Analyse et conception de systèmes orientés objet

Dame Samb

UCAD/FST/DMI

Objectifs du cours

- Comprendre les concepts clés de l'approche orientée objet.
- Savoir aborder un problème en se basant sur une approche orientée objet.
- Maîtriser les principaux diagrammes UML qui permettent de concevoir un système orienté objet.

Plan général du cours

- Introduction à la conception orientée objet.
- Diagramme de cas d'utilisation.
- Diagramme de classes et d'objets.
- Diagramme de séquence.
- Diagramme de collaboration
- Diagramme d'état.
- Diagramme des activités.

concepts

Méthodes d'analyse et de conception

- Une méthode comprend:
  - **une démarche:** explique la marche à suivre en exploitant au mieux les principes de modularité, d'abstraction, de réutilisation, etc.
  - **un formalisme de représentation**: facilite la communication, l'organisation et la vérification.
  - **des modèles**: facilitent les retours sur la conception et l'évolution des applications.

Méthodes d'analyse et de conception

- Il existe de nombreuses méthodes:
  - **Méthodes fonctionnelles**: diviser pour régner
    - SAD, SA-SD, etc.
  - **Méthodes systémiques**: séparation des données et des traitements.
    - Merise, Entité Relation, etc.
  - **Méthodes objets**: intégration des données et des traitements dans un objet unique.
    - OMT, OOSE, etc.

Paradigme orienté objet

- Objectif principal: réduire et gérer la complexité des logiciels:
  - Décomposition modulaire.
  - Regroupement des fonctions et des propriétés concernant un type donné dans un module.
  - Cacher la complexité des fonctions et celle de leurs actions.
  - Fournir une interface qui sera la partie visible du module.
  - Communication par envoi de messages.

MCOO: historique

- Avec l'importance de la POO, la nécessité d'une MACOO devient une évidence.
- Entre 1990 et 1995, apparition de plus de 50 méthodes:
  - OOA de Coad et Yourdon
  - Méthode de Shlaer et Mellor
  - OOM (Merise orienté objet)
  - OMT (Rumbaugh et al. 1991)
  - Objectory (Jaconbson et al. 1992)
  - Méthode de Booch (1994)

MCOO: historique

- En 1994, consensus autour de trois méthodes:
  - **OMT de Rumbaugh:** fournit une représentation graphique des aspects statiques, dynamiques et fonctionnels d'un système.
  - **OOD de Booch:** introduit le concept de paquetage
  - **OOSE de Jacobson:** fonde l'analyse sur la description des besoins des utilisateurs (use cases).
- Recherche d'un langage commun unique:
  - utilisable par toute méthode objet, dans toutes les phases du cycle de vie.
  - compatible avec les techniques de réalisation actuelles.

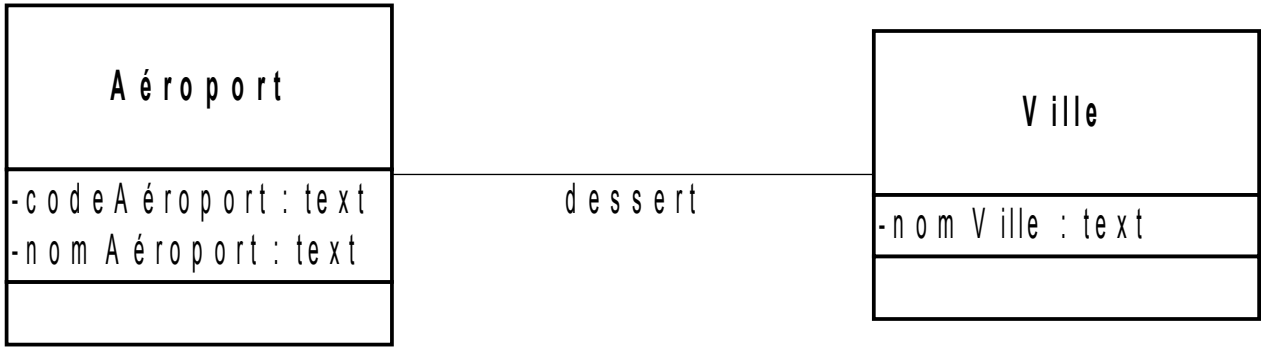
⇨ **UML**

Concepts de l'approche objet

- **Objet:** représente une entité physique, logicielle, ou conceptuelle.
  - Exemples: un client, un logiciel, une voiture.
- Un objet possède:
  - Une **identité**: permet de distinguer chaque objet par rapport aux autres.
  - un **état**: correspond aux valeurs de tous ses attributs à un instant donné.
  - un **comportement**: ensemble des opérations qu'il peut exécuter en réaction aux messages provenant des autres objets.

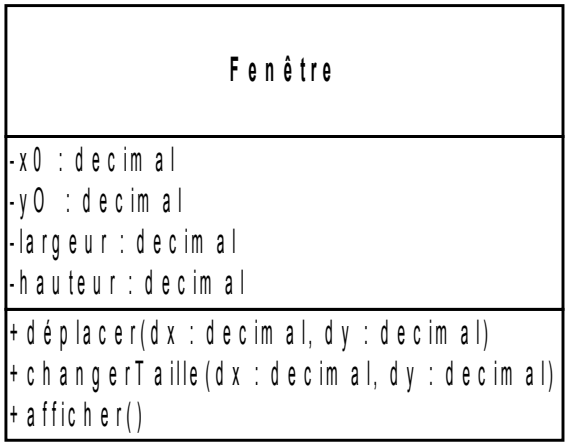
Concepts de l'approche objet

- **Classe:** description abstraite d'un ensemble d'objets possédant une structure identique (liste des attributs), un même comportement (liste des opérations), une même sémantique et les mêmes relations**.**



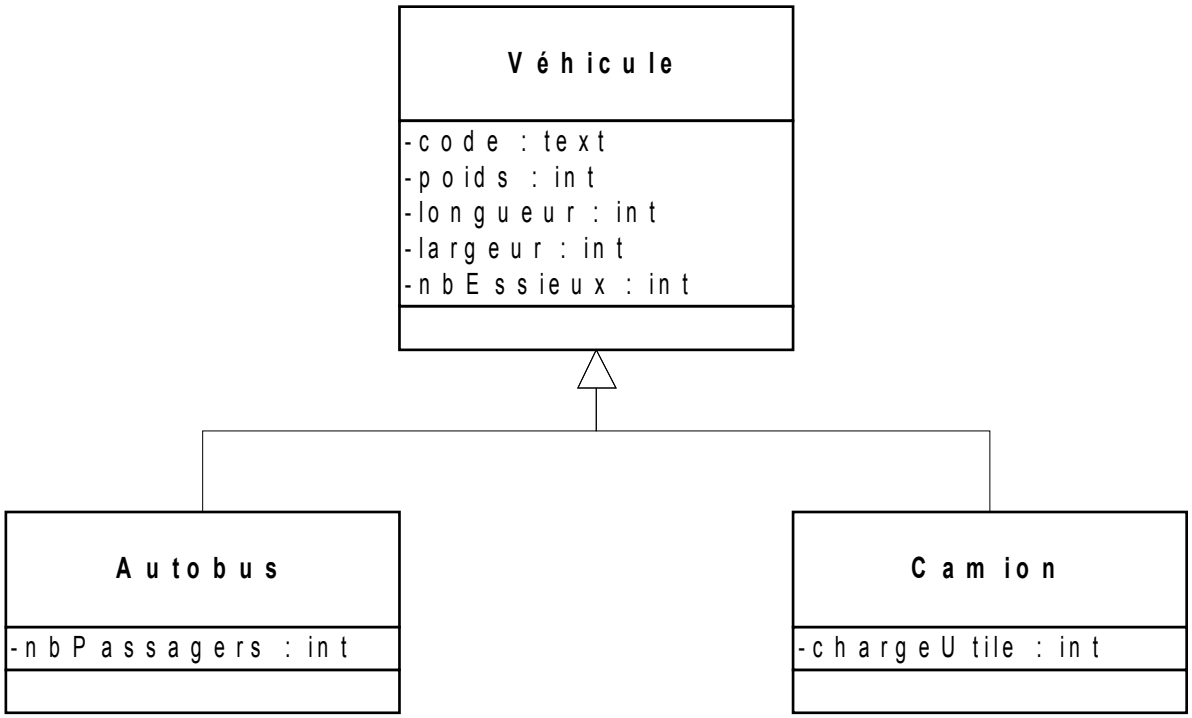
Concepts de l'approche objet

- **Encapsulation:** technique consistant à regrouper les données et les méthodes d'un objet et à masquer les détails de leur implémentation.
- **Interface:** vue externe d'un objet, définit les services accessibles aux utilisateurs de l'objet.



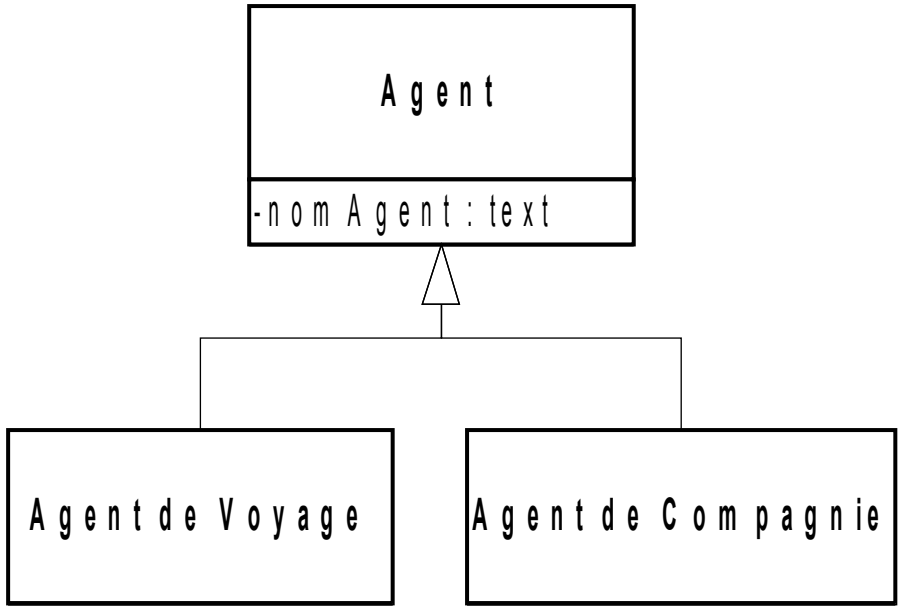
Concepts de l'approche objet

- **héritage:** mécanisme de transmission des propriétés d'une classe (attributs et méthodes) vers une sous classe évitant les duplications d'information



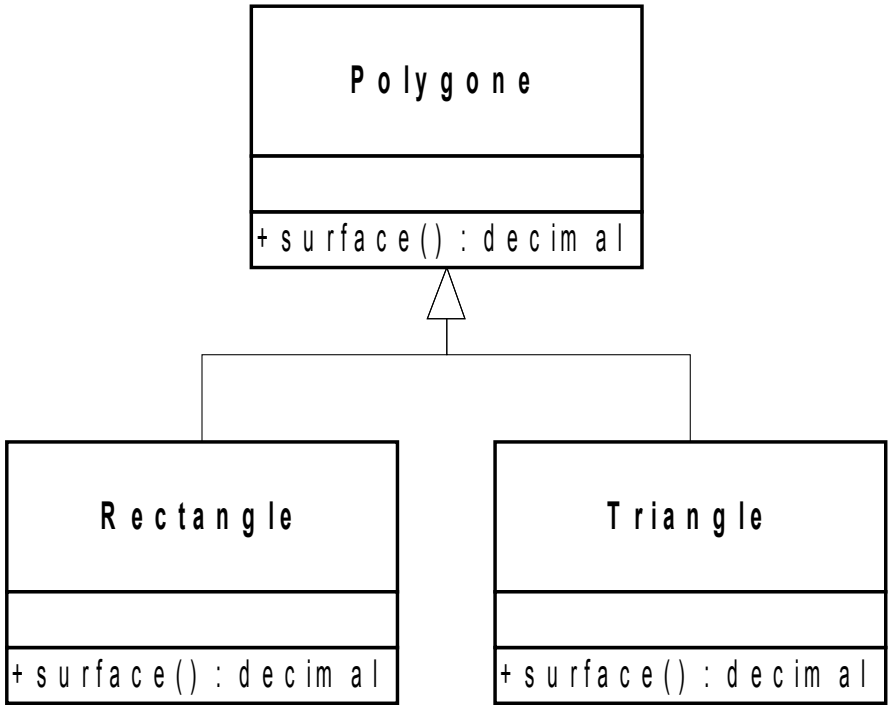
Concepts de l'approche objet

- **généralisation:** factorisation des éléments communs de classes (attributs, méthodes)
- **spécialisation:** adaptation d'une classe générale à un cas particulier.



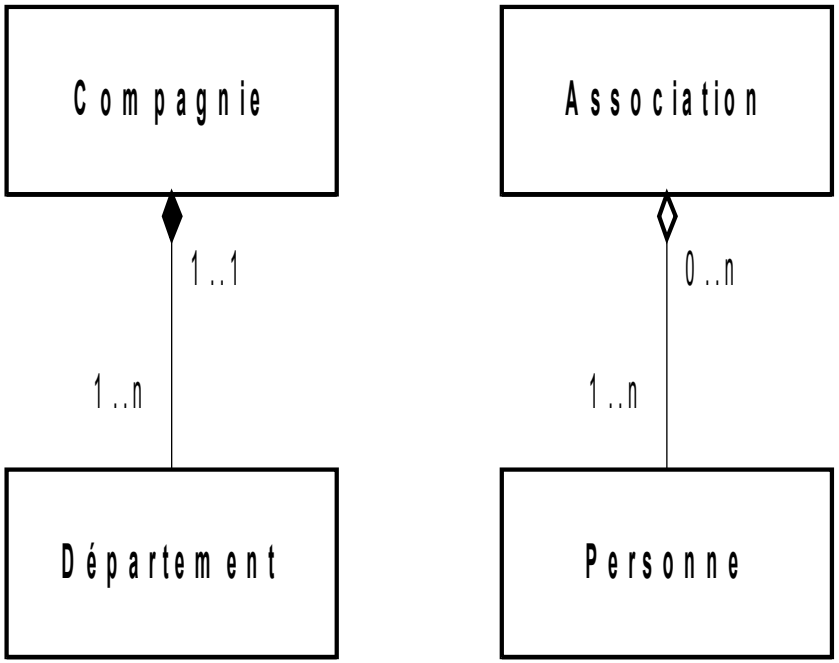
Concepts de l'approche objet

- **polymorphisme:** faculté d'une méthode de s'exécuter différemment suivant le contexte de la classe ou elle se trouve.



Concepts de l'approche objet

- **agrégation**: relation entre plusieurs classes, spécifiant qu'une classe est composée d'une ou plusieurs autres classes.



Phases de développement

Phases de développement

- Le cycle de développement classique comporte cinq étapes:
  - **Analyse des besoins:** s'accorder sur ce qui doit être fait dans le système.
  - **Analyse:** comprendre les besoins et les décrire dans le système.
  - **Conception:** s'accorder sur la manière dont le système doit être construit.
  - **Implémentation**: Codage du résultat de la conception.
  - **Test:** Le système est-il conforme au cahier des charges

Analyse des besoins

- Capturer les besoins des clients.
  - clarifier, filtrer et organiser les besoins, ne pas chercher l'exhaustivité.
- Délimiter les frontières du système.
  - Spécifier le «quoi» fait par le logiciel.
- Étudier la faisabilité du projet
  - Faisabilité organisationnelle.
  - Faisabilité technique.
  - Faisabilité temporelle
  - Faisabilité financière

Analyse

- **Analyse du domaine**: identifier les éléments du domaine ainsi que les relations et interactions entre ces éléments.
- **Analyse de l'existant:** déterminer les fonctions principales du système.
- **Analyse organisationnelle:** déterminer la structure de l'organisation.
- **Analyse technique:** recenser les équipements informatiques en place.

conception

- Définir l'architecture du logiciel
- Définir chaque constituant du logiciel
  - Informations traitées
  - Opérations effectuées
  - Résultats fournis
  - Contraintes à respecter
- Optimiser les modèles
- Choisir un langage de programmation

Implémentation

- Création des objets et des classes dans le langage de POO choisi.
- Réutilisation composants existants (objets, classes, packages).
- Intégration des composants.

Tests

- **Tests unitaires:** permettent de vérifier que chaque module fonctionne correctement indépendamment des autres.
- **Tests d'intégration:** permettent de vérifier que tous les programmes testés individuellement fonctionnent bien ensemble.
- **Tests systèmes**: permettent de vérifier que le système fonctionne correctement dans les conditions réelles d'utilisation.

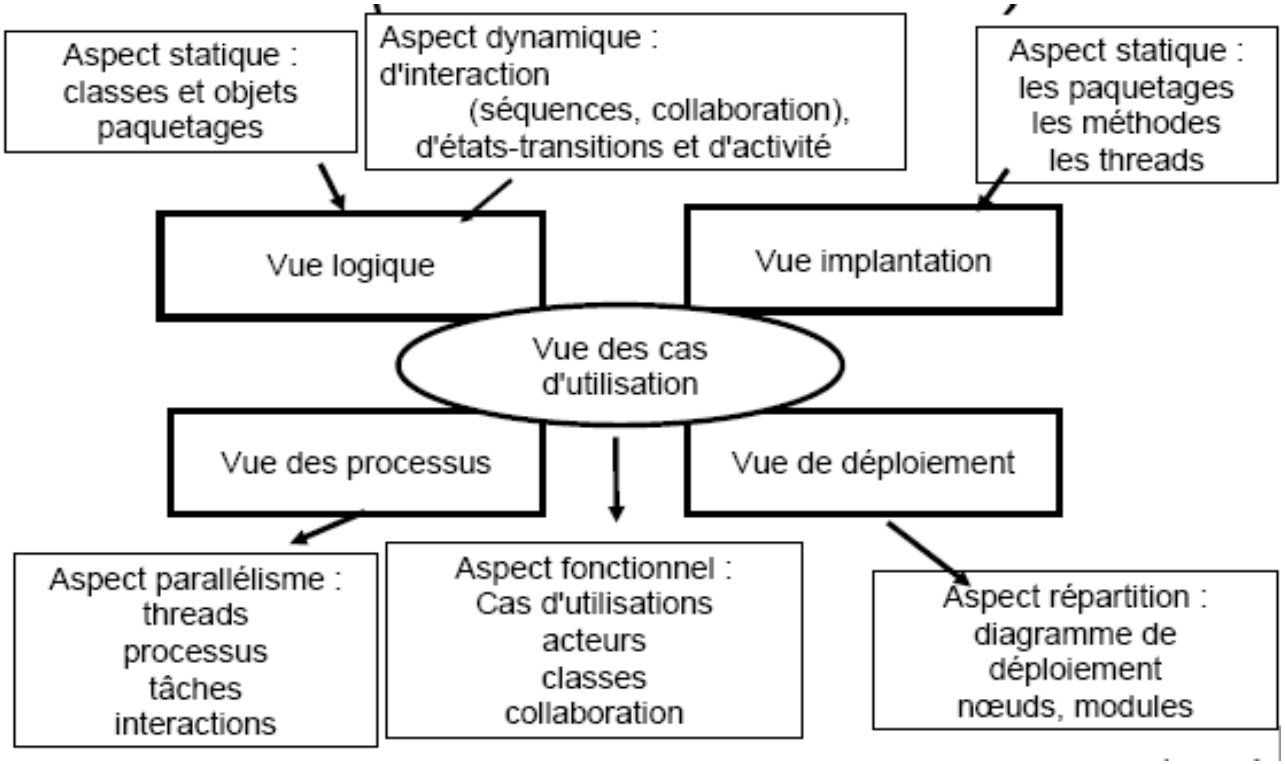
Unified Modeling language

UML

- Standard accepté pour la modélisation OO (convient pour toutes les méthodes objet).
- Langage de modélisation des systèmes utilisant des concepts OO (ce n'est pas une méthodologie)
- Langage de spécification, de visualisation, de construction et de documentation de systèmes.
- utilisable à n'importe quelle phase du processus de développement.
- Constitué de 5 vues et 9 diagrammes.

UML: vues

- Il existe 5 façons de voir le système



UML: vue Cas d'utilisation:

- Décrit les fonctionnalités attendues du système telles que perçues par les acteurs externes (utilisateurs, autres systèmes, etc.).
- personnes impliquées dans cette vue : clients, concepteurs, développeurs et testeurs.
- Diagramme essentiel: cas d'utilisation
- Vue la plus importante car le but final est de réaliser les fonctionnalités qui sont spécifiées par cette vue.

UML: vue logique

- Décrit comment les fonctionnalités du système sont réalisées.
- Décrit la structure statique (classes, objets, relations, etc.).
- Décrit la collaboration dynamique (échange de messages).
- Concerne essentiellement le concepteur et les développeurs.

UML: vue implantation

- Identifie les modules qui réalisent les classes de la vue logique.
- Montre les dépendances entre modules.
- Montre l'organisation des modules en sous systèmes (paquetages) et leurs dépendances (avec d'autres sous systèmes ou modules).
- Concerne les développeurs.

UML: vue processus

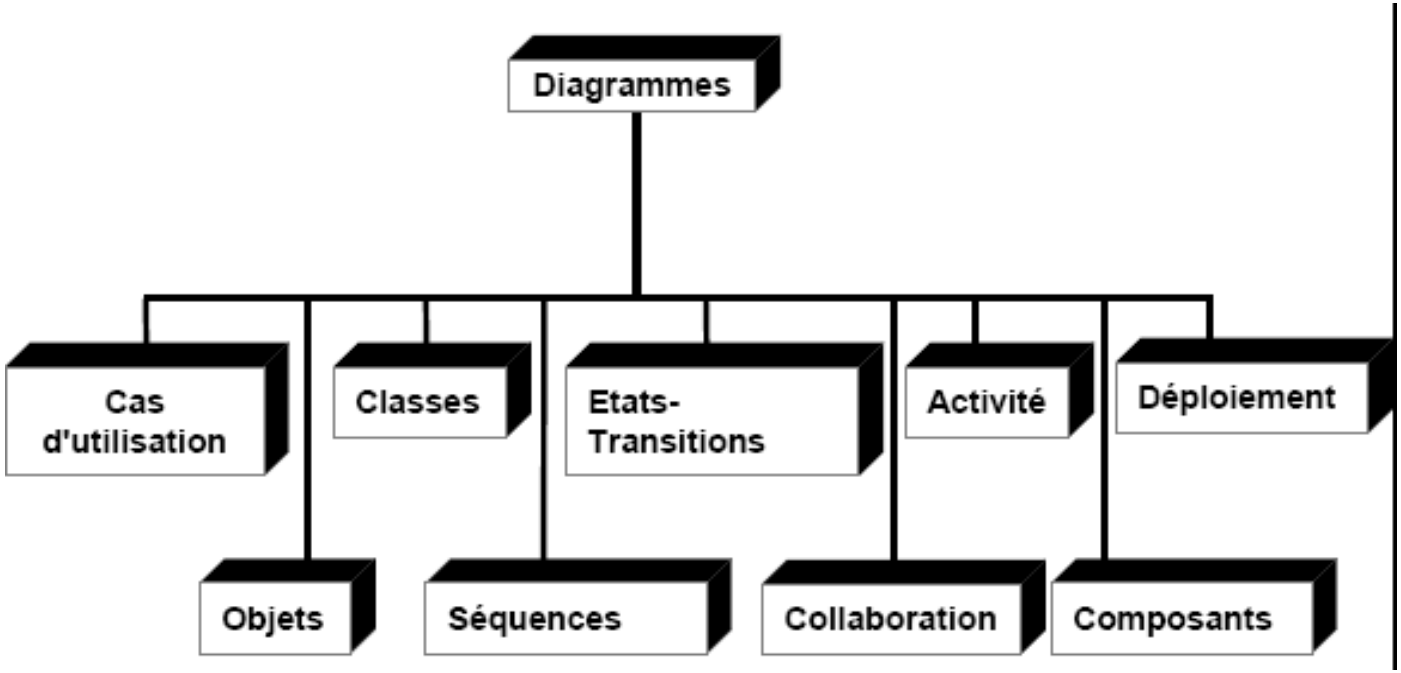
- Concerne la division du système en processus dans un environnement multi tache.
- Montre les interactions entre les processus
- Montre la synchronisation et la communication des activités parallèles (thread)
- Destinée aux développeurs et aux intégrateurs du système.

UML: vue déploiement

- Montre le déploiement physique du système
  - Décomposition en noeuds d'exécution
  - Rôle d'un noeud
  - Inter-connectivité, topologie
- Concerne les développeurs, les intégrateurs et les testeurs.

UML: diagrammes

- Il existe 9 diagrammes dans UML



UML: Diagrammes

- **Diagramme de cas d'utilisation**
  - Décrit les fonctionnalités du système telles que perçues par les acteurs externes.
- **Diagramme de classes**
  - Montre la structure statique des classes dans le système.
- **Diagramme d'objets**
  - Montre comment le système est vu à un instant donné dans le temps.

UML: Diagrammes

- **Diagramme d'états**
  - Montre les états possibles qu'un objet peut avoir en réaction aux événements (envoi/réception d'un message, condition satisfaite, etc.).
- **Diagramme de séquence**
  - Décrit les scénarios de chaque cas d'utilisation en mettant l'accent sur la chronologie des opérations en interaction avec les objets.
- **Diagramme de collaboration**
  - Représente des scénarios de cas d'utilisation en mettant plus l'accent sur les objets et les messages échangés.

UML: Diagrammes

- **Diagramme d'activité**
  - Décrit les activités qui sont exécutées dans une opération ou un cas d'utilisation.
- **Diagramme de composants**
  - Montre la structure physique du code.
- **Diagramme de déploiement**
  - Montre l'architecture physique du matériel et du logiciel dans le système.

références

- *Transparent de cours de **Nafaa Jabeur**, IFT-21453, Université de Laval.*
- *Transparent de cours de **Robert Ogor**, Modélisation avec UML, ENST Bretagne.*
- *Notes de cours de **Laurent Audibert**, UML 2.0, UIT de villetaneuse.*
- *Notes de cours de* **Jacques Lonchamp**, génie logiciel, CNAM - CRA Nancy.