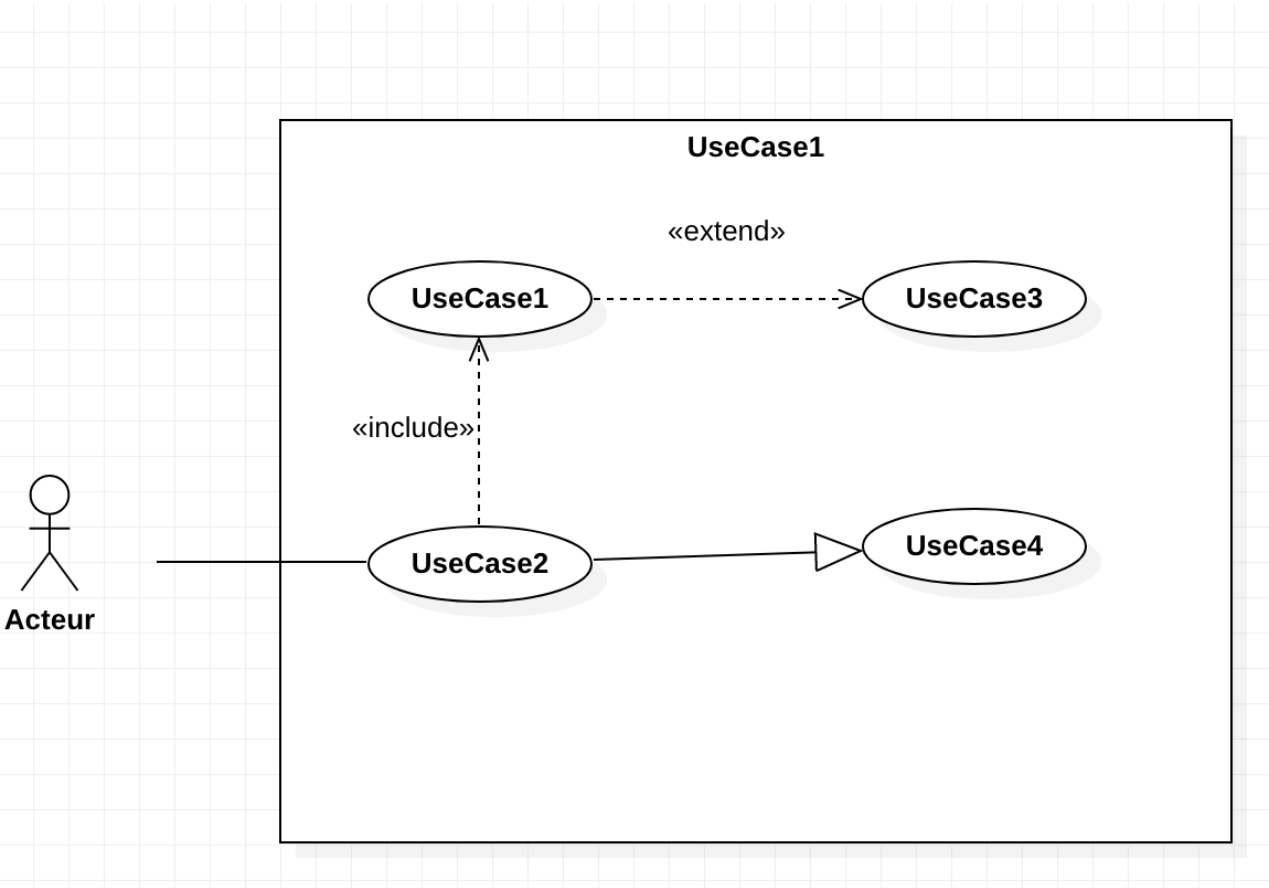
Acteur



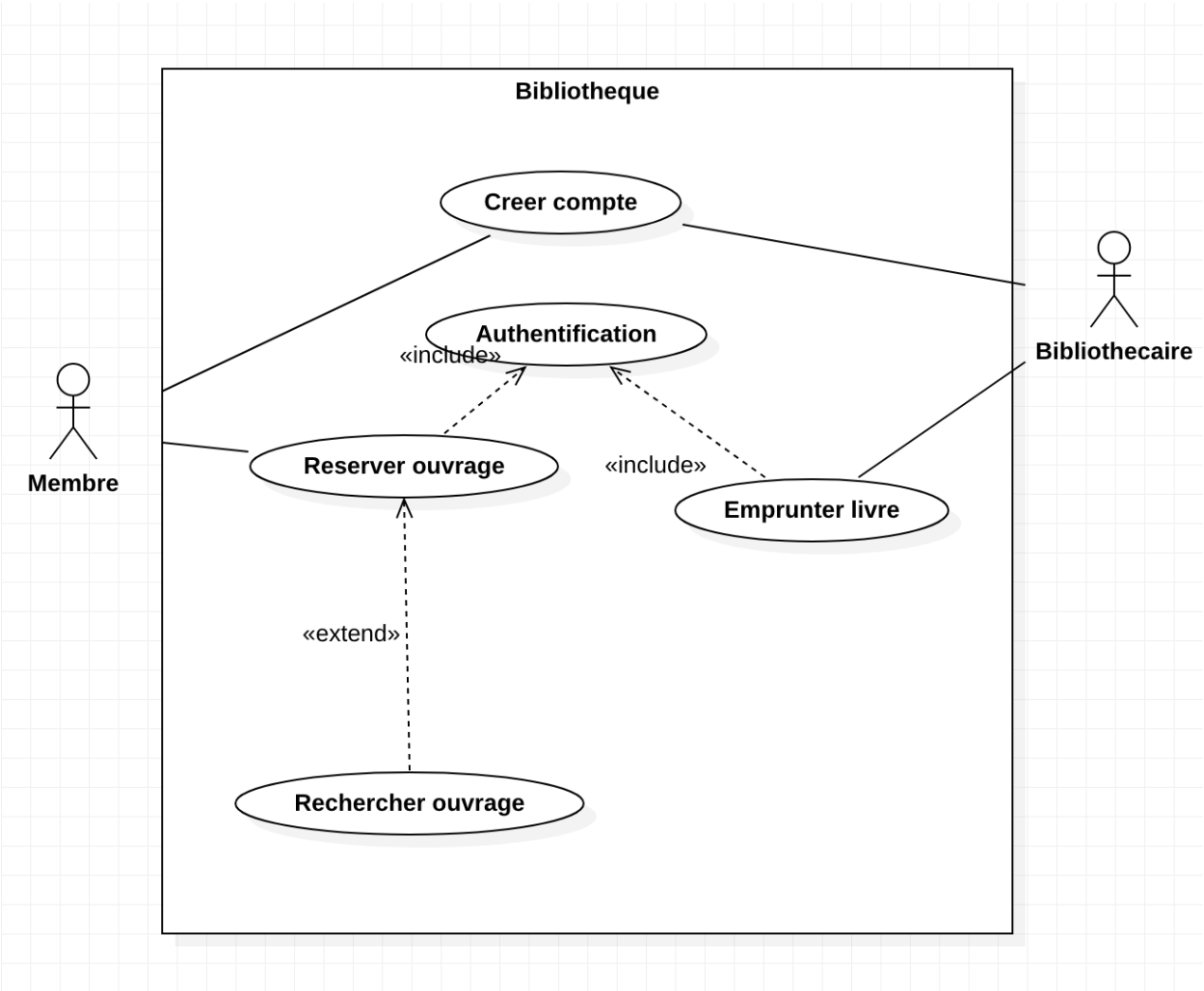
Usecase



Exemple


UseCase1
«extend»
UseCase1
UseCase3
«include»
Acteur
UseCase2
UseCase4

Exemple: Extrait Bibliotheque



(2) Diagramme d'activité (Activity diagram)

- définit la séquence et les conditions de coordination des comportements de niveau inférieur
- ceux-ci sont communément appelés modèles de flux de contrôle et de flux d'objets
- exemple
  - paiement de frais par un étudiant: recherche de l'étudiant, vérification de son compte, validation du paiement et livraison de preuve de paiement

(2) Diagramme d'activité (Activity diagram)

- montre différents flux d'un système
- le but d'un diagramme d'activités
  - dessiner le flux d'activité du système
  - décrire la séquence d'une activité à une autre
  - décrire le flux parallèle et concurrent du système

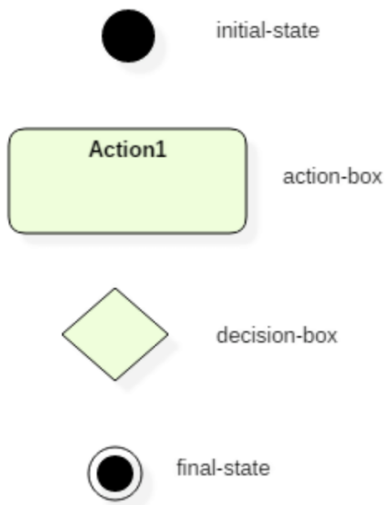
(2) Diagramme d'activité (Activity diagram)

- avant de dessiner un diagramme d'activités, les éléments suivants doivent être identifiés
  - activités
  - association
  - conditions
  - contraintes du système

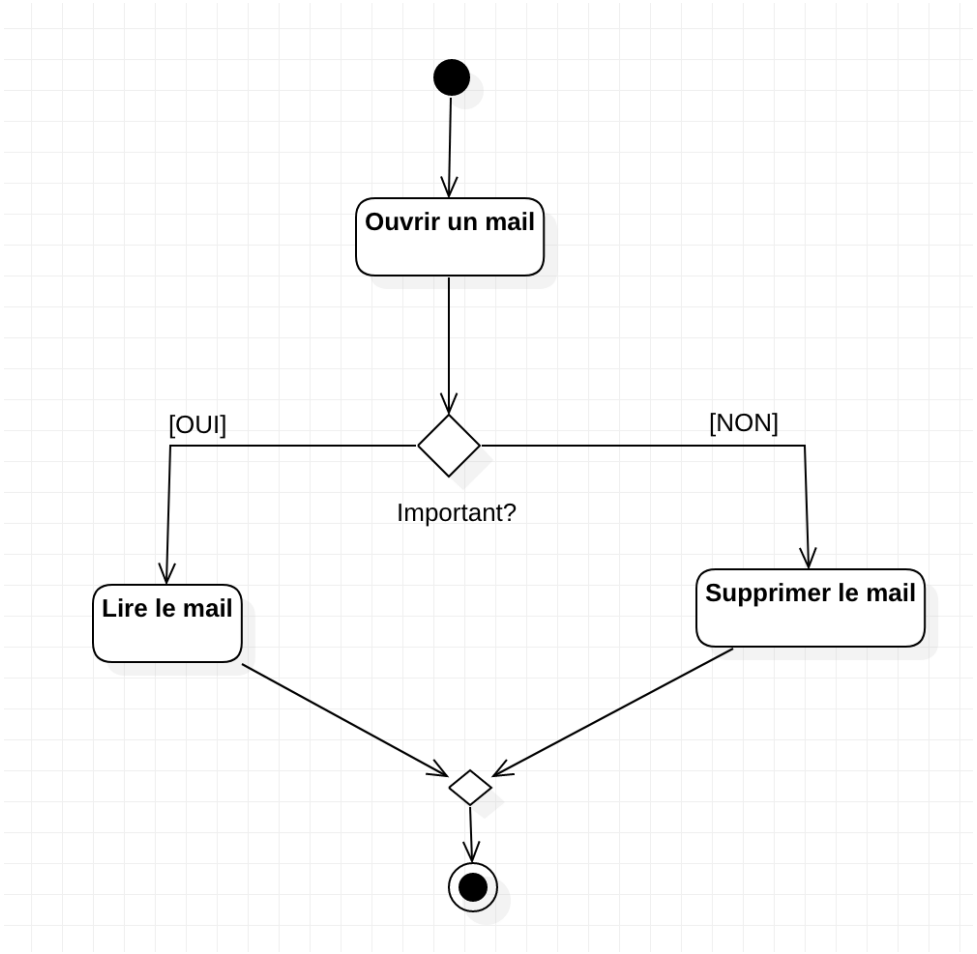
(2) Diagramme d'activité (Activity diagram)

- les éléments d'un diagramme d'activités
  - **état initial**: étape de départ avant qu'une activité ait lieu
  - **état final**: état atteint par le système à la fin d'un processus
  - **activité**
  - **décision**

Symboles



Exemple: Mail



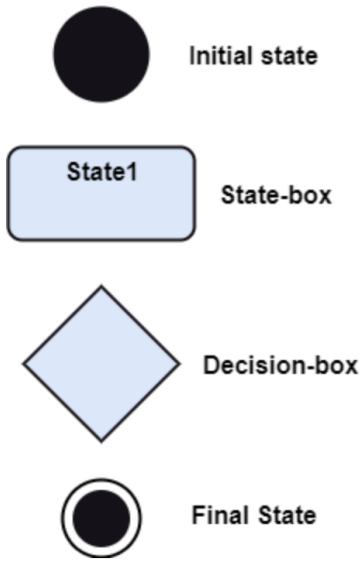
(3) Diagramme d'états-transitions (State machine diagram)

- utilisé pour modéliser un comportement discret via des transitions d'états finis
- exemple
  - états étudiant: appliquant $\rightarrow$ admis $\rightarrow$ junior $\rightarrow$ senior $\rightarrow$ licencié

(3) Diagramme d'états-transitions (State machine diagram)

- montre l'ordre des états subis par un objet dans le système
- capture le comportement du système logiciel
- modélise le comportement d'une classe, d'un sous-système ou d'un package

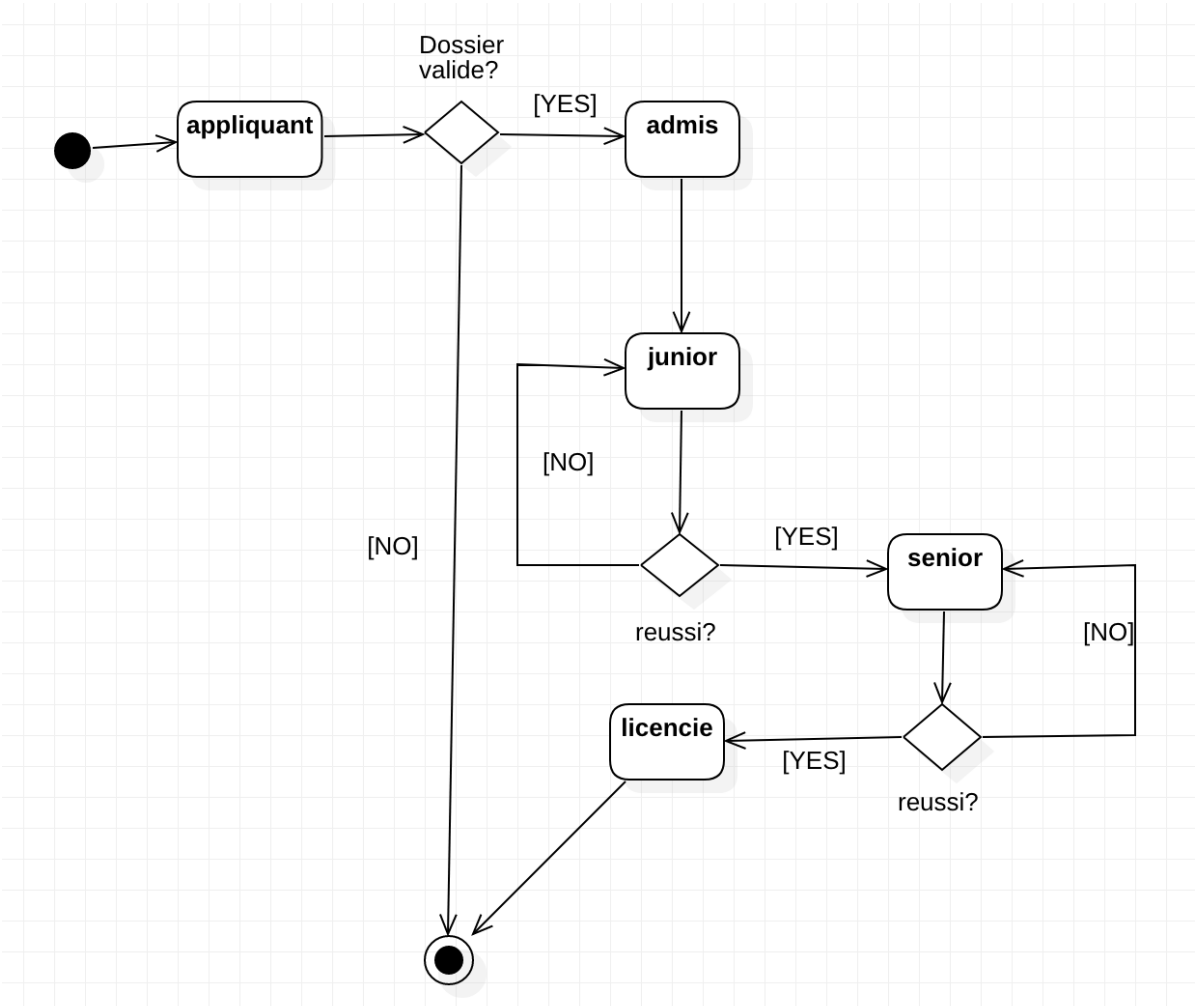
Éléments



(3) Diagramme d'états-transition

- les éléments d'un diagramme d'états-transition
  - **état initial**: début d'un système
  - **état final**: fin d'un système
  - **décision**: décisions à prendre
  - **transition**: changement de contrôle d'un état à un autre
  - **zone d'état**: elle décrit les conditions d'un objet particulier d'une classe à un moment précis

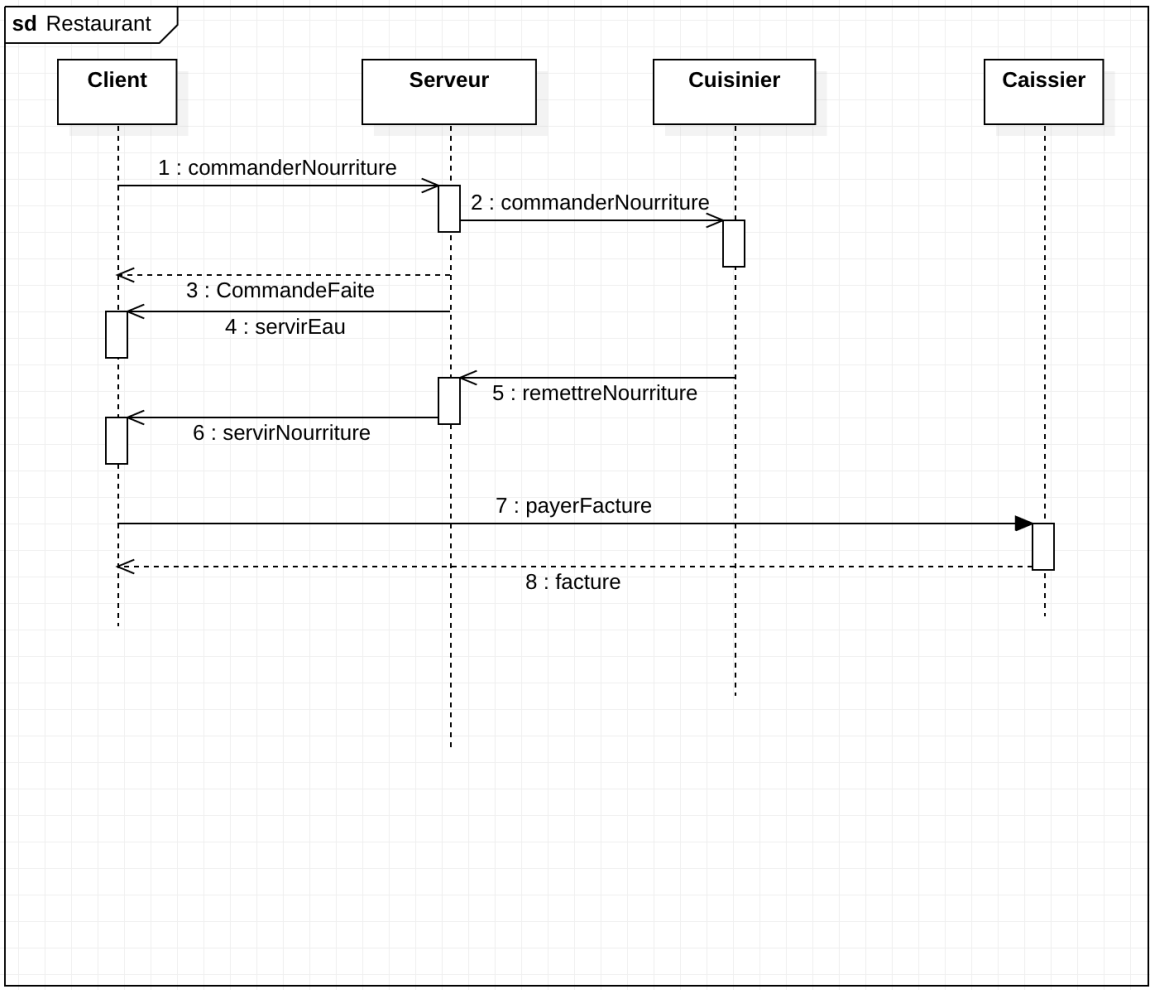
Exemple: états d'un étudiant



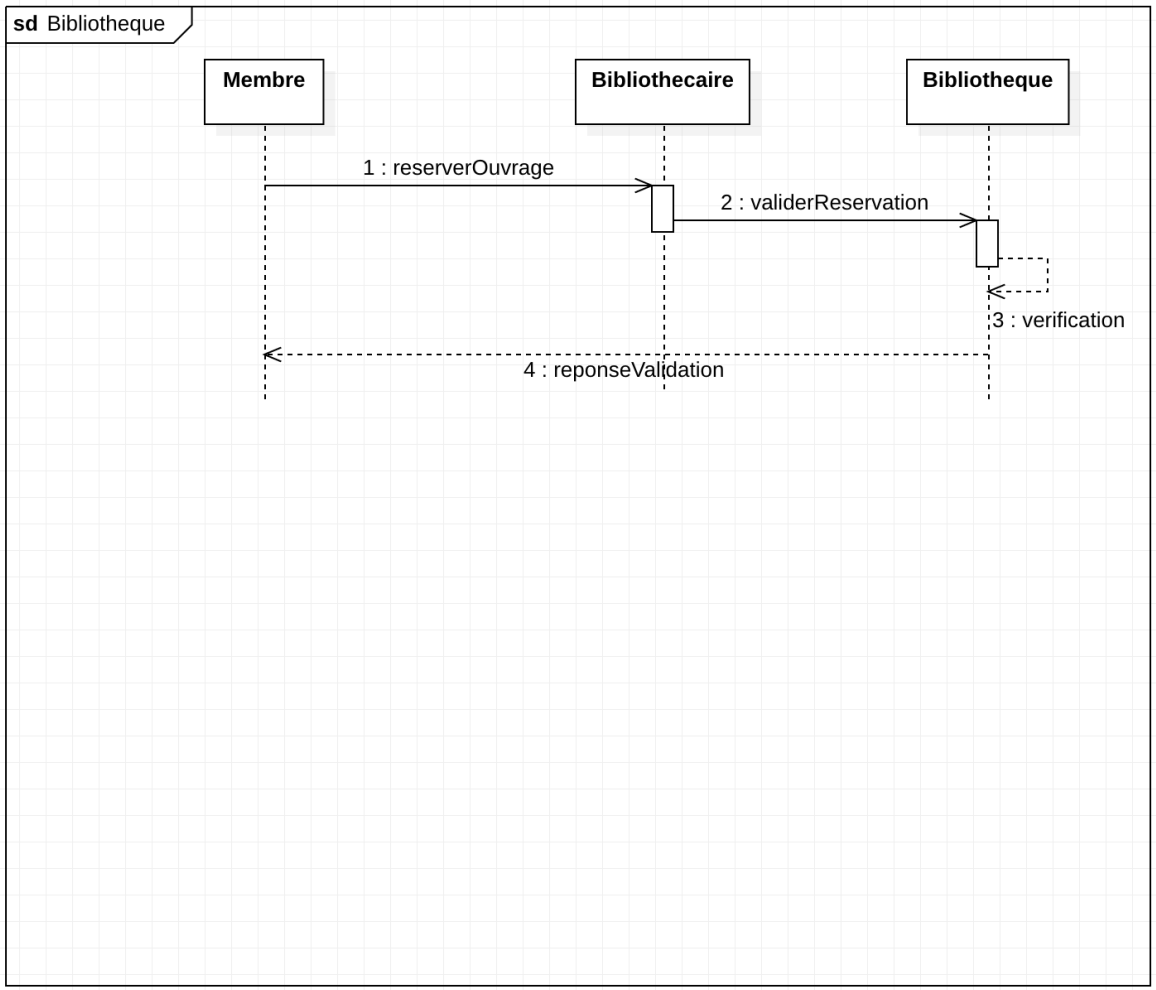
(4) Diagramme de séquence (Sequence diagram)

- se concentre sur l'échange de messages entre les éléments du système ou de ses acteurs
- exemple: restaurant
  - Jordan commande la nourriture au serveur
  - le serveur à son tour commande la nourriture au cuisiner
  - le serveur sert de l'eau à Jordan
  - le cuisinier remet la nourriture au serveur
  - le serveur sert la nourriture à Jordan
  - Jordan paie au caissier

Exemple


sd Restaurant
Client
Serveur
Cuisinier
Caissier
1 : commanderNourriture
2 : commanderNourriture
3 : CommandeFaite
4 : servirEau
5 : remettreNourriture
6 : servirNourriture
7 : payerFacture
8 : facture

Exemple


sd Bibliotheque
Membre
Bibliothecaire
Bibliotheque
1 : reserverOuvrage
2 : validerReservation
3 : verification
4 : reponseValidation

(5) Diagramme de communication (Communication diagram)

- se concentre sur les échanges de messages entre les objets
- représentation simplifiée d'un diagramme de séquence
- exemple: impression d'un document
  - Jordan se connecte à un ordinateur comme utilisateur
  - il choisit le document à imprimer
  - l'ordinateur envoie le document choisi à l'imprimante
- **TD**: Diagramme de communication pour le cas de bibliotheque
- **deadline: 23 Février 2021, 8h30**

(6) Diagramme global d'interaction (Interaction overview diagram)

- définit les interactions via une variante de diagrammes d'activités d'une manière qui favorise la vue d'ensemble du flux de contrôle
- **TD**: Diagramme global d'interaction pour le cas de bibliotheque
- **deadline: 23 Février 2021, 8h30**

(7) Diagramme de temps

- définit les interactions lorsqu'un objectif principal du diagramme est fonction du temps
- montre les changements d'état d'un objet quand ceux-ci dépendent exclusivement du temps
- indique la durée minimale ou maximale de chaque état à l'aide de contraintes temporelles

(7) Diagramme de temps

- représente une visualisation des états d'un cycle de vie par unité de temps
- met en évidence le moment où un message a été envoyé entre les objets
- explique en détail le traitement temporel d'un objet
- est utilisé avec des systèmes distribués et embarqués
- explique également comment un objet subit des changements de forme tout au long de son cycle de vie

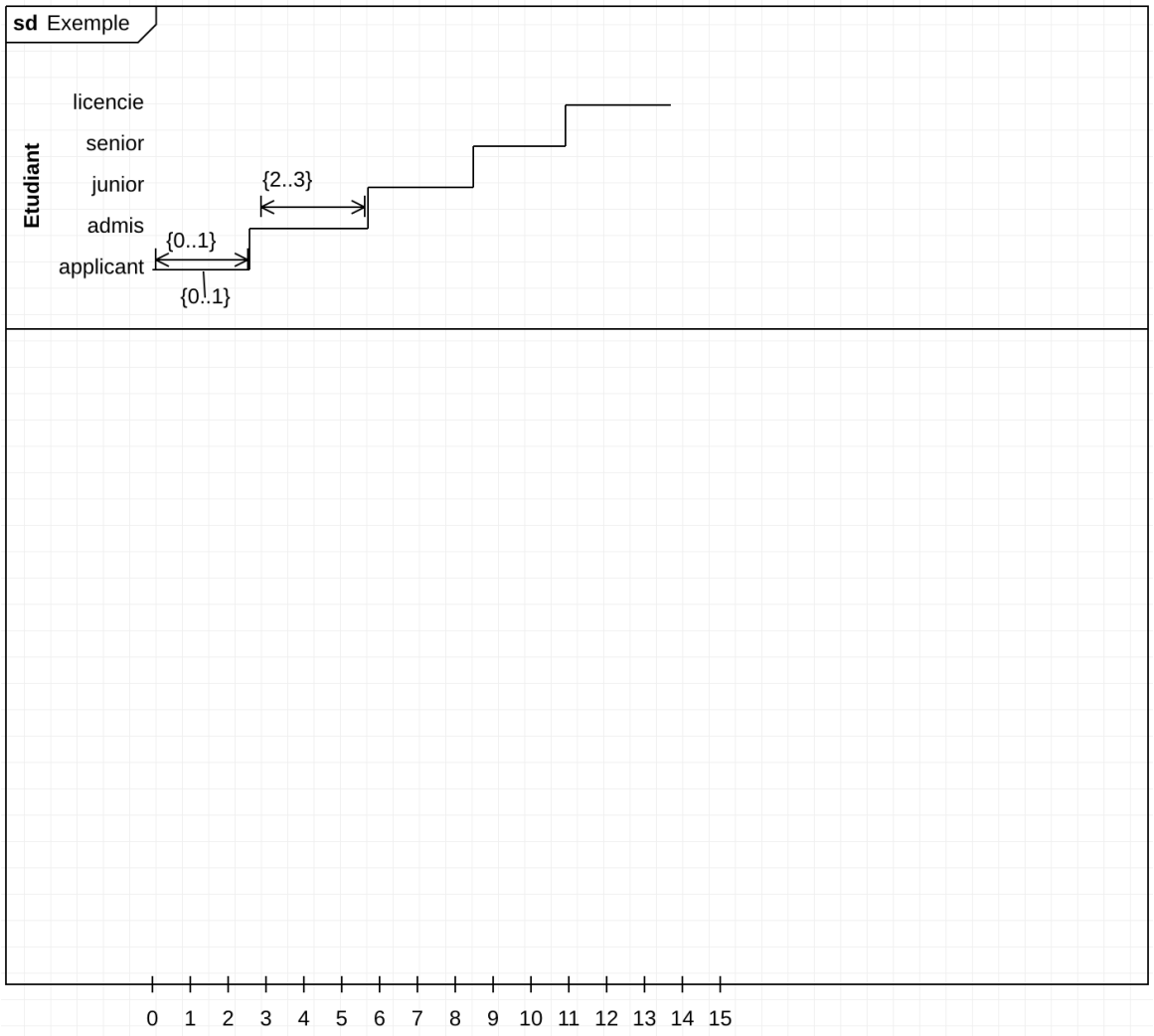
(7) Diagramme de temps

- le cycle de vie est nommé sur le côté gauche du bord
- le chronogramme est lu de gauche à droite

Elements

- Lifeline
- State/Condition
- Time Segment
- Time Tick
- Time constraint
- Duration
- Message

Exemple


sd Exemple
Etudiant
licencie
senior
junior
admis
applicant
{2..3}
{0..1}
{0..1}
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Logiciels

- installation
  - MySQL Serveur
  - MySQLWorkBench
- téléchargement
  - MySQL Connector Jar

TP en groupe

- Sujets
  - Groupe 1: gestion d'une pharmacie
  - Groupe 2: gestion d'un cyber-café
  - Groupe 3: gestion de paiement des frais académiques
  - Groupe 4: gestion du personnel d'une entreprise
  - Groupe 5: gestion d'un distributeur de monnaie (ATM)
  - Groupe 6: gestion de délibération des étudiants
  - Groupe 7: gestion d'un chariot automobile

TP en groupe

- Questions
  - détaillez la problématique de votre cas
  - énumérez les besoins fonctionnels et non-fonctionnels
  - proposez 7 extraits de diagrammes de structure (UML 2.5) pour votre cas d'utilisation
  - proposez 7 extraits de diagrammes de comportement pour votre cas d'utilisation
- **deadline: 3 Mars 2021, 18h30**

TP en groupe

- Groupe 1: Benjamin Oleko, Aggee Mbuya, Nseya et Ruth
- Groupe 2: Jacques Makabi, Martin, Ismael et Jessica Yuma
- Groupe 3: Tsheleka, Jonathan Twite, Josue Kaleta et Daniel
- Groupe 4: Shamirani Kibonge, Mwauka Lois, Ngoy Mwanza mateo et Nafisa Zabibu Therese
- Groupe 5: Ntumba Kalala Omer, Nzau Matondo Diex et Munyongenu wa Tambo Bonheur
- Groupe 6: Okytembo Paul Lambert, Kalasa Mwenda Axel, Lunde Badiambile Aaron et Kahungu Pangele Alfred
- Groupe 7: Muzinga Kilonda kethia, Ngoie Sakombi Eben-ezer, Mupandila Wadimuka Daniel et Kabamba A Mwamb Ismael

Modèles de conception (Design pattern)

- représentent les meilleures pratiques utilisées par les développeurs de logiciels orientés objet
- sont des solutions aux problèmes généraux auxquels les développeurs de logiciels ont été confrontés lors du développement de logiciels
- ces solutions ont été obtenues par essais et erreurs par de nombreux développeurs de logiciels sur une longue période
- l'apprentissage de ces modèles aide les développeurs inexpérimentés à apprendre la conception de logiciels de manière simple et rapide.

Modèles de conception (Design pattern)

- ont deux utilisations principales dans le développement de logiciels
  - **plateforme commune pour les développeurs**
    - fournissent une terminologie standard et sont spécifiques à un scénario particulier
    - par exemple, un modèle de conception singleton signifie l'utilisation d'un seul objet
  - **meilleures pratiques**
    - ont évolué sur une longue période et ils fournissent les meilleures solutions à certains problèmes rencontrés lors du développement de logiciels

Documentation

- Gamma, Erich. Design patterns: elements of reusable object-oriented software. Pearson Education India, 1995.

Types de modèles de conception

- il existe 23 modèles de conception qui peuvent être classés en trois catégories
  - modèles créatifs
  - modèles structurels
  - modèles comportementaux.
- mais il existe une quatrième catégorie de modèle de conception: modèles J2EE

Modèles créatifs

- fournissent un moyen de créer des objets tout en masquant la logique de création, en lieu et place de créer des objets directement à l'aide d'un nouvel opérateur
  - ceci donne au programme plus de flexibilité pour décider quels objets doivent être créés pour un cas d'utilisation donné
- **types**: abstract factory, builder, factory method, object pool, prototype et singleton

Modèles structurels

- concernent la composition des classes et des objets
- le concept d'héritage permet de composer des interfaces et de définir des nouvelles manières de composition des objets pour obtenir de nouvelles fonctionnalités
- **types**: adapter, bridge, composite, decorator, facade, flyweight, private class data et proxy

Modèles comportementaux

- concernent spécifiquement la communication entre les objets
- **types**: chain of responsibility, command, interpreter, iterator, mediator, memento, null object, observer, state, strategy, template method et visitor

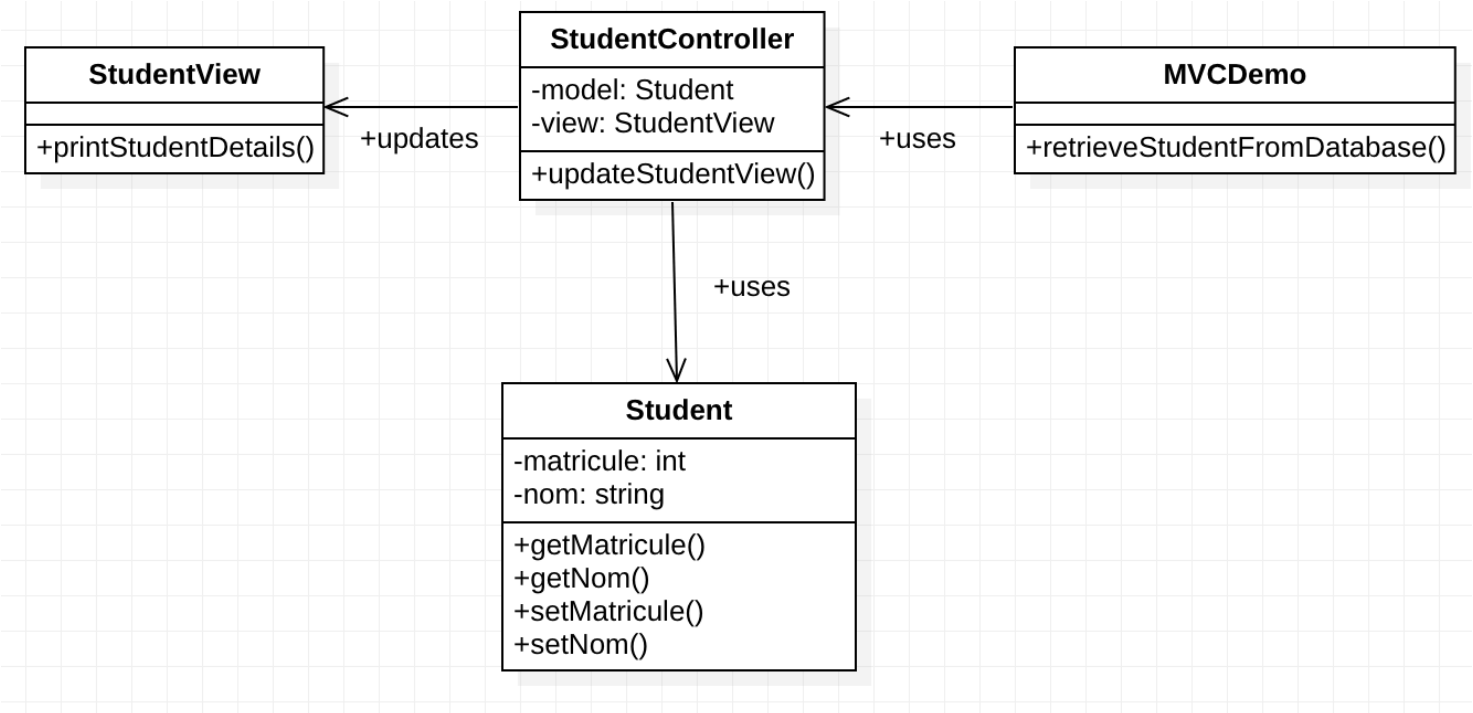
Modèles J2EE

- concernent spécifiquement le niveau de présentation
- identifiés par Sun Java Center

Model-View-Controller Pattern (MVC)

- est utilisé pour séparer les préoccupations de l'application
- est souvent utilisé pour des applications web
- il est composé de
  - **model**
    - représente un objet ou Java POJO transportant des données
    - Il peut également avoir une logique pour mettre à jour le contrôleur si ses données changent
  - **view**
    - représente la visualisation des données contenues dans le modèle
  - **controller**
    - agit à la fois sur le modèle et sur la vue
    - Il contrôle le flux de données dans l'objet modèle et met à jour la vue chaque fois que les données changent
    - Il maintient la vue et le modèle séparés

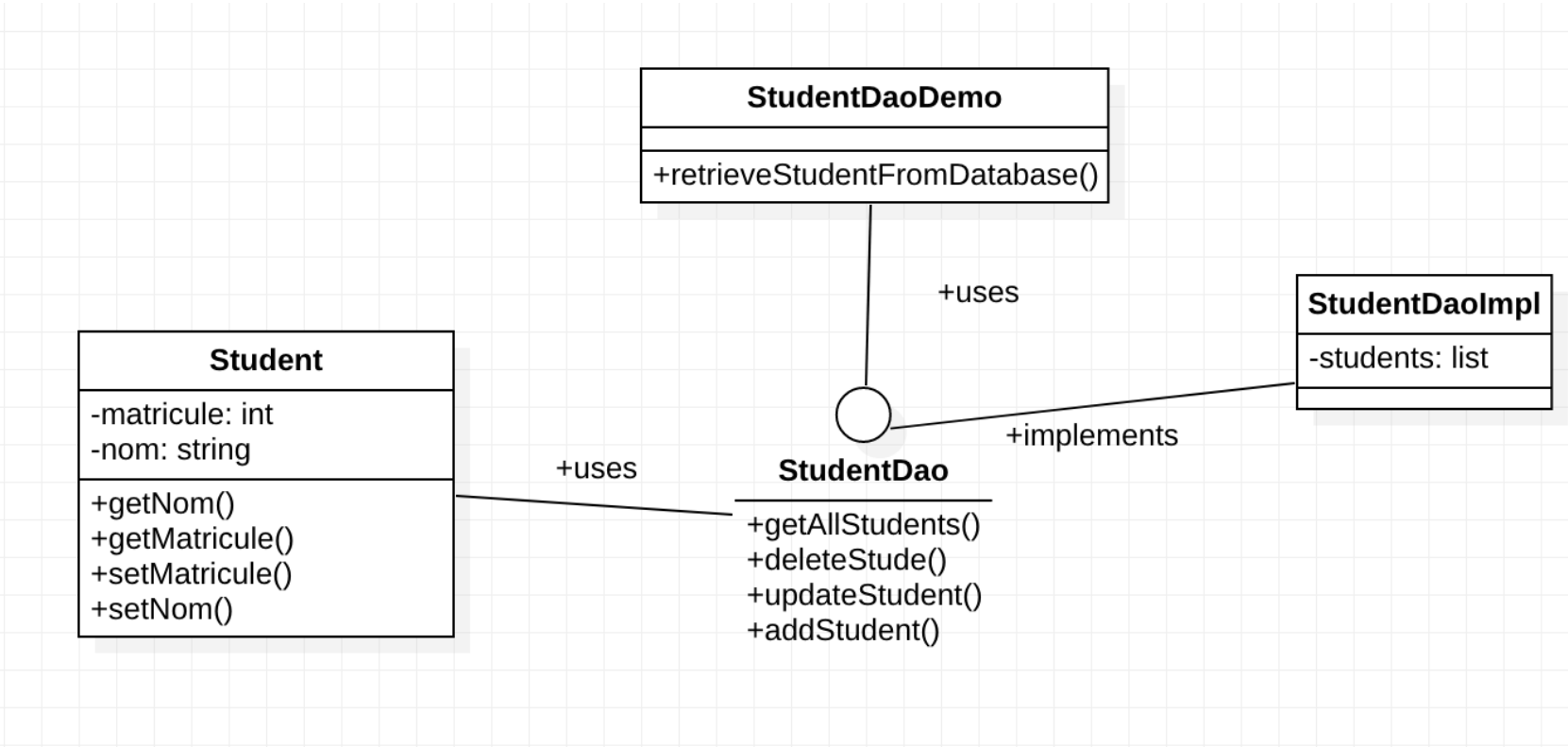
Exemple: MVC


StudentView
+printStudentDetails()
+updates
StudentController
-model: Student
-view: StudentView
+updateStudentView()
+uses
MVCDemo
+retrieveStudentFromDatabase()
+uses
Student
-matricule: int
-nom: string
+getMatricule()
+getNom()
+setMatricule()
+setNom()

Data access object pattern (DAO)

- DAO est utilisé pour séparer les données de bas niveau accédant à l'API ou aux opérations des services métier de haut niveau
- il est composé de
  - **DAO interface**
    - cette interface définit les opérations standard à effectuer sur un ou plusieurs objets de modèle
  - **DAO concrete class**
    - cette classe implémente l'interface ci-dessus
    - elle est chargée d'obtenir des données à partir d'une source de données
  - **model object**
    - cet objet est un simple POJO contenant des méthodes get / set pour stocker les données récupérées à l'aide de la classe DAO
    - POJO: a plain old Java object

Exemple: DAO


StudentDaoDemo
+retrieveStudentFromDatabase()
+uses
StudentDaoImpl
-students: list
Student
-matricule: int
-nom: string
+getNom()
+getMatricule()
+setMatricule()
+setNom()
+uses
StudentDao
+getAllStudents()
+deleteStude()
+updateStudent()
+addStudent()
+implements

Cas d'utilisation: Gestion d'une bibliothèque

Problématique

- la méthode de gestion de la bibliothèque par l'outil Microsoft Excel est moins efficace
- Microsoft Excel n'est pas un système de gestion de base de données, donc inapproprié pour une gestion efficace d'une bibliothèque
- par exemple
  - Microsoft Excel ne permet pas un système de rappel des dates de retour d'emprunts déjà à terme
  - inefficacité dans la collection des ouvrages perdus
  - manque de confidentialité des données
  - manque de pérennité
  - etc.

Cas d'utilisation: Gestion d'une bibliothèque

Les besoins fonctionnels

- Le système doit permettre au bibliothécaire de gérer :
  - les ouvrages (e.g. consulter, supprimer ou modifier, ou vérifier la disponibilité des exemplaires)
  - des membres (e.g. ajouter, supprimer ou modifier);
  - des emprunts (e.g. consulter, supprimer, modifier ou rappeler, ou définir la durée des emprunts, définir le max ouvrage à emprunter, consulter les ouvrages les plus sollicités, l'ouvrage le moins emprunté ou les nouveautés)
  - des pénalités (e.g. ajouter ou supprimer une pénalité, consulter la liste des pénalités ou définir la durée de la pénalité)
- Le système doit permettre au membre :
  - d'effectuer une recherche ou de réserver un ouvrage

Cas d'utilisation: Gestion d'une bibliothèque

Les besoins non-fonctionnels...

- Extensibilité
- Fonctionnel en réseau
- Portabilité
- Sécurité

Content layout

- Paradigmes de la programmation orientée objet
- Unified Modelling Language (UML)
- Cycle de vie du développement logiciel
- Industrie logicielle

Cycle de vie du développement logiciel

- le cycle de vie du développement logiciel est un processus utilisé par l'industrie du logiciel pour concevoir, développer et tester des logiciels de haute qualité
- il vise à produire un logiciel de haute qualité qui répond aux attentes des clients et développé dans le délais

Étapes de construction d'un logiciel

(1) **analyse**: élaboration des spécifications de l'architecture du logiciel
(2) **conception**: définition précise de chaque partie du logiciel
(3) **développement**: implémentation ou programmation
(4) **test**: vérifie le bon fonctionnement d'une partie du logiciel
(5) **validation**: démontre (y compris la documentation) qu' une activité conduit effectivement aux résultats escomptés
(6) **déploiement**: mise en production

Modèles de construction

- Il existe divers modèles de cycle de vie de développement logiciel conçus qui sont suivis pendant le processus de développement logiciel
- ces modèles sont également appelés modèles de processus de développement logiciel
- chaque modèle de processus suit une série d'étapes uniques à son type pour assurer le succès du processus de développement logiciel

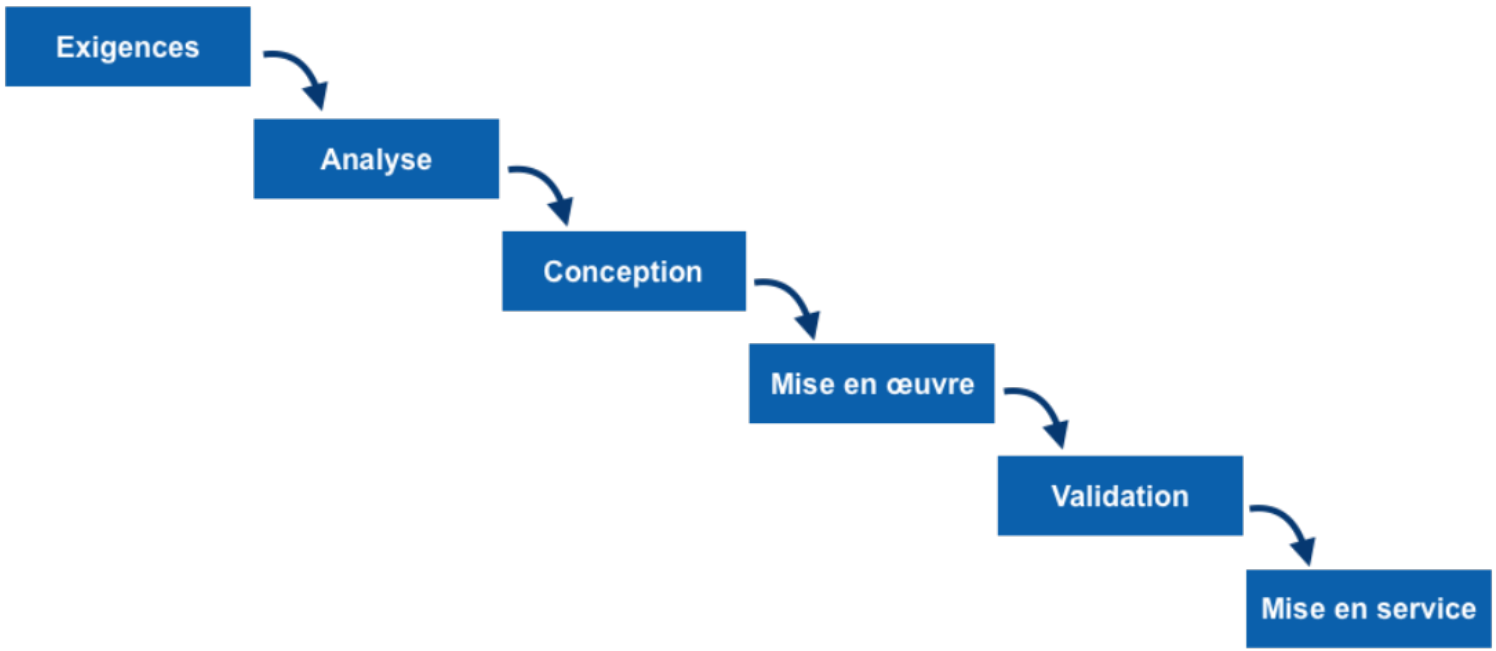
Documentation

- https://www.tutorialspoint.com/sdlc/index.htm
- Ruparelia, Nayan B. "Software development lifecycle models." ACM SIGSOFT Software Engineering Notes 35.3 (2010): 8-13.
- Leau, Yu Beng, et al. "Software development life cycle AGILE vs traditional approaches." International Conference on Information and Network Technology. Vol. 37. No. 1. 2012.

Modèle en cascade

- toutes les phases sont en cascade les unes aux autres
- la phase suivante ne commence qu'à la fin de la phase précédente
  - les phases ne se chevauchent pas

Modèle en cascade



Modèle en cascade: applications

- les exigences sont très bien documentées, claires et fixes
- la définition du produit est stable
- la technologie est comprise et n'est pas dynamique
- il n'y a pas d'exigences ambiguës
- de nombreuses ressources avec l'expertise requise sont disponibles pour prendre en charge le produit
- le projet est court

Modèle en cascade: avantages

- simple et facile à comprendre et à utiliser
- facile à gérer grâce à la rigidité du modèle
  - chaque phase a des livrables spécifiques et un processus d'examen
- des étapes clairement définies
- facile à organiser les tâches
- le processus et les résultats sont bien documentés
- etc.

Modèle en cascade: inconvénients

- aucun logiciel fonctionnel n'est produit jusqu'à la fin du processus
- risque et incertitude élevés
- pas un bon modèle pour les projets complexes et orientés objet
- ne peut pas s'adapter aux exigences changeantes
- etc.

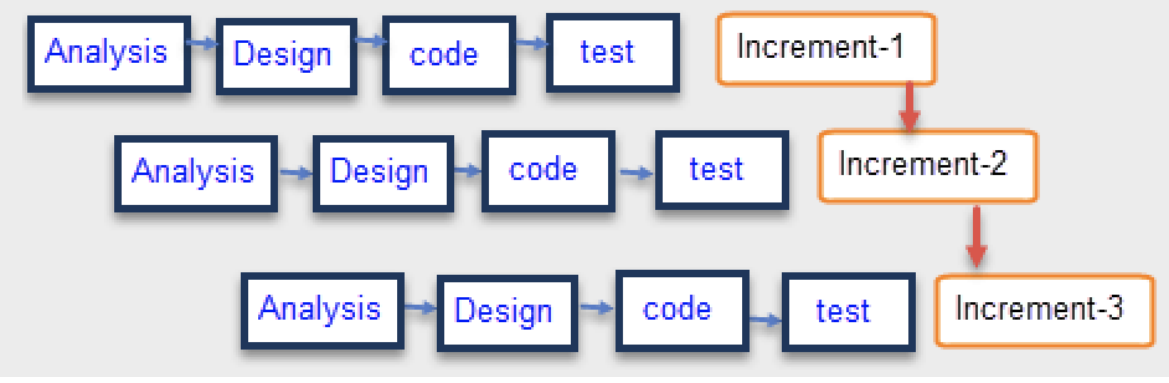
Modèle incrémentiel

- est un processus où les exigences sont décomposées en plusieurs modules autonomes du cycle de développement logiciel
- se fait par étapes depuis de l'analyse au déploiement
  - chaque version ultérieure (incrément) du système ajoute une fonction à la version précédente jusqu'à ce que toutes les fonctionnalités conçues aient été implémentées

Modèle incrémentiel

- le système est mis en production lors de la livraison du premier incrément
- le premier incrément est souvent un produit de base où les exigences de base sont satisfaites
- des fonctionnalités supplémentaires sont ajoutées dans les incréments suivants
- une fois que le produit principal est analysé par le client, il y a le développement d'un plan pour l'incrément suivant

Modèle incrémentiel



Modèle incrémentiel: applications

- les exigences du système sont clairement comprises
- une demande pour une version anticipée d'un produit se fait sentir
- l'équipe d'ingénierie logicielle n'est pas très qualifiée ou formée
- des caractéristiques et des objectifs à haut risque sont impliqués
- est plus utilisé pour les applications web et les entreprises commerciales

Modèle incrémentiel: avantages

- le logiciel sera rapidement généré
- flexible et moins coûteux de modifier les exigences et la portée
- tout au long des étapes de développement, des changements peuvent être effectués
- est moins coûteux
- un client peut répondre à chaque incrément
- les erreurs sont faciles à identifier
- etc.

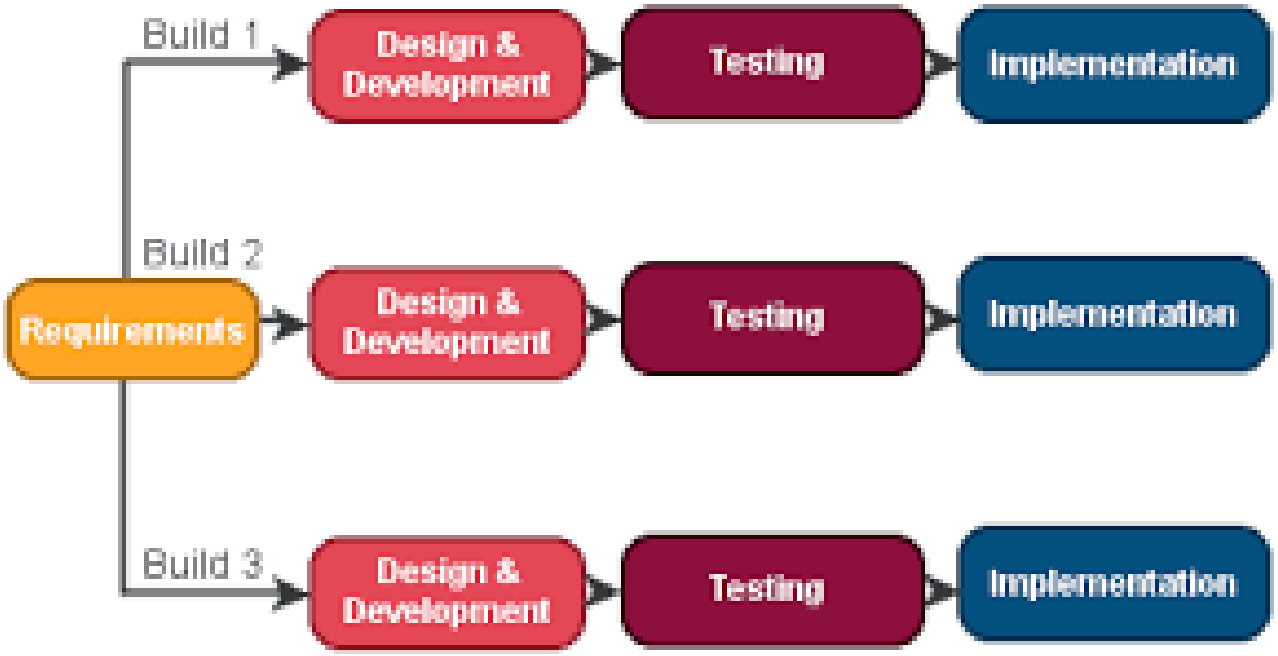
Modèle incrémentiel: inconvénients

- une bonne planification de la conception
- chaque phase d'itération est rigide et ne se chevauche pas
- la résolution d'un problème dans une unité nécessite une correction dans toutes les unités et prend beaucoup de temps
- etc.

Modèle itératif

- commence par une mise en œuvre simple d'un sous-ensemble des exigences logicielles
- améliore de manière itérative des versions jusqu'à ce que le système complet soit mis en œuvre
- à chaque itération, des modifications de conception sont apportées et de nouvelles capacités fonctionnelles sont ajoutées
  - l'idée de base derrière ce modèle est de développer un système à travers des cycles répétés (itératifs) et en plus petites portions à la fois (incrémentiel)

Modèle itératif



Modèle itératif: applications

- les exigences du système sont clairement définies et comprises
- les exigences majeures doivent être définies
  - certaines fonctionnalités ou améliorations demandées peuvent évoluer avec le temps
- il y a un deadline à la contrainte du marché
- une nouvelle technologie est utilisée et apprise par l'équipe de développement pendant le développement du projet
- les ressources avec les compétences nécessaires ne sont pas disponibles et sont prévues pour être utilisées sur une base contractuelle
- certains caractéristiques et objectifs à haut risque peuvent changer à l'avenir

Modèle itératif: avantages

- certaines fonctionnalités de travail peuvent être développées rapidement et tôt
- un développement parallèle peut être planifié
- les progrès peuvent être mesurés
- moins coûteux pour modifier la portée / les exigences
- les tests au cours d'une itération sont faciles
- les risques sont identifiés et résolus lors de l'itération
- à chaque incrément, le produit opérationnel est livré
- l'analyse des risques est meilleure
- il prend en charge les exigences changeantes
- mieux adapté aux projets de grande envergure

Modèle itératif: inconvénients

- plus de ressources peuvent être nécessaires
- il n'est pas très adapté à l'évolution des besoins
- une plus grande attention de la direction est requise
- des problèmes d'architecture ou de conception du système peuvent survenir car toutes les exigences ne sont pas rassemblées au début du processus
- ne convient pas aux petits projets
- la fin du projet peut ne pas être connue, ce qui constitue un risque
- des ressources hautement qualifiées sont nécessaires pour l'analyse des risques
- l'avancement des projets dépend fortement de la phase d'analyse des risques

Modèle en spirale

- est une combinaison de modèle itératif et de modèle de modèle en cascade avec un très fort accent sur l'analyse des risques
- il permet des versions incrémentielles du produit ou un raffinement incrémentiel à chaque itération
- l'analyse des risques comprend l'identification, l'estimation et le suivi de la faisabilité technique et des risques de gestion
  - tels que le retard de calendrier et le dépassement des coûts
- après avoir testé la version, à la fin de la première itération, le client évalue le logiciel et fournit des commentaires
- en cas de satisfaction du client, le développement entre dans l'itération suivante et suit ensuite l'approche en cascade

Modèle en spirale: applications

- lorsqu'il y a une contrainte budgétaire et qu'une évaluation des risques est importante
- des projets à risque moyen et élevé
- engagement de projet à long terme en raison des changements potentiels des priorités économiques
- le client n'est pas sûr de ses exigences
- les exigences sont complexes et nécessitent une évaluation pour être clarifiées
- des changements importants sont attendus au cours du développement

Modèle en spirale: avantages

- des exigences changeantes peuvent être adaptées
- permet une utilisation intensive des prototypes
- les utilisateurs voient le système tôt
- le développement peut être divisé en parties plus petites et les parties à risque peuvent être développées plus tôt
  - ce qui contribue à une meilleure gestion des risques

Modèle en spirale: inconvénients

- la gestion est plus complexe
- la fin du projet peut ne pas être connue à l'avance
- ne convient pas aux projets à faible risque
- pourrait être coûteux pour les petits projets
- le processus est complexe
- un grand nombre d'étapes intermédiaires nécessite une documentation excessive

Modèle en V

- est une extension du modèle en cascade
- repose sur l'association d'une phase de test pour chaque étape de développement
  - pour chaque phase du cycle de développement, il y a une phase de test directement associée
- un modèle hautement discipliné et la phase suivante ne commence qu'après l'achèvement de la phase précédente

Modèle en V: applications

- les exigences sont bien définies, clairement documentées et fixées
- la définition du produit est stable
- la technologie n'est pas dynamique et est bien comprise par l'équipe du projet
- pas d'exigences ambiguës ou indéfinies.
- le projet est court

Modèle en V: avantages

- un modèle hautement discipliné
  - les phases sont exécutées une par une
- simple et facile à comprendre et à utiliser
- facile à gérer grâce à la rigidité du modèle
  - chaque phase a des livrables spécifiques et un processus d'examen

Modèle en V: inconvénients

- risque élevé et incertitude
- pas un bon modèle pour les projets complexes et orientés objet
- ne convient pas aux projets où les exigences présentent un risque modéré ou élevé de changement
- une fois qu'une application est en phase de test, il est difficile de revenir en arrière et de modifier une fonctionnalité
- aucun logiciel fonctionnel n'est produit jusqu'à la fin du cycle de vie

Modèle Big Bang

- pas de processus spécifique à suivre
- le développement commence juste avec les ressources disponibles
- le logiciel développé peut ou ne peut pas être conforme aux exigences du client

Modèle Big Bang: applications

- idéal pour des petits projets
- avec un ou deux développeurs
- utile pour des projets académiques
- les exigences ne sont pas bien comprises et le deadline n'est pas connu

Modèle Big Bang: avantages

- très simple
- quasiment pas de planification requise
- facile à gérer
- très peu de ressources requises
- flexibilité des développeurs
- idéal pour les novices

Modèle Big Bang: inconvénients

- risque et incertitude très élevés
- pas un bon modèle pour les projets complexes et orientés objet
- peut s'avérer très coûteux si les exigences sont mal comprises

Modèle Agile

- une combinaison de modèles itératif et incrémentiel
  - mettant l'accent sur l'adaptabilité des processus et la satisfaction du client grâce à la livraison rapide d'un produit logiciel fonctionnel
- divise le produit en petites versions incrémentielles
- ces incréments sont fournis en itérations
  - chaque itération dure généralement d'environ une à trois semaines
  - chaque itération implique travail en synergie des diverses équipes des différents domaines

Modèle Agile: principes

- individus et équipe
- demos réguliers
- collaboration avec le client
- réponse rapide au changement

Modèle Agile: recommandations

- une approche adaptative
- pas de planification détaillée
- pas de clarté sur les tâches futures
- l'équipe s'adapte de manière dynamique aux exigences changeantes du produit
- le produit est testé très fréquemment

Modèle Agile: avantages

- très réaliste et facile à gérer
- favorise le travail en équipe
- une fonctionnalité peut être développée et démontrée rapidement
- les besoins en ressources sont minimaux
- adapté aux exigences fixes ou changeantes
- fournit des solutions de travail partielles précoces
- développement et livraison simultanés
- presque pas de planification requise
- donne de la flexibilité aux développeurs

Modèle Agile: inconvénients

- ne convient pas à la gestion des dépendances complexes
- risque de durabilité, de maintenabilité et d'extensibilité élevé
- un plan global et un leader agile sont indispensables
- dépend fortement de l'interaction avec le client
  - si le client n'est pas clair, l'équipe peut être conduite dans une mauvaise direction
- le transfert de technologie aux nouveaux membres de l'équipe peut être assez difficile en raison du manque de documentation

Autres modèle

- Rapid Application Development (RAD)
- Software Prototype

Content layout

- Paradigmes de la programmation orientée objet
- Unified Modelling Language (UML)
- Cycle de vie du développement logiciel
- Industrie logicielle

Documentation

-

Industrie logicielle

- se compose de la partie de l'activité de programmation informatique qui est échangée entre les organisations productrices de logiciels et les consommateurs de logiciels d'entreprise ou individuels
- comprend des services logiciels, tels que la formation, la documentation, le conseil et la récupération de données

Industrie logicielle

- comprend des entreprises de développement, de maintenance et de publication de logiciels qui utilisent différents modèles commerciaux
- modèles commerciaux basés sur
  - les licences / maintenance (sur le site)
  - cloud: e.g. SaaS, PaaS et IaaS
    - SaaS: software as a service (e.g. Dropbox)
    - PaaS: platform as a service (e.g. Windows Azure)
    - IaaS: infrastructure as a service (e.g. Amazon Web Service)

Développeurs de logiciels

- sont les créateurs des programmes informatiques
  - certains développent des applications qui permettent aux gens d'effectuer des tâches spécifiques sur un ordinateur
  - d'autres développent les systèmes d'exploitation ou de contrôle
    - par exemple, contrôleur d'un réseau

Tâches des développeurs de logiciels

- analyser les besoins des utilisateurs, puis concevoir, tester et développer des logiciels pour répondre à ces besoins
- recommander des mises à niveau logicielles pour les programmes et systèmes existants des clients
- concevoir chaque élément d'une application ou d'un système et planifier la manière dont les éléments fonctionneront ensemble

Tâches des développeurs de logiciels

- créer une variété de modèles (e.g. UML) et de diagrammes (e.g. organigrammes) qui montrent aux programmeurs le code logiciel nécessaire pour une application
- s'assurer qu'un programme continue de fonctionner normalement grâce à la maintenance et aux tests du logiciel
- documenter chaque aspect d'une application ou d'un système comme référence pour la maintenance et les mises à niveau futures
- collaborer avec d'autres informaticiens pour créer un logiciel optimal

Besoins de développeurs logiciels

- l'emploi des développeurs de logiciels accroit bien plus rapidement que la moyenne de toutes les professions
- le besoin de nouvelles applications sur les smart phones et les tablettes contribuera à accroître la demande de développeurs de logiciels d'application

Besoins de développeurs logiciels

- l'industrie d'assurance maladie et médicale aura besoin de logiciels innovants pour gérer les nouvelles souscriptions de polices de soins de santé
- les développeurs de systèmes sont susceptibles de voir de nouvelles opportunités en raison d'une augmentation du nombre de produits utilisant des logiciels
  - par exemple, de plus en plus de systèmes informatiques sont intégrés à des produits électroniques

Besoins de développeurs logiciels

- les préoccupations concernant les menaces à la sécurité informatique pourraient se traduire par un investissement plus important dans les logiciels de sécurité
- logiciels de sécurité protégeront les réseaux informatiques et l'infrastructure électronique

Besoins de développeurs logiciels

- en outre, une augmentation des logiciels proposés sur Internet devrait réduire les coûts de développement
- réduction de coûts de développement permettra un plus grand besoin de personnalisation des entreprises
  - ce qui implique augmentation de demande de développeurs logiciels

Types d'ingénieurs logiciels

- choisir un profil d'ingénieur
- connaitre son profil d'ingénieur
- il est possible de combiner plusieurs profils à la fois?

Types d'ingénieurs logiciels

- **chercheurs informatique**: inventent et conçoivent de nouvelles approches de la technologie informatique et trouvent des utilisations innovantes de la technologie existante
  - ils étudient et résolvent des problèmes complexes en informatique dans plusieurs domaines

Types d'ingénieurs logiciels

- **responsables des systèmes informatiques et d'information**: souvent appelés responsables des technologies de l'information (IT) ou chefs de projet informatique
  - ils planifient, coordonnent et dirigent les activités informatiques dans une organisation
  - ils aident à déterminer les objectifs informatiques d'une organisation et sont responsables de la mise en œuvre des systèmes informatiques pour atteindre ces objectifs

Types d'ingénieurs logiciels

- **ingénieurs en matériel informatique** recherchent, conçoivent, développent et testent des systèmes et des composants informatiques tels que des processeurs, des cartes de circuits imprimés, des périphériques de mémoire, des réseaux et des routeurs
- **architectes de réseaux informatiques**: conçoivent et construisent des réseaux de communication de données (e.g. LAN, WAN et intranets)
  - ces réseaux vont des petites connexions entre deux bureaux aux capacités de mise en réseau de nouvelle génération, comme une infrastructure cloud qui dessert plusieurs clients

Types d'ingénieurs logiciels

- **programmeurs informatiques**: écrivent le code et testent son bon fonctionnement
  - ils transforment les conceptions de programmes créées par les développeurs de logiciels et les ingénieurs en instructions qu'un ordinateur peut suivre
- **spécialistes du support informatique**: fournissent aide et conseils aux utilisateurs d'ordinateurs et aux organisations
  - ils prennent en charge les réseaux informatiques ou fournissent une assistance technique directement aux utilisateurs d'ordinateurs

Types d'ingénieurs logiciels

- **analystes** ou **architectes de systèmes informatiques**: étudient les systèmes et procédures informatiques actuels d'une organisation et conçoivent des solutions pour aider l'organisation à fonctionner plus efficacement
  - ils réunissent les entreprises et les technologies de l'information en comprenant les besoins et les limites des deux

Types d'ingénieurs logiciels

- **administrateurs de base de données (DBA)**: utilisent des logiciels spécialisés pour stocker et organiser les données, telles que les informations financières et les enregistrements d'expédition des clients
  - ils s'assurent que les données sont disponibles pour les utilisateurs et protégées contre tout accès non autorisé
- **analystes de la sécurité de l'information**: planifient et mettent en œuvre des mesures de sécurité pour protéger les réseaux et les systèmes informatiques d'une organisation
  - leurs responsabilités ne cessent de s'étendre à mesure que le nombre de cyberattaques augmente

Types d'ingénieurs logiciels

- **développeurs web**: conçoivent et créent des sites web
  - ils sont responsables du design du site
  - ils sont également responsables des aspects techniques du site, tels que ses performances et sa capacité, qui sont des mesures de la vitesse d'un site web et du trafic que le site peut gérer
  - en outre, les développeurs Web peuvent créer du contenu pour le site
- etc.

Quelques entreprises type de l'industrie logicielle

- Microsoft
- Oracle
- IBM
- VMware
- Facebook
- etc.

THANK YOU!