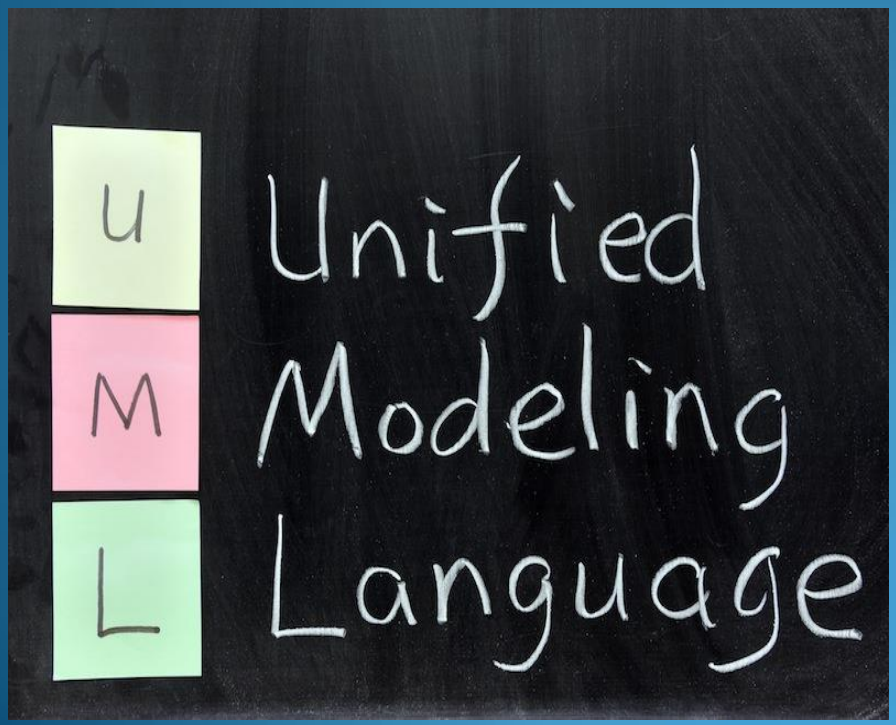
Conception et UML Cours

Naby Daouda Diakite



Mes interventions

I. Analyse du besoin : 0,5 jour

II. UML : 2,5 jours

III. Projet Ecole : 1 jour

Présentation des participants

- Formateur
- Stagiaires
- Déroulement de la formation (*combinaison des parties théoriques et pratiques*)
  - ***UML est un sujet très dense***
- Echange sur des aspects transverses de nos métiers
- Partage des expériences

Programme

I. **Analyse du besoin (TD) – 1ere Journée de cours**
II. Concepts de l'approche objet
III. Introduction à l'UML
IV. Principaux diagrammes UML
V. Diagramme de classe **(TD)**
VI. Diagramme d'objet – 2e Journée de cours
VII. Diagramme de composants
VIII. Diagramme de déploiement
IX. Diagramme des paquet
X. Diagramme de structure composite
XI. Diagramme de profil
XII. Diagramme des cas d'utilisation **(TD)**
XIII. Diagramme états-transition
XIV. Diagramme d'activité (TD) – 3e Journée de cours
XV. Diagramme de séquence **(TD)**
XVI. Diagramme de communication
XVII. Diagramme global d'interaction
XVIII. Diagramme de temps
XIX. Avantages & Limites
XX. Outils de modélisation

I. Analyse du besoin

- I.I. Objectif
- I.II. Démarche
- I.III. Méthodes de modélisation

I. Analyse du besoin

I.I. Objectif (1)

- L'analyse des besoins consiste à comprendre les enjeux et les problématiques du client ➔ ***Etape critique et indispensable***
  - Définir les acteurs et leurs rôles
  - Définir le cadre du système : « Architecture logicielle »
  - Définir les interactions des acteurs avec le système
    - Interfaces utilisateurs
    - Fonctionnalités
    - Contraintes
  - Etc.

I. Analyse du besoin

I.I. Objectif (2)

- L'analyse des besoins consiste à comprendre les enjeux et les problématiques du client ➔ ***Etape critique et indispensable***
  - Prioriser les besoins exprimés par le client
  - Accompagner le client dans l'expression de son besoin
  - Etc.

I. Analyse du besoin

I.II. Démarche (1)

- Pour cela, plusieurs étapes sont nécessaires :
  - **Etude détaillée des informations et documents fournis par le client**
    - Ces documents peuvent être :
      - Cahier des charges
      - Spécifications fonctionnelles
      - Maquettes
  - **Cadrage**
    - Des séances de travail pour échanger sur les points peu précis et poser les bonnes questions
    - Des maquettes pour mieux appréhender les attentes du client ou de ces utilisateurs

I. Analyse du besoin

I.II. Démarche (2)

- Pour cela, plusieurs étapes sont nécessaires :
  - **Rédaction de documents**
    - Spécifications fonctionnelles
      - Fonctionnalités et leurs détails
    - Spécifications techniques
      - Technologies et leurs limites
      - PoC : Proof of concept
      - Maquettes des interfaces utilisateurs
    - Architecture logicielle
      - Organisation des blocs applicatifs et leurs interactions
        - Applications mobile, web, backend, etc.

I. Analyse du besoin

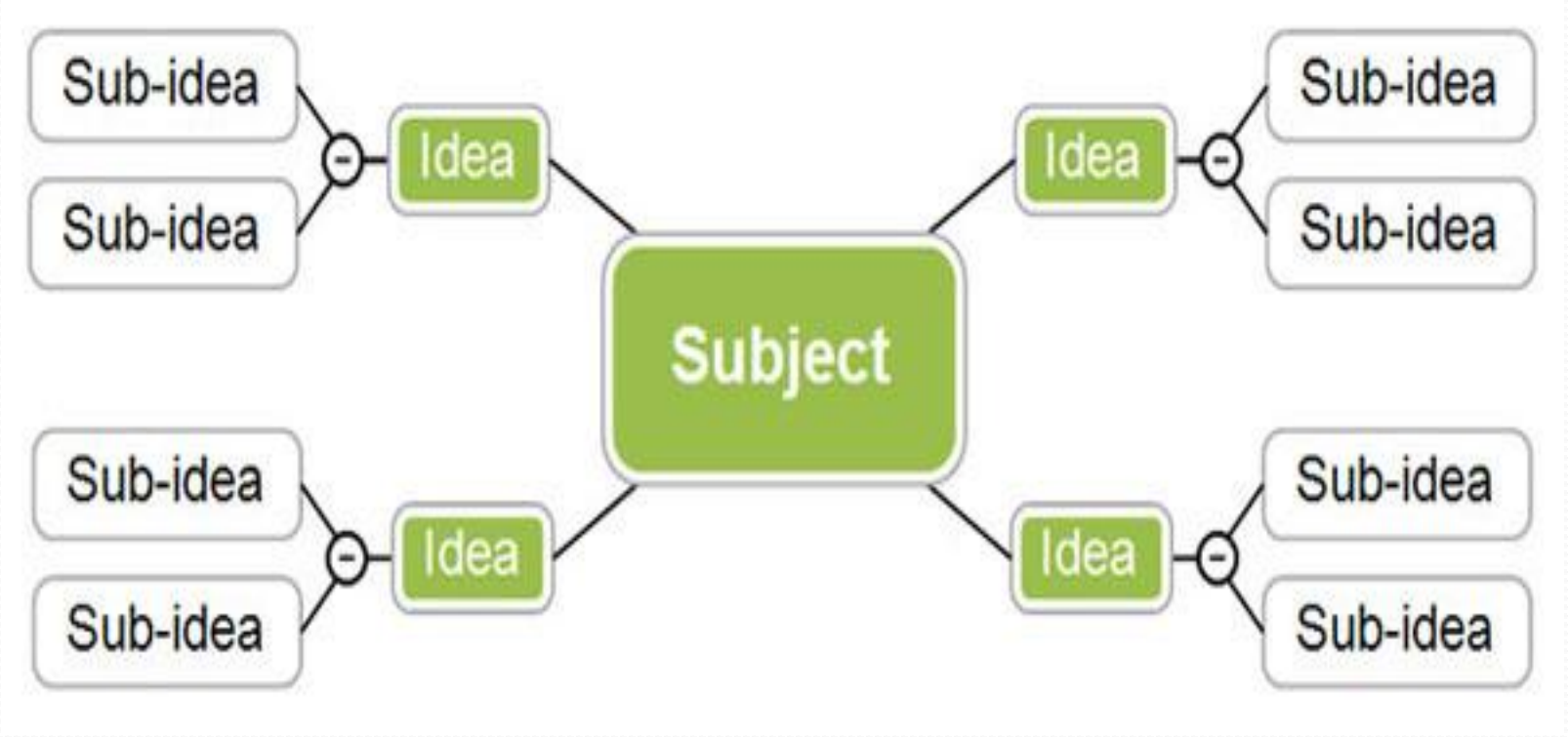
I.II. Démarche (3)

- Pour cela, plusieurs étapes sont nécessaires :
  - **Méthodes d'analyse du besoin**
    - Mind Mapping
    - Diagramme Ishikawa
    - Etc.

I. Analyse du besoin

I.II. Démarche (4)

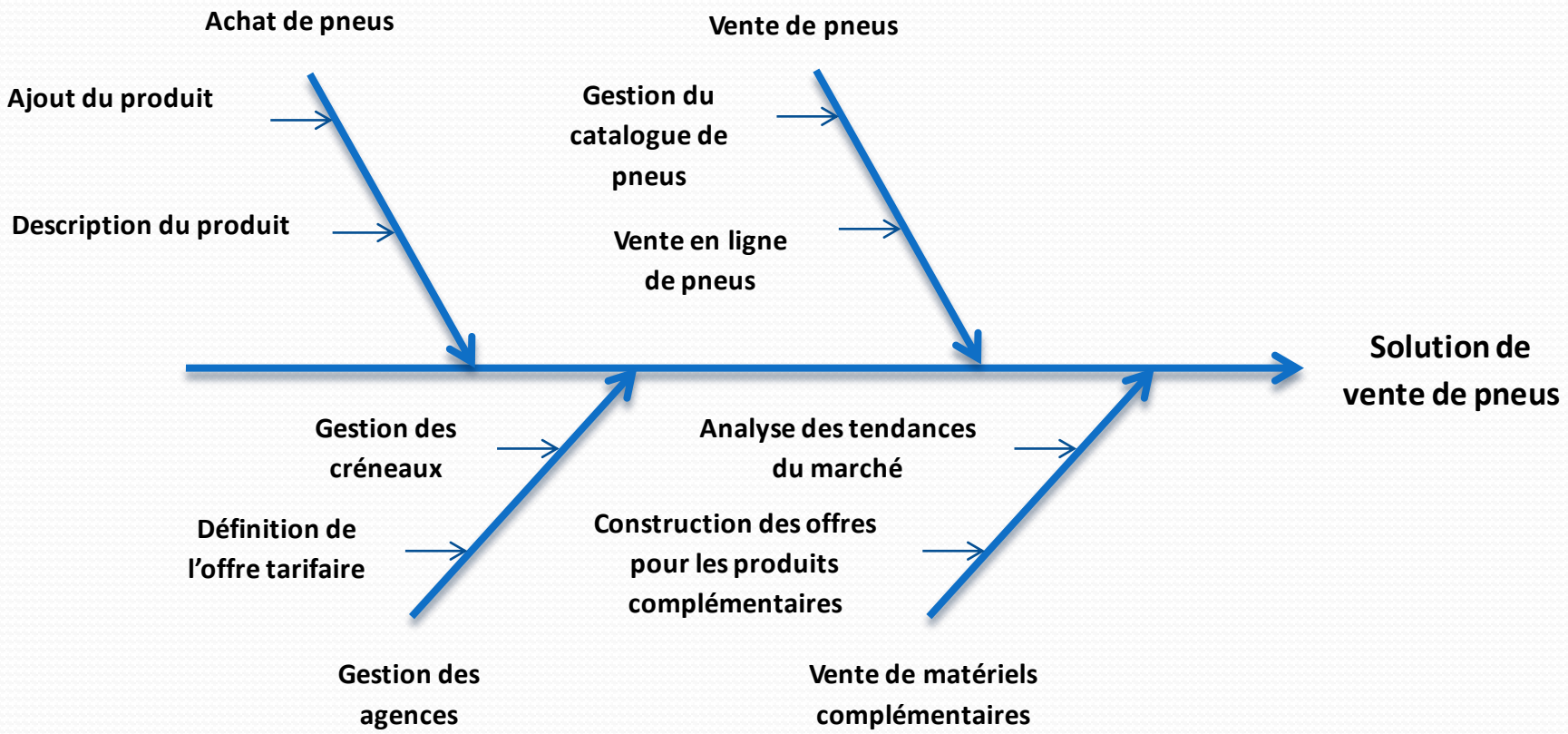
- Mind Mapping : technique de réflexion pour analyser un problème



I. Analyse du besoin

I.II. Démarche (5)

- **Diagramme Ishikawa « Solution de vente de pneus »**



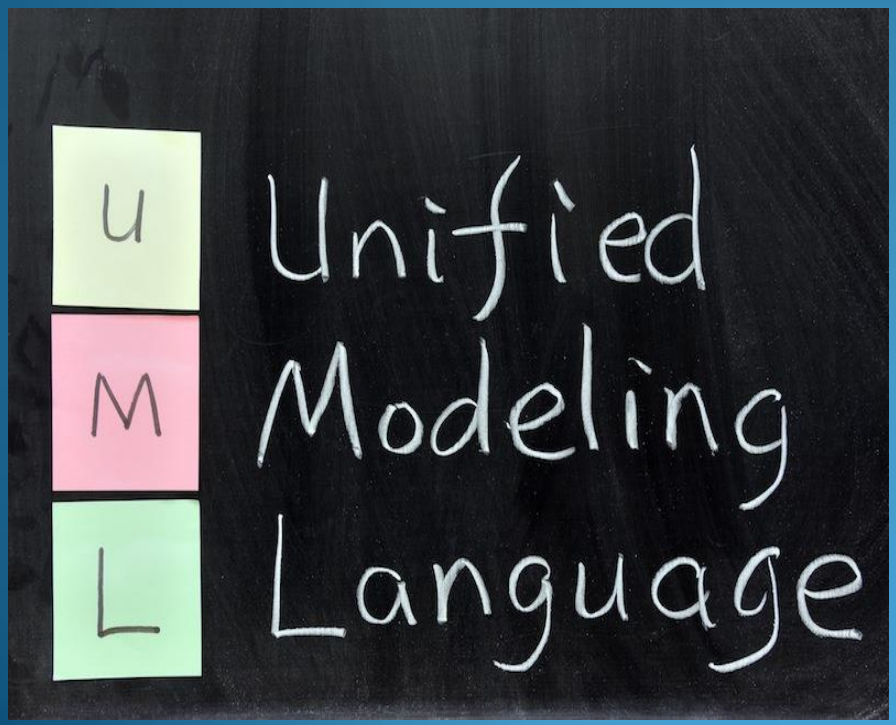
I. Analyse du besoin

I.III. Méthodes de modélisation

- Après l'analyse du besoin, il faut modéliser les besoins du client afin de construire la solution informatique

- Pour cela, différents méthodes sont utilisées :
  - Approche Objet
  - UML
  - Merise
  - Etc.

Conception et UML
Travaux Pratiques



Naby Daouda Diakite

II. Concepts de l'approche objet

- II.I. Définition
- II.II. Notion d'objet ou d'instance
- II.III. Notion de classe
- II.IV. Encapsulation
- II.V. Types de relation
- II.VI. Héritage
- II.VII. Association
- II.VIII. Contenance
- II. IX. Dépendance
- II.X. Interface
- II.XI. Polymorphisme

II. Concepts de l'approche objet

II.I. Définition

- **Programmation Orientée Objet**
  - Modéliser informatiquement des éléments d'une partie du monde réel en un ensemble d'entités informatiques (*objets*)

- **Intérêt d'une méthode objet**
  - Définir le problème à haut niveau sans rentrer dans les spécificités du langage
  - Réutilisation du code
  - Etc.

II. Concepts de l'approche objet

II.II. Notion d'objet ou d'instance

Notion d'*Objet ou d'instance*

Une abstraction du monde réel c.-à-d. des données informatiques regroupant des caractéristiques du monde réel

Exemple

une personne, une voiture, une maison, ...

Caractérisation d'un objet

- **Nom**

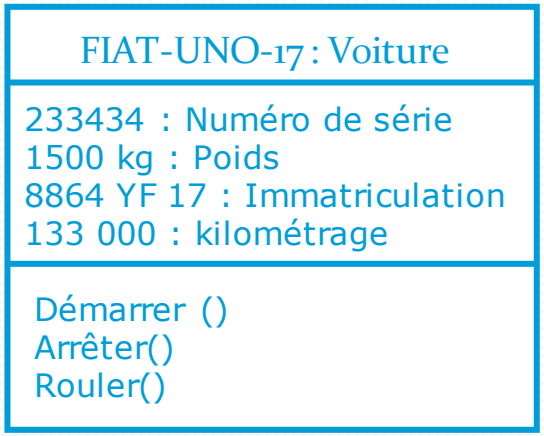
  permet de le distinguer des autres objets

- **Attributs**

  données caractérisant l'objet

- **Méthodes**

  actions que l'objet est à même de réaliser



II. Concepts de l'approche objet

II.III. Notion de classe (1)

Notion de *Classe*

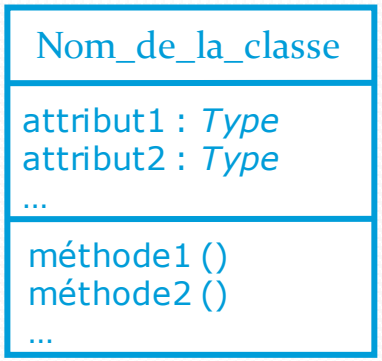
- Structure d'un objet, c.-à-d. une déclaration de l'ensemble des entités qui composeront l'objet
- Un objet est donc "issu" d'une classe, c'est le produit qui sort d'un moule

Notation

un objet est une **instanciation** (***occurrence***) d'une classe

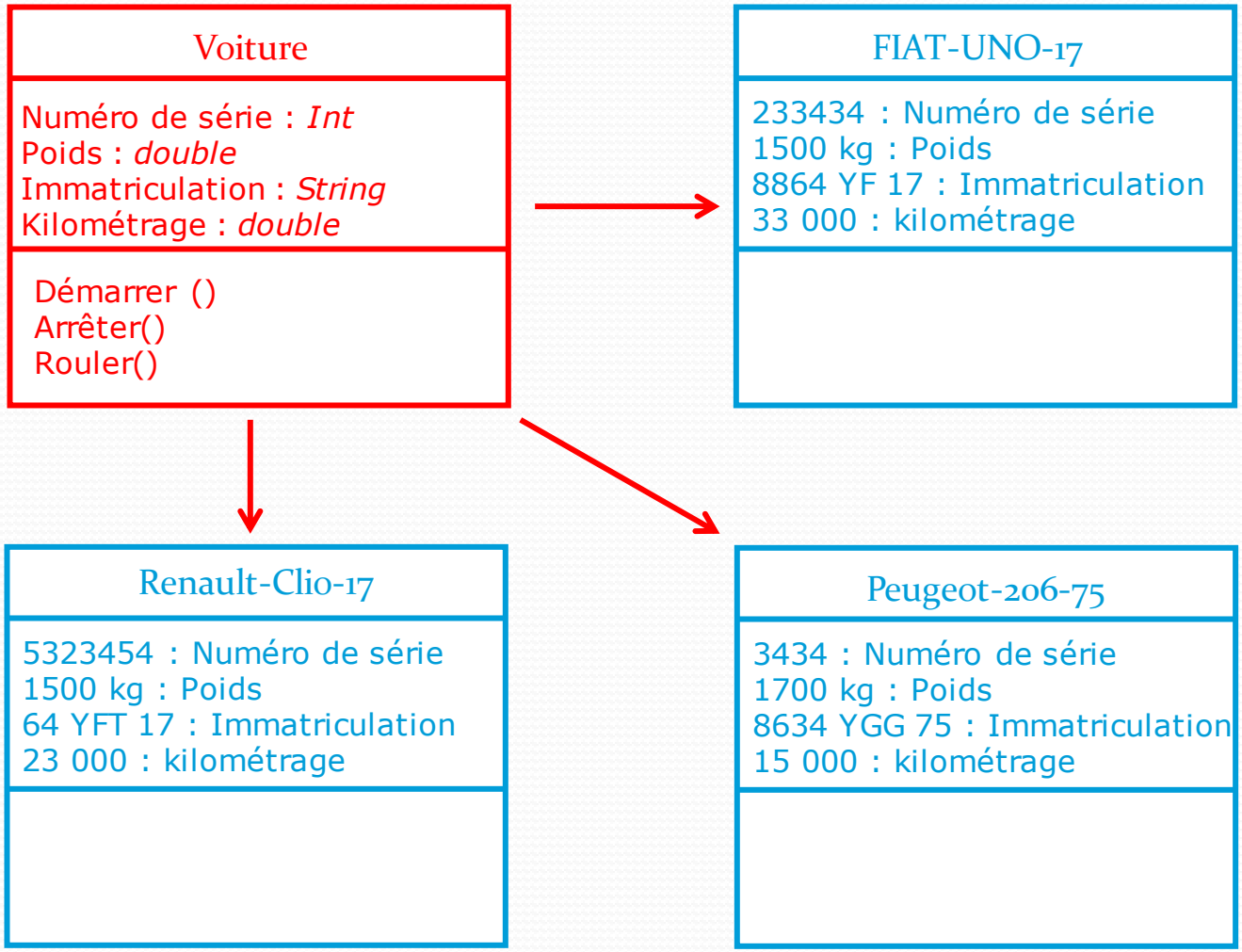
Une classe est composée:

- **Nom**
- **Attributs**
  données dont les valeurs représentent l'état de l'objet
- **Méthodes**
  opérations applicables aux objets



II. Concepts de l'approche objet

II.III. Notion de classe (2)



II. Concepts de l'approche objet

II.IV. Encapsulation

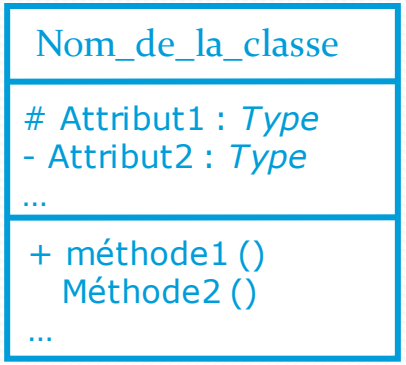
Visibilité des attributs et des méthodes

Définissent les droits d'accès aux données (pour la classe elle-même, d'une classe héritière, ou bien d'une classe quelconque)

- **Publique (+)**
les classes peuvent accéder aux données et méthodes d'une classe définie avec le niveau de visibilité *public*

- **Protégée (#)**: l'accès aux données est réservé aux fonctions des classes héritières

- **Privée (-)**: l'accès aux données est limité aux méthodes de la classe elle-même



II. Concepts de l'approche objet

II.V. Types de relation

Héritage

Association

Contenance (Agrégation et composition)

Dépendance

II. Concepts de l'approche objet

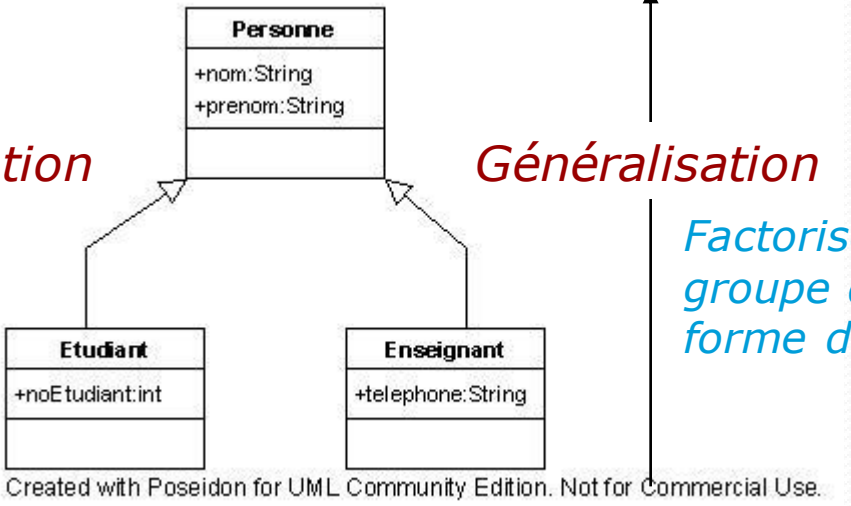
II.VI. Relation – Héritage (1)

Principe

- Permet de créer une nouvelle classe à partir d'une classe existante
- Classe dérivée contient les attributs et les méthodes de sa superclasse

Spécialisation

Etendre les propriétés d'une classe, sous forme de sous-classes



Généralisation

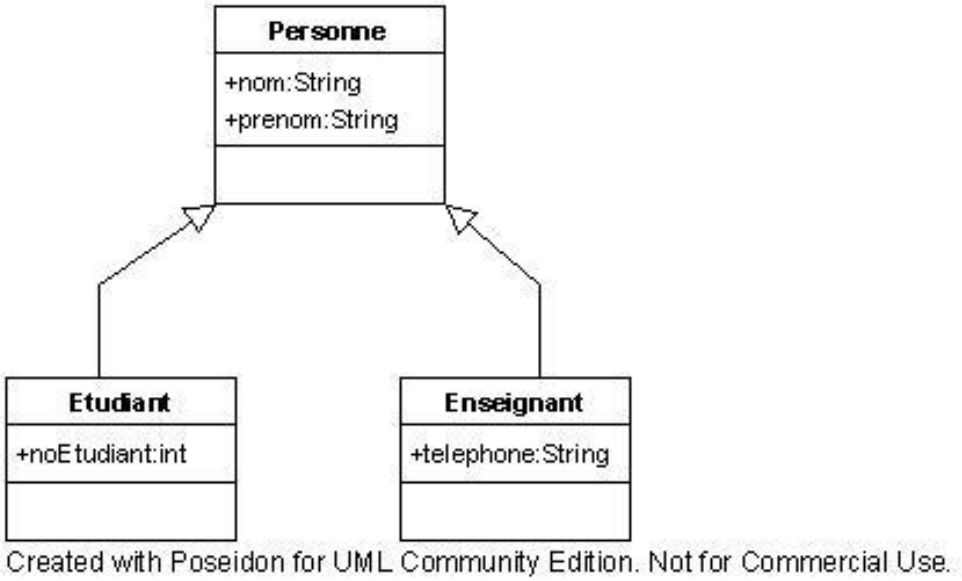
Factoriser les propriétés groupe de classes sous forme de super-classe

Chaque personne de l'université est identifiée par son nom, prénom
Les étudiants ont en plus un noEtudiant
Les enseignants ont en plus un numéro de téléphone interne

II. Concepts de l'approche objet

II.VI. Relation – Héritage (2)

```
public class Personne {
        public String nom;
        public String prenom;
}
```




```
public class Etudiant extends Personne {
        public int noEtudiant;
}
```

II. Concepts de l'approche objet

II.VII. Relation – Association (1)

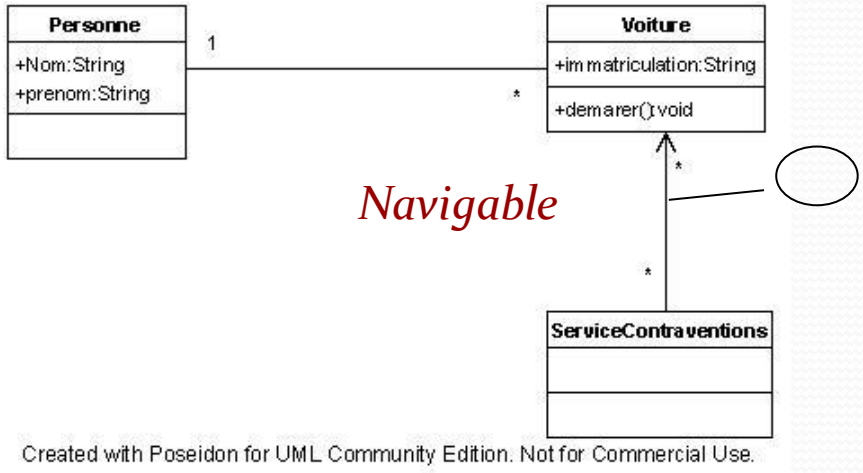
Description

- Connexion sémantique entre deux classes

Navigabilité

- Bidirectionnelle
  - Chaque instance de voiture a un lien vers le propriétaire
  - Chaque instance de Personne a un ensemble de lien vers les voitures

- Unidirectionnelle
  - Le service de contravention est associé à une ou plusieurs voiture(s)
  - La voiture ne connaît pas service de contravention



- Association interdite

II. Concepts de l'approche objet

II.VII. Relation – Association (2)

Documentation d'une association

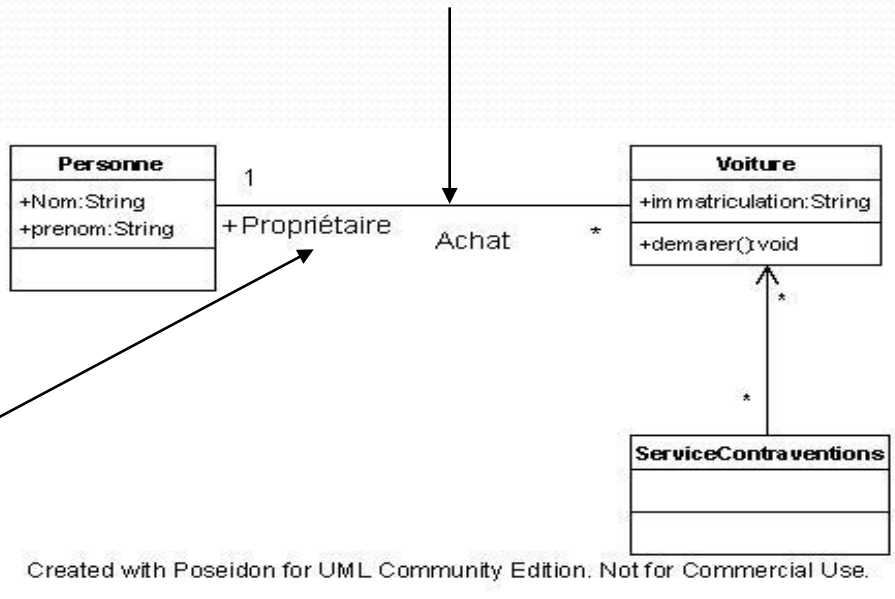
- **Nom de l'association**

  lien sémantique entre les classes

La personne achète la voiture
La voiture est achetée

- **Rôle d'une association**

  Spécification du rôle de la classe



La personne joue le rôle de propriétaire de la voiture

II. Concepts de l'approche objet

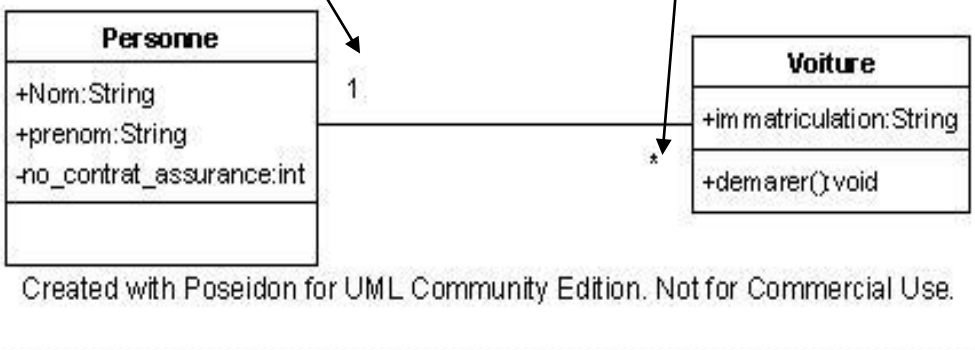
II.VII. Relation – Association (3)

Multiplicités

1 : la classe est en relation avec un et un seul objet de l'autre classe
1..* : la classe est en relation avec au moins un objet de l'autre classe
0..* : la classe est en relation avec 0 ou n objets de l'autre classe
0..1 : la classe est en relation avec au plus un objet de l'autre classe

Une voiture est achetée par une et une seule personne

Une personne peut acheter 0 ou n voitures

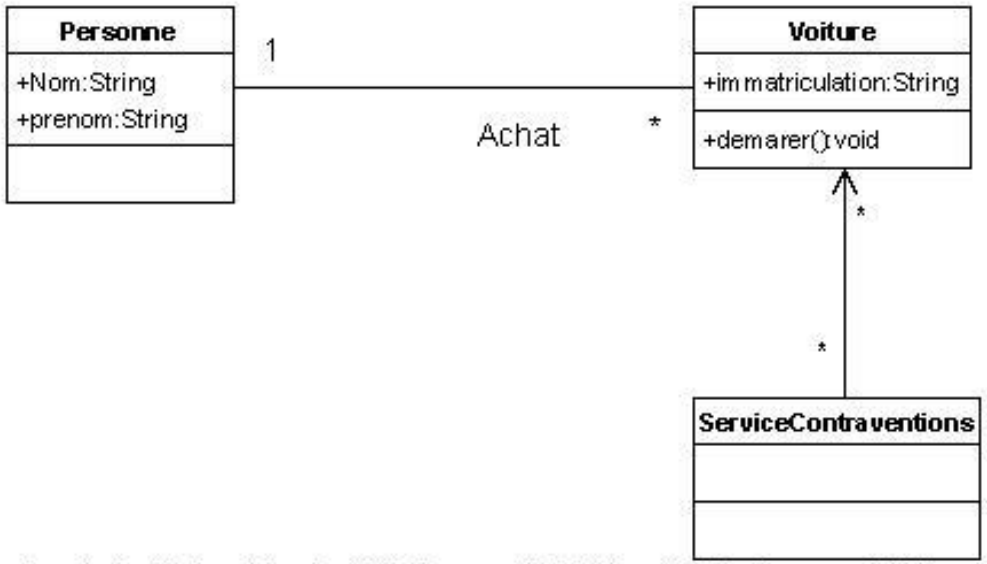


II. Concepts de l'approche objet

II.VII. Relation – Association (4)

```
public class Personne {

    public String Nom;
    public String prenom;
    public List<Voiture> voitures;
}
```




```
public class Voiture {

    public String immatriculation;
    public Personne Propriétaire;
    public void demarer() {   }
}
```

```
public class ServiceContraventions {
    public List<Voiture> voitures;
}
```

II. Concepts de l'approche objet

II.VIII. Relation – Contenance (1)

Deux types de relations de contenance en UML

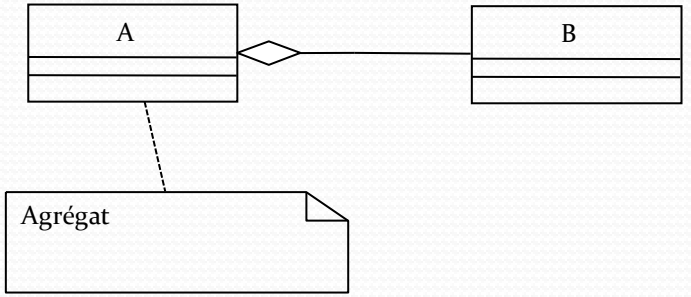
- Agrégation 
- Composition (Agrégation forte) 

Exemples

- Une présentation PowerPoint est composé de transparents
- Une équipe de recherche est composée d'un ensemble de personnes

II. Concepts de l'approche objet

II.VIII. Relation – Contenance (2)



Agrégation

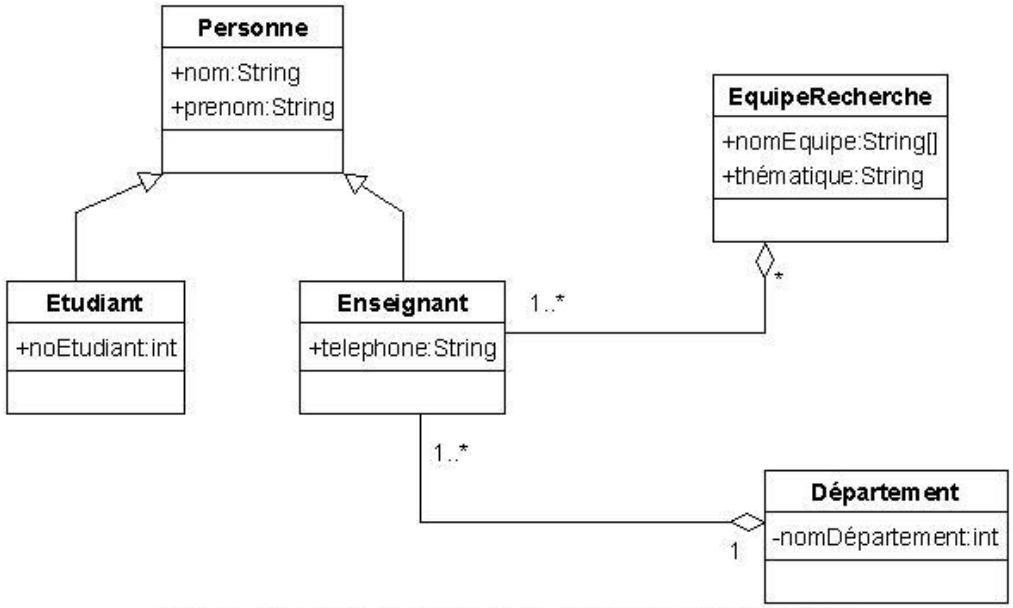
- A « contient » des instances de B,

Propriétés de l'agrégation

- La suppression de A n'implique pas la suppression de B
- L'élément agrégé peut être partagé

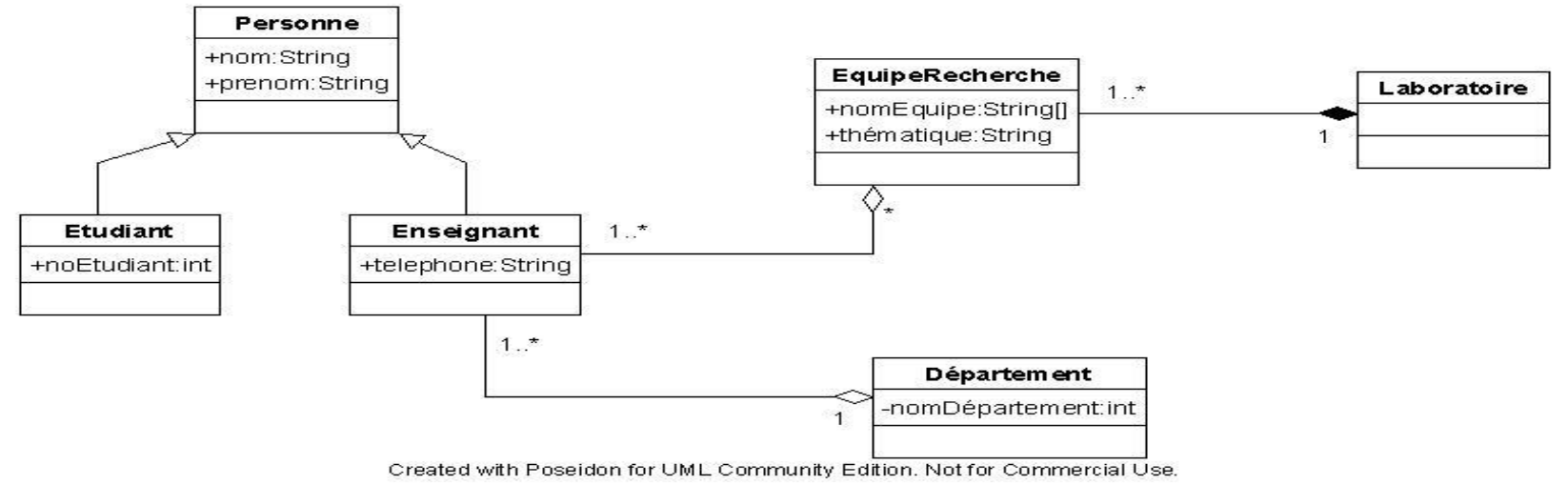
Exemples :

- L'enseignant est un composant d'une (ou plusieurs) équipe de recherche d'un seul département
- La disparition d'une équipe de recherche n'entraine pas la disparition d'un enseignant



II. Concepts de l'approche objet

II.VIII. Relation – Contenance (3)




```
public class Enseignant extends Personne {
  public String telephone;
  public List<EquipeRecherche> equipeRecherches;
  public Departement departement;
 }
```

```
public class Département {
  private int nomDépartement;
  private int codetheme;
  public List<Enseignant> enseignants;
}
```

II. Concepts de l'approche objet

II.VIII. Relation – Contenance (4)

Composition

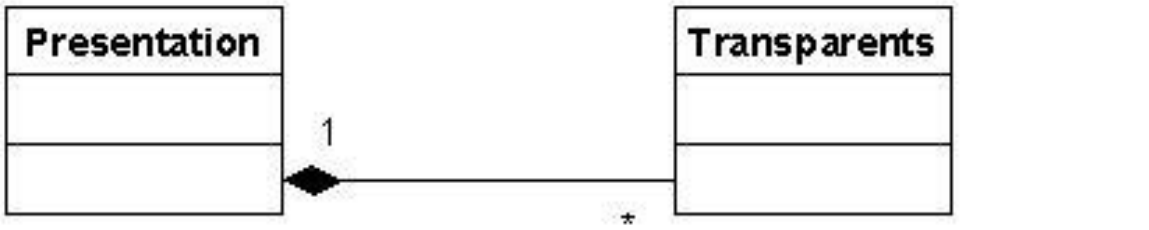
- La suppression de A entraine la suppression de B

Propriétés de l'agrégation

- La suppression de la présentation entraine la disparition des transparents qui la compose

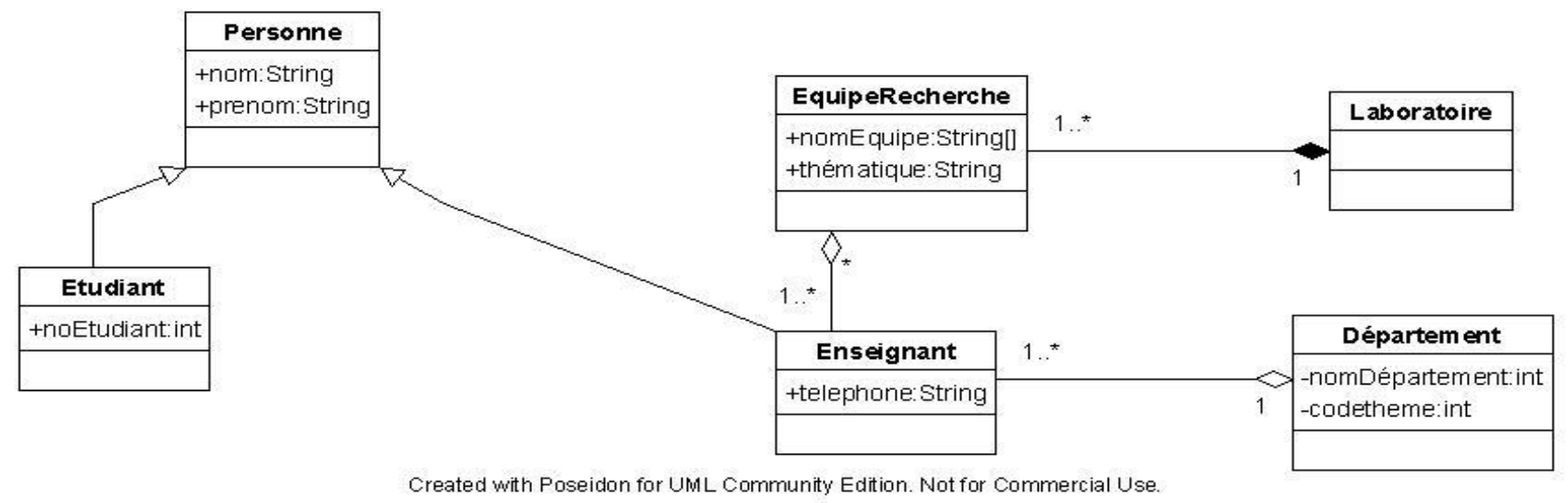
Exemples

- *Une présentation PowerPoint est composé de transparents »*



II. Concepts de l'approche objet

II.VIII. Relation – Contenance (5)




```
public class EquipeRecherche {
  public String[] nomEquipe;
  public String thématique;
  public java.util.Collection enseignant = new java.util.TreeSet();
  public Laboratoire laboratoire;
}
```

```
public class Laboratoire {
  public List<EquipeRecherche> equipeRecherches;
}
```

II. Concepts de l'approche objet

II.IX. Relation – Dépendance (1)

Description

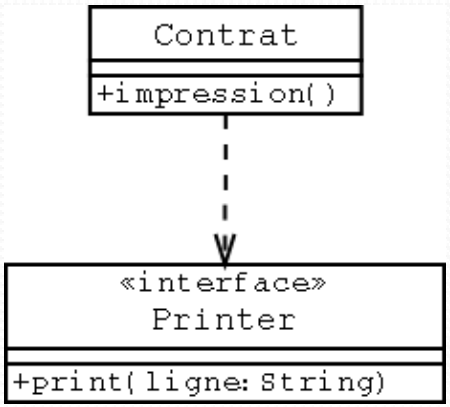
- La suppression de A n'entraine pas la suppression de B

- Lorsque cette relation est réalisée par des liens entre objets, ces derniers sont limités dans le temps, contrairement à d'autres relations plus structurelles

Exemple

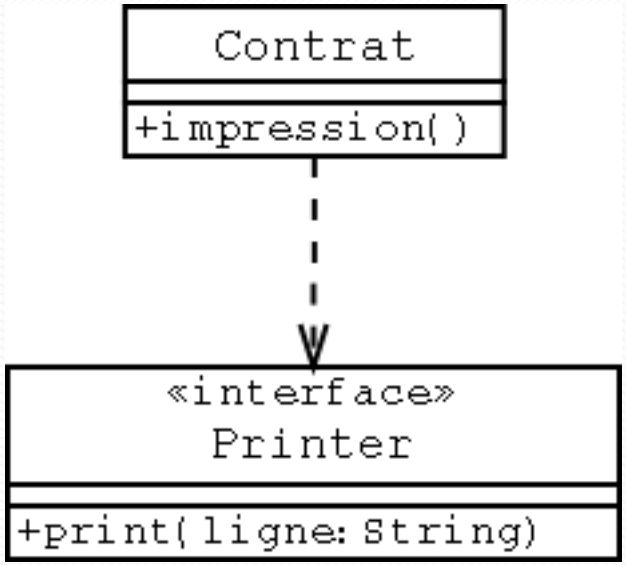
« Un contrat est dépendant d'une imprimante »

La suppression de l'imprimante n'entraine pas la disparition des contrats, ni inversement



II. Concepts de l'approche objet

II.IX. Relation – Dépendance (2)




```
public class Contrat {
  ...
  public void impression() {
    Printer imprimante = PrinterFactory.getInstance();
    ...
    imprimante.print(client.getName());
    ...
  }
}
```

II. Concepts de l'approche objet

II.X. Interface (1)

Description

- **Le concept d'interface** est utilisée pour représenter les fonctions transverses de classes. Une interface nous dit que telle classe possède telle propriété, indépendamment de ce qu'elle représente.
- Permet de gérer l'héritage dans certains langages (*par exemple en Java*)
- L'héritage permet de masquer l'implémentation et d'éviter une forte dépendance

Exemple

```
package fr.dtek.dms.core.service.zformationuml;

public interface PersonInterface {
    void walk();
    void run();
}
```

II. Concepts de l'approche objet

II.X. Interface (2)

Exemple

```
package fr.dtek.dms.core.service.zformationuml;

public class Doctor implements PersonInterface {
    @Override
    public void walk() {
        System.out.print("\nDoctor walking\n");
    }

    @Override
    public void run() {
        System.out.print("\nDoctor running\n");
    }
}
```

II. Concepts de l'approche objet

II.X. Interface (3)

Exemple

```
package fr.dtek.dms.core.service.zformationuml;

public class Soldier implements PersonInterface {
    @Override
    public void walk() {
        System.out.print("\nSoldier walking\n");
    }

    @Override
    public void run() {
        System.out.print("\nSoldier running\n");
    }
}
```

II. Concepts de l'approche objet

II.X. Interface (4)

Exemple

```
package fr.dtek.dms.core.service.service;
import fr.dtek.dms.core.service.zformationuml.Doctor;
import fr.dtek.dms.core.service.zformationuml.PersonInterface;
import fr.dtek.dms.core.service.zformationuml.Soldier;
import org.junit.Test;

public class _zFormationUml {
    @Test
    public void runApplication() {
        // Doctor
        PersonInterface doctor = new Doctor();
        doctor.walk();
        doctor.run();

        // Doctor
        PersonInterface soldier = new Soldier();
        soldier.walk();
        soldier.run();
    }
}
```

II. Concepts de l'approche objet

II.X. Interface (5)

Résultat de l'exemple

On peut changer les implémentations sans impacter l'utilisation des services

Doctor walking

Doctor running

Soldier walking

Soldier running

II. Concepts de l'approche objet

II.XI. Polymorphisme (1)

Description

- **Le concept de polymorphisme** se base sur l'héritage. Le principe est le suivant.
  - Si la classe B hérite de la classe A ➔ B "EST-UNE" Classe A
  - Toute méthode de A peut-être invoquée sur une instance de la classe B
  - Le polymorphisme consiste à exploiter cela en fournissant un B dans les expressions "qui attendent" un A.

Exemple

```
package fr.dtek.dms.core.service.zformationuml;

public class PersonBase {
    private String fullName;

    public PersonBase(String fullName) {
        this.fullName = fullName;
    }

    public void displayInformation() {
        System.out.print("\nFull name : " + fullName + "\n");
    }
}
```

II. Concepts de l'approche objet

II.XI. Polymorphisme (2)

Exemple

```
package fr.dtek.dms.core.service.zformationuml;

public class Doctor extends PersonBase {
    public Doctor(String fullName) {
        super(fullName);
    }
}
```

II. Concepts de l'approche objet

II.XI. Polymorphisme (3)

Exemple

```
package fr.dtek.dms.core.service.zformationuml;

public class Soldier extends PersonBase {
    public Soldier(String fullName) {
        super(fullName);
    }
}
```

II. Concepts de l'approche objet

II.XI. Polymorphisme (4)

Exemple

```
package fr.dtek.dms.core.service.service;
import fr.dtek.dms.core.service.zformationuml.Doctor;
import fr.dtek.dms.core.service.zformationuml.PersonBase;
import fr.dtek.dms.core.service.zformationuml.Soldier;
import org.junit.Test;

public class _zFormationUml {
    @Test
    public void runApplication() {
        // Doctor
        PersonBase doctor = new Doctor("Louis Legrand");
        doctor.displayInformation();

        // Doctor
        PersonBase soldier = new Soldier("Laura Dupuis");
        soldier.displayInformation();
    }
}
```

II. Concepts de l'approche objet

II.XI. Polymorphisme (5)

Résultat de l'exemple

Full name : Louis Legrand

Full name : Laura Dupuis

III. Introduction à l'UML

- III.I. Historique
- III.II. Evolution
- III.III. Définition

III. Introduction à l'UML

III.I. Historique (1)

- L'orientation fonctionnelle (années 1970)
  - Programmation structurée, méthodes d'analyse structurées.
- L'orientation conceptuelle (années 1980)
  - méthodes de conception systémique (Données et traitement)
- L'orientation Objet (les années 1990)
  - Programmation Objet
  - Méthodes d'analyse et de conception orientée Objet
- **1990: Naissance d'une cinquantaine de méthodes orienté objet**
  - **Confusion autour de l'analyse et de la conception**

III. Introduction à l'UML

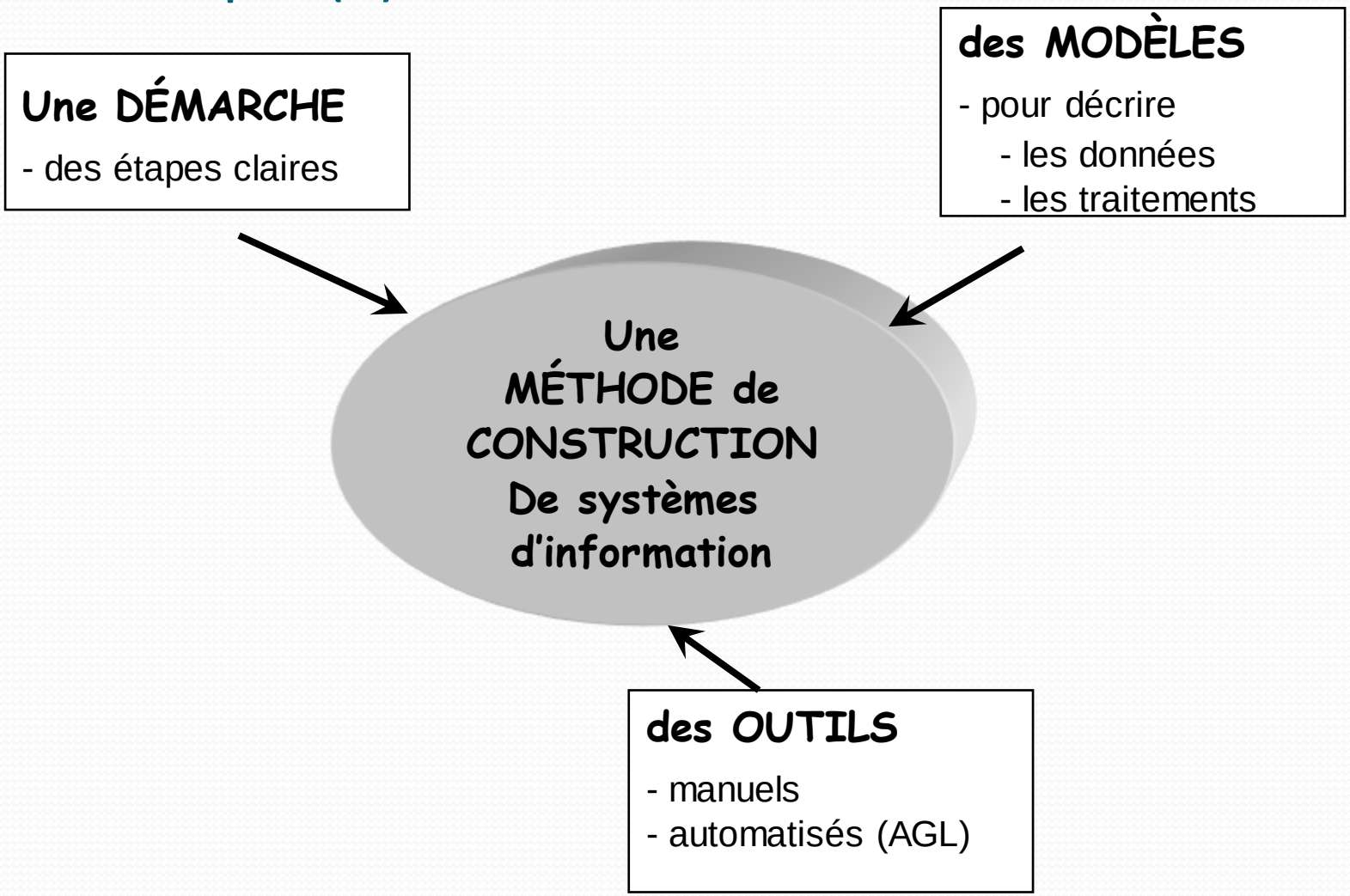
III.I. Historique (2)

- Idée : Examen des méthodes dominantes pour permettre de dégager un consensus autour d'idées communes.

- Besoin d'ingénierie et de méthodes formelles

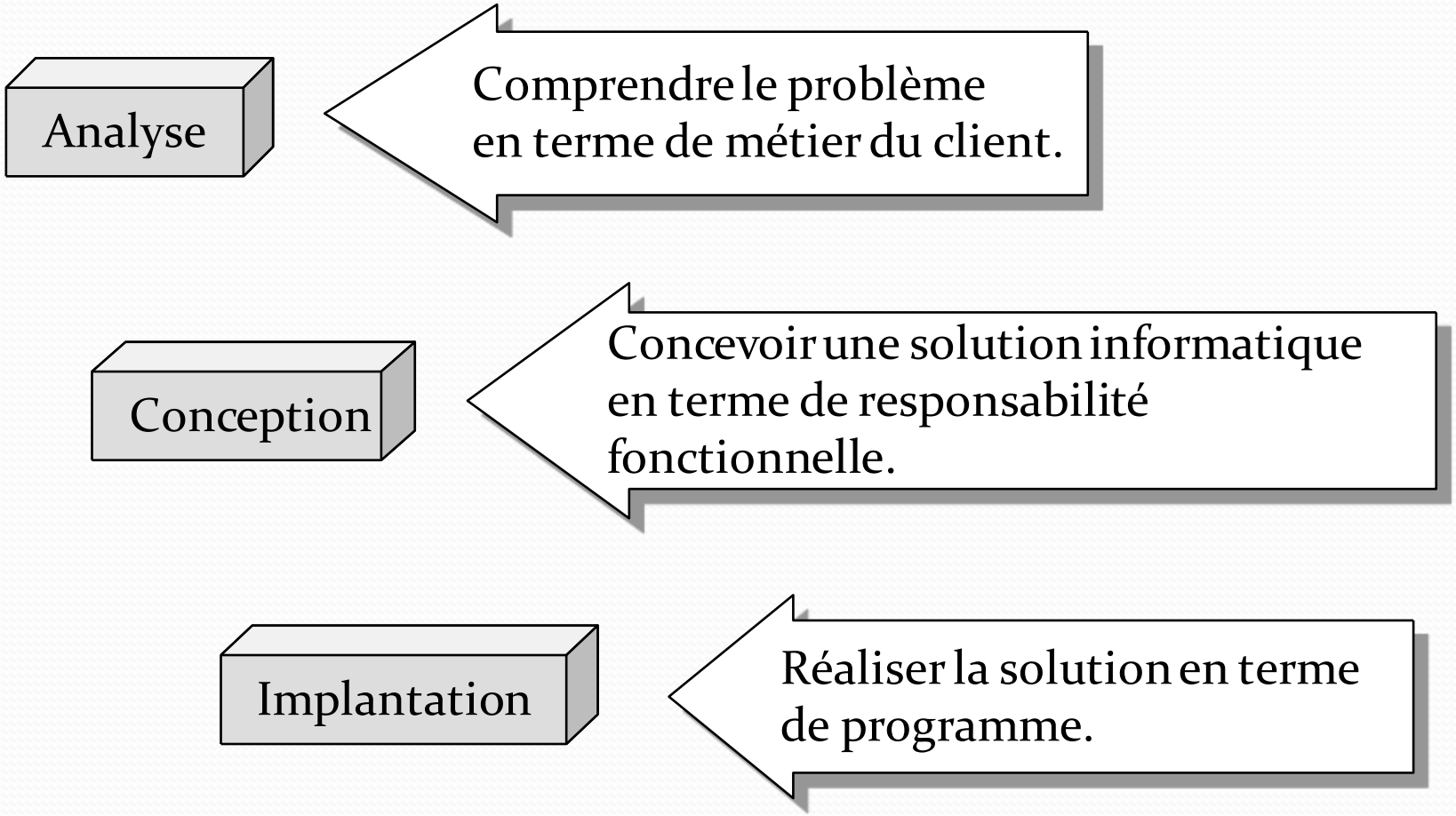
III. Introduction à l'UML

III.I. Historique (3)



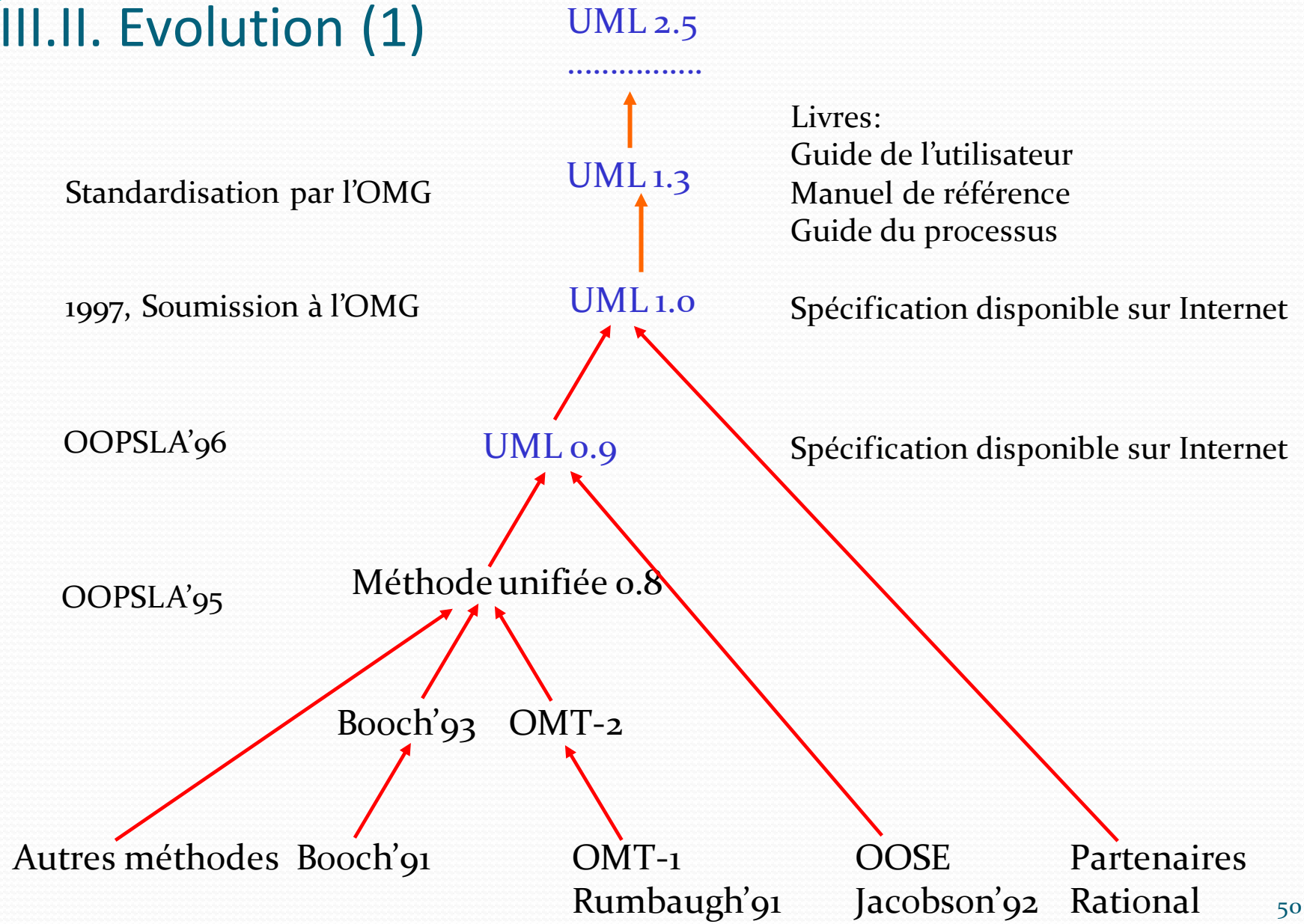
III. Introduction à l'UML

III.I. Historique (4)



III. Introduction à l'UML

III.II. Evolution (1)



III. Introduction à l'UML

III.II. Evolution (2)

- 10-1996 : UML 0.91
- 10-1998 : Adoption de UML 1.3 par l'**OMG (Object Management Group)**
  - 9 types de diagrammes
- 11-2007 : Diffusion de UML 2.1.2 par l'OMG
  - 14 types de diagrammes
- 05-2010 : Diffusion de UML 2.3 par l'OMG
- **09-2013 : Diffusion par l'OMG d'UML 2.5**

III. Introduction à l'UML

III.III. Définition

- UML est une **notation, pas une méthode**
- UML est un **langage de modélisation objet**
- UML convient pour toutes les méthodes objet
- UML est dans le domaine public

IV. Principaux diagrammes UML

- IV.I. Types de diagrammes UML
- IV.II. Diagrammes de structure ou diagrammes statiques
- IV.III. Diagrammes de comportement
- IV.IV. Diagrammes d'interaction ou diagrammes dynamiques
- IV.V. Notations communes

IV. Principaux diagrammes UML

IV.I. Types de diagrammes UML (1)

- **14 types diagrammes UML**

- Repartis dans **3 grands groupes de diagrammes**
  - Diagrammes de structure ou diagrammes statiques
  - Diagrammes de comportement
  - Diagrammes d'interaction ou diagrammes dynamiques

IV. Principaux diagrammes UML

IV.II. Diagramme de structure

- Description
  - Permet de décrire les aspects structurels, figés des éléments

- Types
  - **Diagramme de classes**
  - Diagramme d'objets
  - Diagramme de composants
  - Diagramme de déploiement
  - Diagramme des paquets
  - Diagramme de structure composite
  - Diagramme de profils

IV. Principaux diagrammes UML

IV.III. Diagramme de comportement

- Description
  - Permet de décrire le comportement, le fonctionnement des éléments

- Types
  - **Diagramme des cas d'utilisation**
  - Diagramme états-transitions
  - **Diagramme d'activité**

IV. Principaux diagrammes UML

IV.IV. Diagramme d'interaction

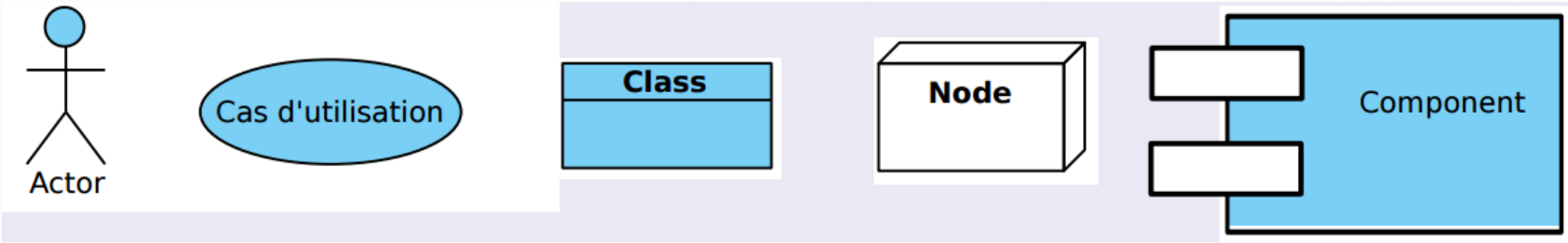
- Description
  - Permet de décrire les interactions entre les éléments

- Types
  - **Diagramme de séquence**
  - Diagramme de communication
  - Diagramme global d'interaction
  - Diagramme de temps

IV. Principaux diagrammes UML

IV.V. Notations communes (1)

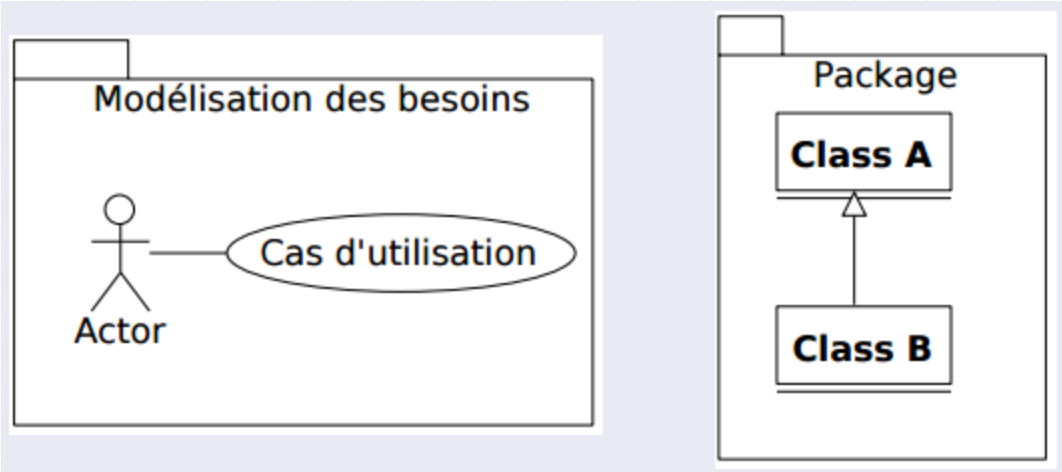
- **Classeur**
  - Un classeur est un élément de modèle qui est doté d'une identité, possède des caractéristiques structurelles (attributs, participation à des relations) et comportementales (opérations).



IV. Principaux diagrammes UML

IV.V. Notations communes (2)

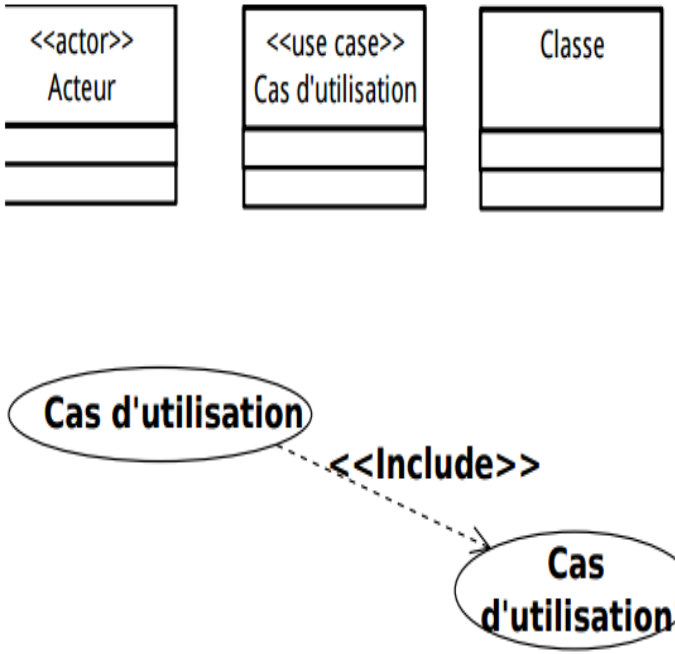
- **Paquetage**
  - Un paquetage est un regroupement d'éléments de modèle ou de diagrammes



IV. Principaux diagrammes UML

IV.V. Notations communes (3)

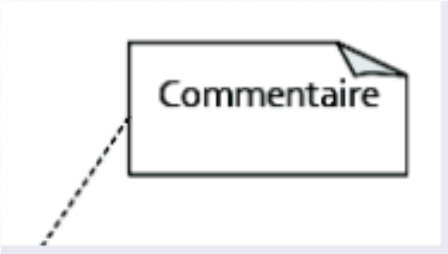
- **Stéréotype**
  - Annotation s'appliquant sur un élément de modèle
  - Utilisation particulière d'éléments de modélisation avec interprétation (sémantique) particulière
  - Modification de la signification d'un élément
  - Notation : « nomDuStéréotype » avant le nom de l'élément auquel il s'applique ou icône associée
  - Prédéfinis : « actor », « includes », « use case », « interface », « include » … « class » stéréotype par défaut d'un classeur



IV. Principaux diagrammes UML

IV.V. Notations communes (4)

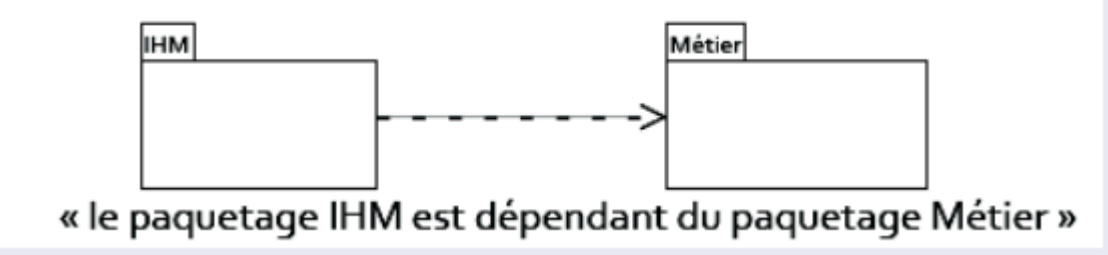
- **Commentaire**
  - Information textuelle (explication utile, observations, renvoi vers document), pas de contenu sémantique



IV. Principaux diagrammes UML

IV.V. Notations communes (5)

- **Relations de dépendance**
  - Modification de la cible peut impliquer une modification de la source
  - La source dépend de la cible
  - Notation : [source]---------->[cible]

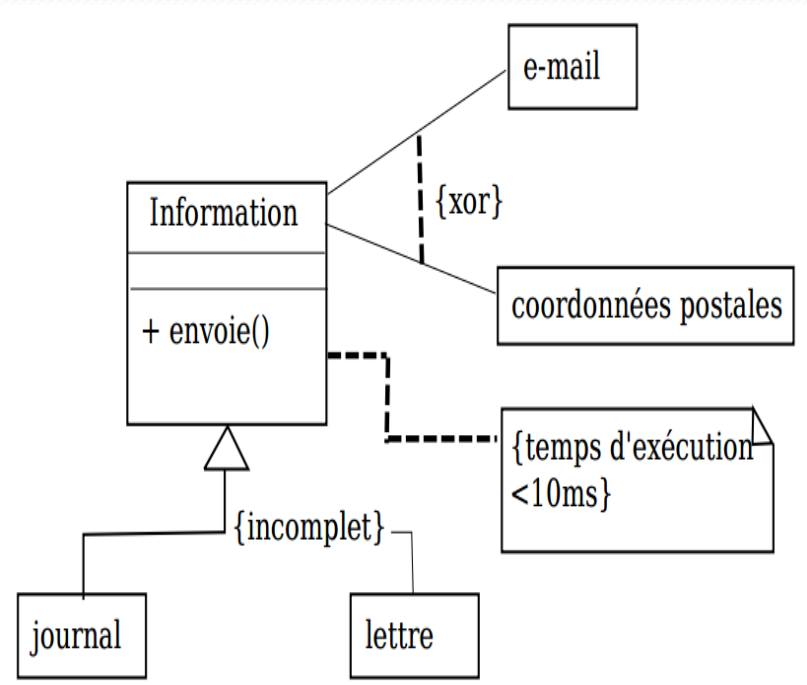


  - Toute modification dans paquetage métier peut avoir des conséquences sur paquetage IHM.
  - Toute modification dans paquetage IHM n'a au vu de cette dépendance pas de conséquence sur paquetage métier.
  - Nombreux stéréotypes prédéfinis : « extend », « include », …

IV. Principaux diagrammes UML

IV.V. Notations communes (6)

- **Contrainte**
  - Relation entre éléments de modélisation
  - Définition de propriétés devant être vérifiées pour garantir la validité du système modélisé
  - Notation : – – – – – – – – – {contrainte}
  - Stéréotypes : « précondition », « postcondition »



V. Diagramme de classe

- V.I. Description
- V.II. Eléments de base
- V.III. Exemple
- V.IV. Elaboration

V. Diagramme de classe

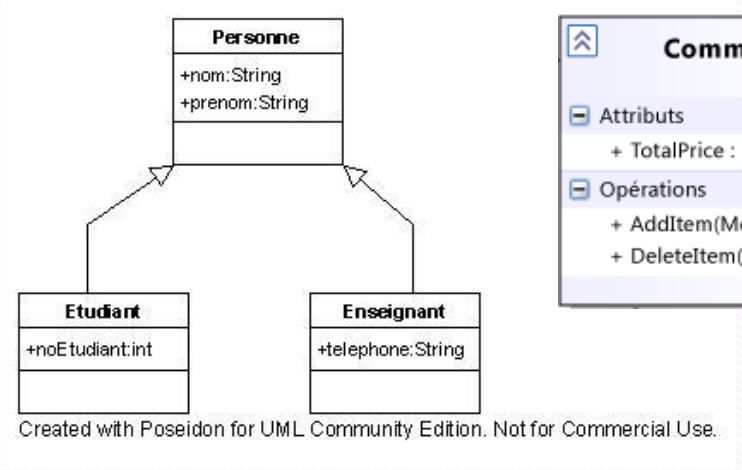
V.I. Description

- Décrit les classes d'une application et leurs relations statiques
- Une classe est un ensemble de fonctions et de données (attributs) qui sont liées ensemble par un champ sémantique.
- Les classes sont utilisées dans la programmation orientée objet.
- Elles permettent de modéliser un programme et ainsi de découper une tâche complexe en plusieurs petits travaux simples.

V. Diagramme de classe

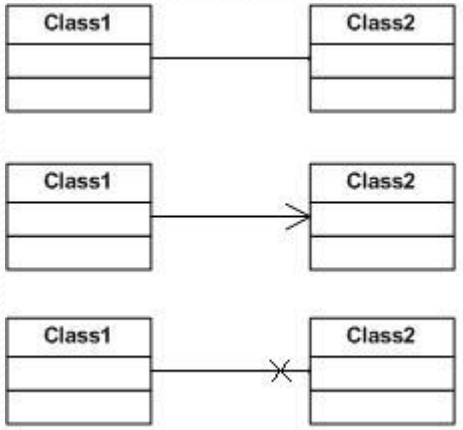
V.II. Eléments de base

Héritage



Classe

Association



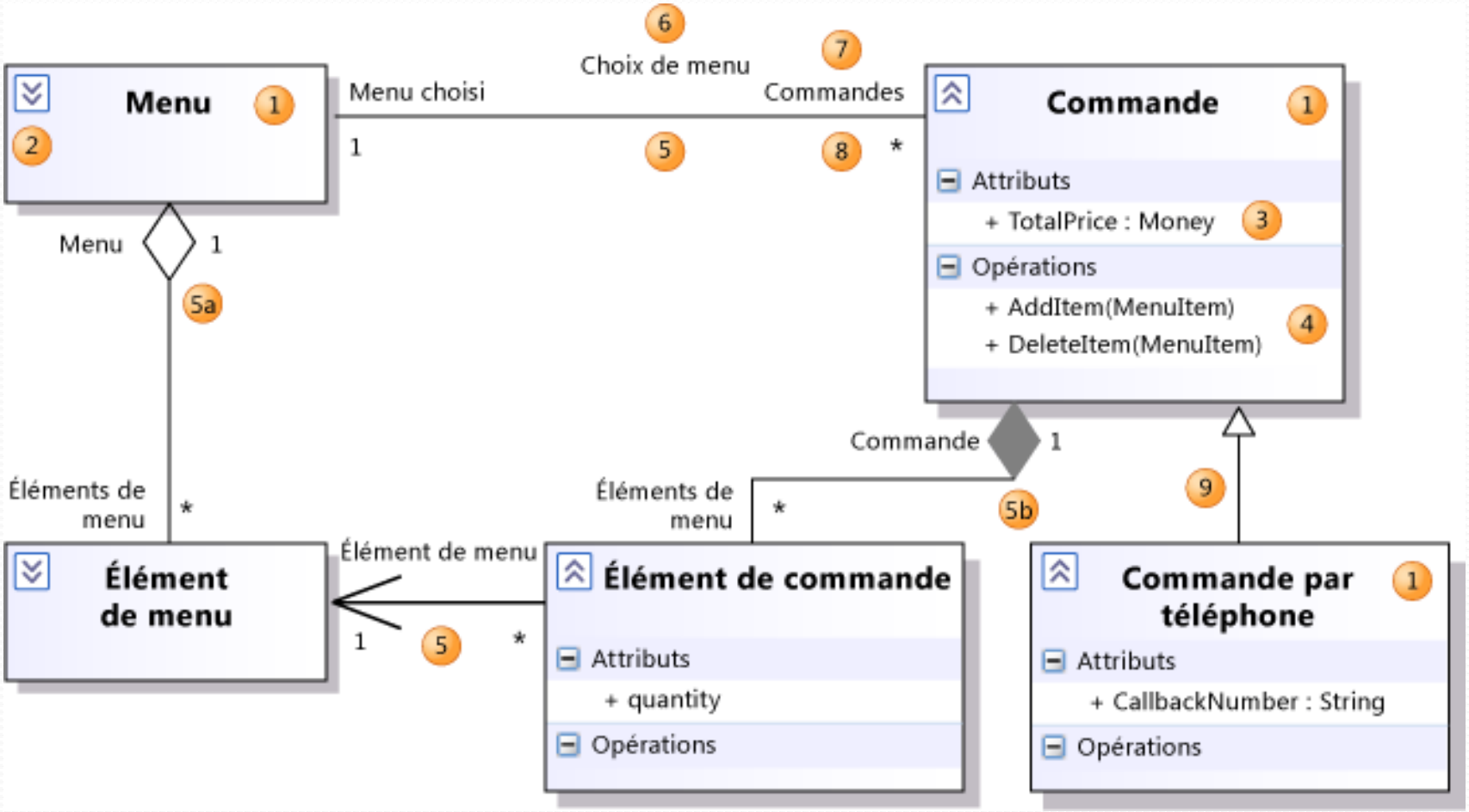
Agrégation

Dépendance

Composition

V. Diagramme de classe

V.III. Exemple



V. Diagramme de classe

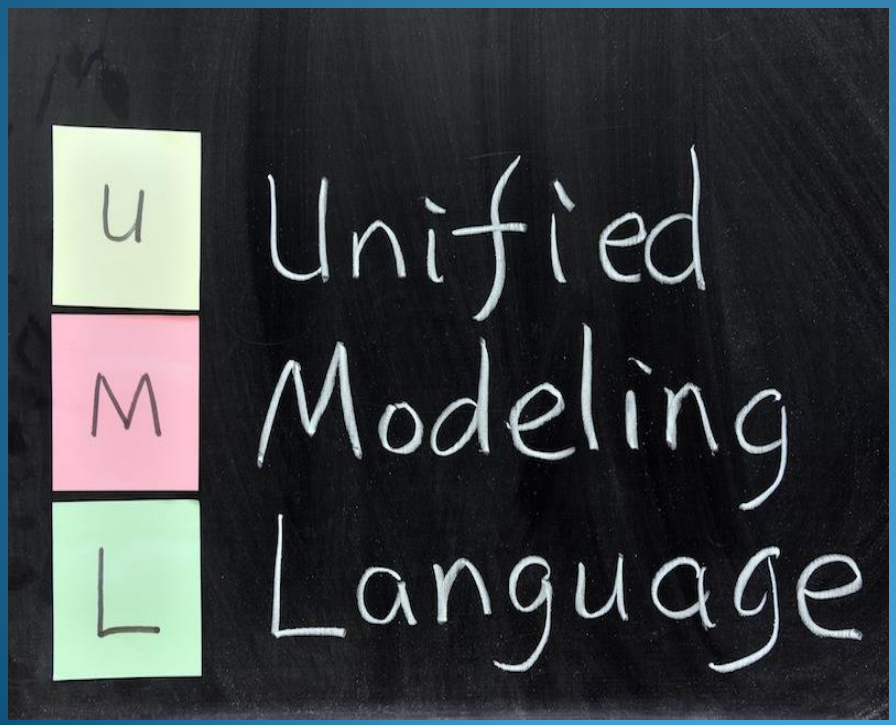
V.IV. Elaboration

- Analyser le besoin métier

- Identifier les classes
  - Nom
  - Attributs & Visibilités
  - Spécialisation
  - Généralisation

- Identifier les liens entre les classes
  - Associations
  - Dépendances

Conception et UML
Travaux Pratiques



Naby Daouda Diakite

VI. Diagramme d'objet

- VI.I. Description
- VI.II. Eléments de base
- VI.III. Exemple
- VI.IV. Elaboration

VI. Diagramme d'objet

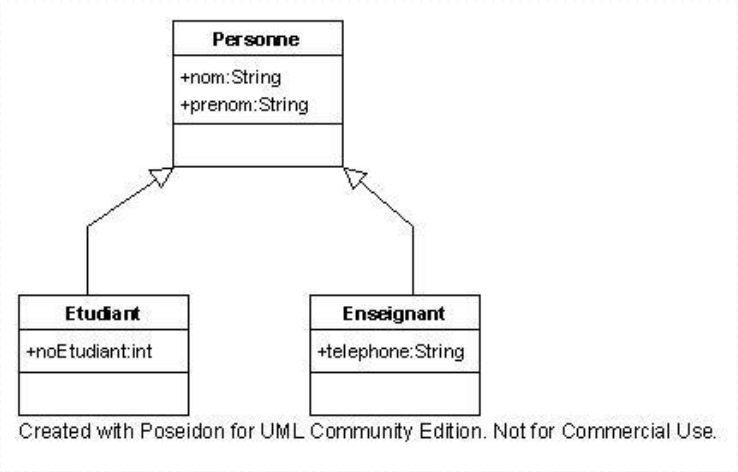
VI.I. Description

- Représente les objets et leurs liens à un instant donné.
- Mais aussi l'état des objets, ce qui permet d'exprimer des contextes d'exécution.
- En ce sens, ce diagramme est moins général que le diagramme de classes.

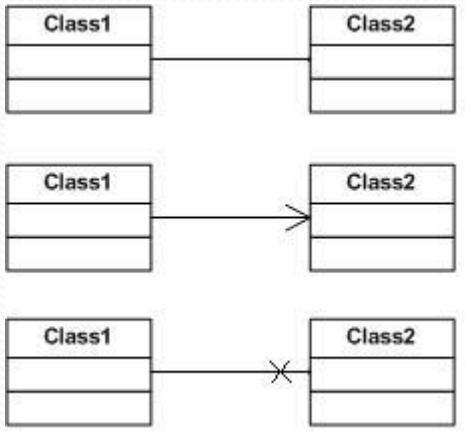
VI. Diagramme d'objet

VI.II. Eléments de base

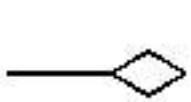
Héritage



Association



Agrégation

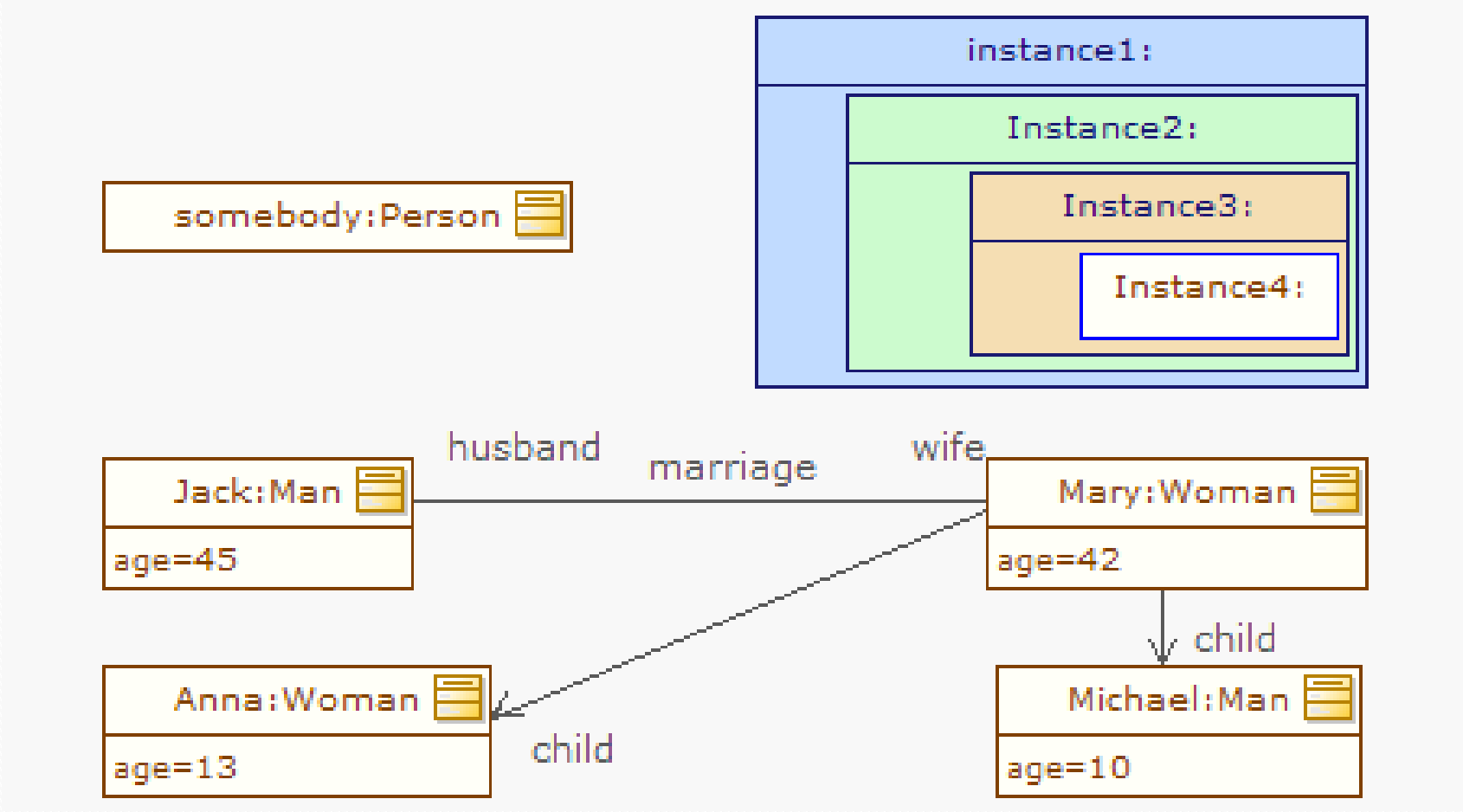


Dépendance

Composition

VI. Diagramme d'objet

VI.III. Exemple



VI. Diagramme d'objet

VI.IV. Elaboration

- Analyser le besoin métier
- Identifier les classes
  - Nom
  - Attributs & Visibilités
  - Spécialisation
  - Généralisation
- Identifier les liens entre les classes
  - Associations
  - Dépendances

VII. Diagramme de composant

- VII.I. Description
- VII.II. Eléments de base
- VII.III. Exemple
- VII.IV. Elaboration

VII. Diagramme de composant

VII.I. Description

- Représente l'organisation du système du point de vue des éléments logiciels
  - les modules (paquetages, fichiers sources, bibliothèques, exécutables)
  - les données (fichiers, bases de données)
  - Les éléments de configuration (paramètres, scripts, fichiers de commandes).

- Ce diagramme permet de mettre en évidence les dépendances entre les composants (qui utilise quoi).

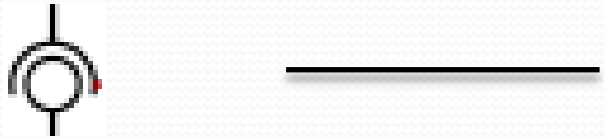
VII. Diagramme de composant

VII.II. Eléments de base

Système / Composant

Interface Fournis / Requis

Connecteurs

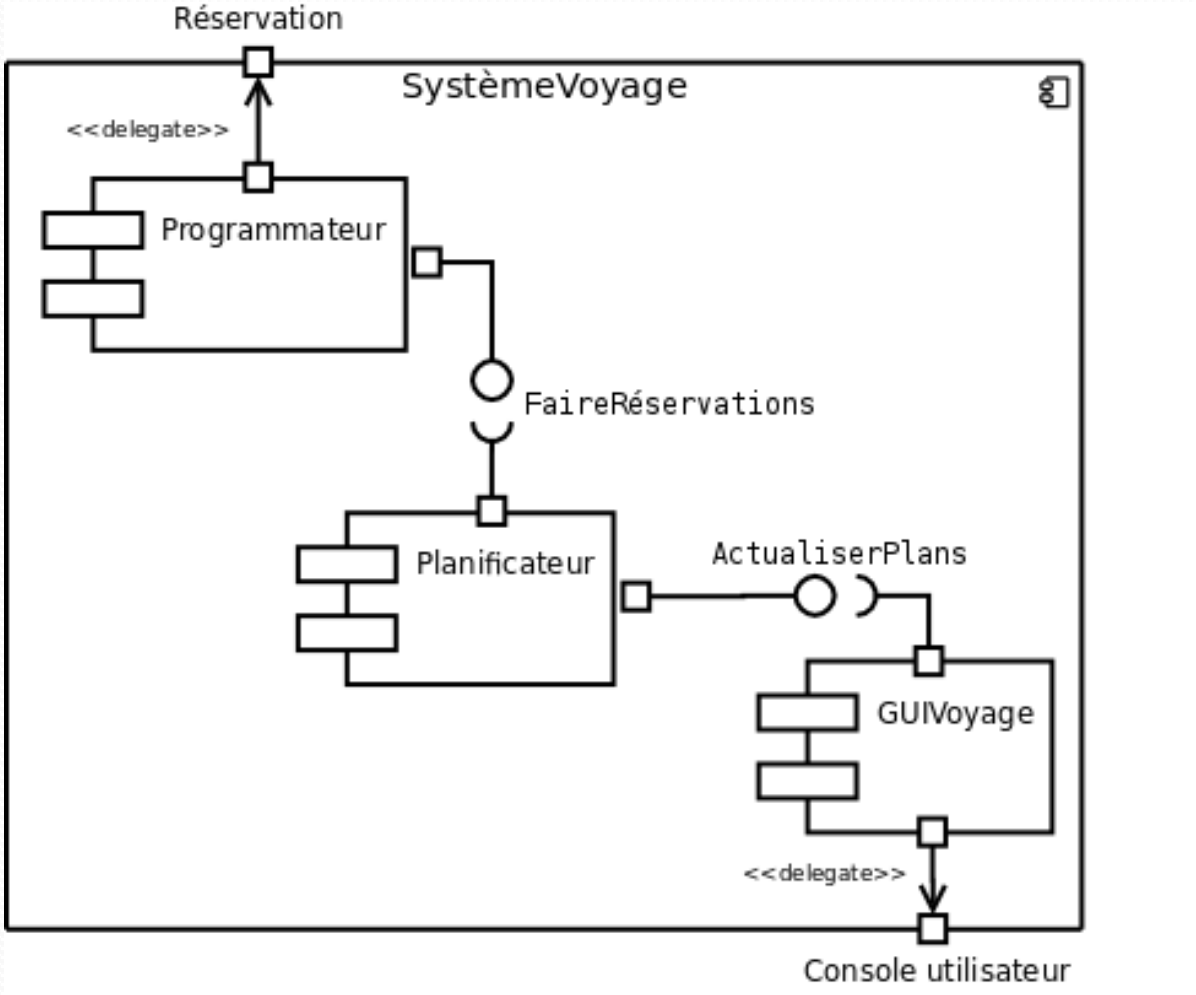


Port

Dépendance

VII. Diagramme de composant

VII.III. Exemple



VII. Diagramme de composant

VII.IV. Elaboration

- Analyser le besoin métier
- Identifier les systèmes et composants
  - Ports
  - Interfaces
  - Connecteurs
- Identifier les liens entre les composants
  - Dépendances
  - Appels des interfaces externes

VIII. Diagramme de déploiement

- VIII.I. Description
- VIII.II. Eléments de base
- VIII.III. Exemple
- VIII.IV. Elaboration

VIII. Diagramme de déploiement

VIII.I. Description

- Vue statique qui sert à représenter l'utilisation de l'infrastructure physique par le système et la manière dont les composants du système sont répartis ainsi que leurs relations entre eux.

- Les caractéristiques des ressources matérielles physiques et des supports de communication peuvent être précisées par stéréotype.
  - Ordinateurs
  - Serveurs
  - Périphériques
  - Etc.

VIII. Diagramme de déploiement

VIII.II. Eléments de base

- Sous-système
- Composant
- Connecteurs d'assemblage
- Connecteurs de délégation
  - Entrant
  - Sortant
- Port
- Interfaces
  - Fournis
  - Requises
- Dépendance

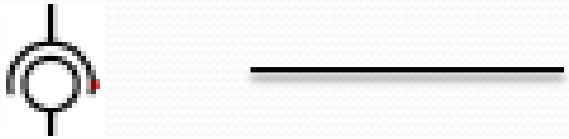
VIII. Diagramme de déploiement

VIII.II. Eléments de base

Sous-système / Composant

Interface Fournis / Requis

Connecteurs

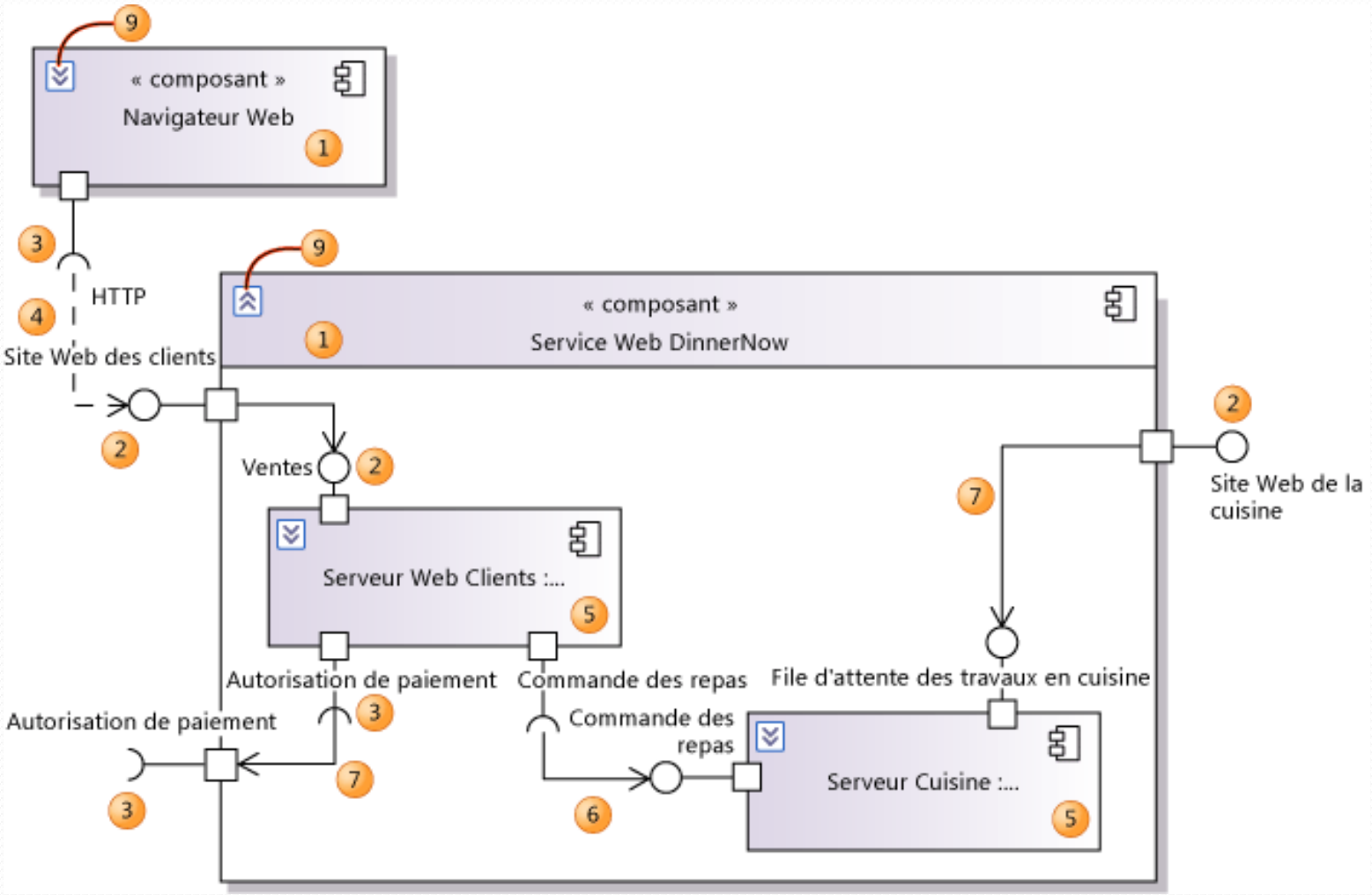


Port

Dépendance

VIII. Diagramme de déploiement

VIII.III. Exemple



VIII. Diagramme de déploiement

VIII.IV. Elaboration

- Analyser le besoin métier
- Identifier les systèmes et composants
  - Ports
  - Interfaces
  - Connecteurs
- Identifier les liens entre les composants
  - Dépendances
  - Appels des interfaces externes

IX. Diagramme de paquet

- IX.I. Description
- IX.II. Eléments de base
- IX.III. Exemple
- IX.IV. Elaboration

IX. Diagramme de paquet

IX.I. Description

- Représente des conteneurs logiques regroupant des éléments

- Les relations existant entre les paquetages (ou espaces de noms) composant un système

IX. Diagramme de paquet

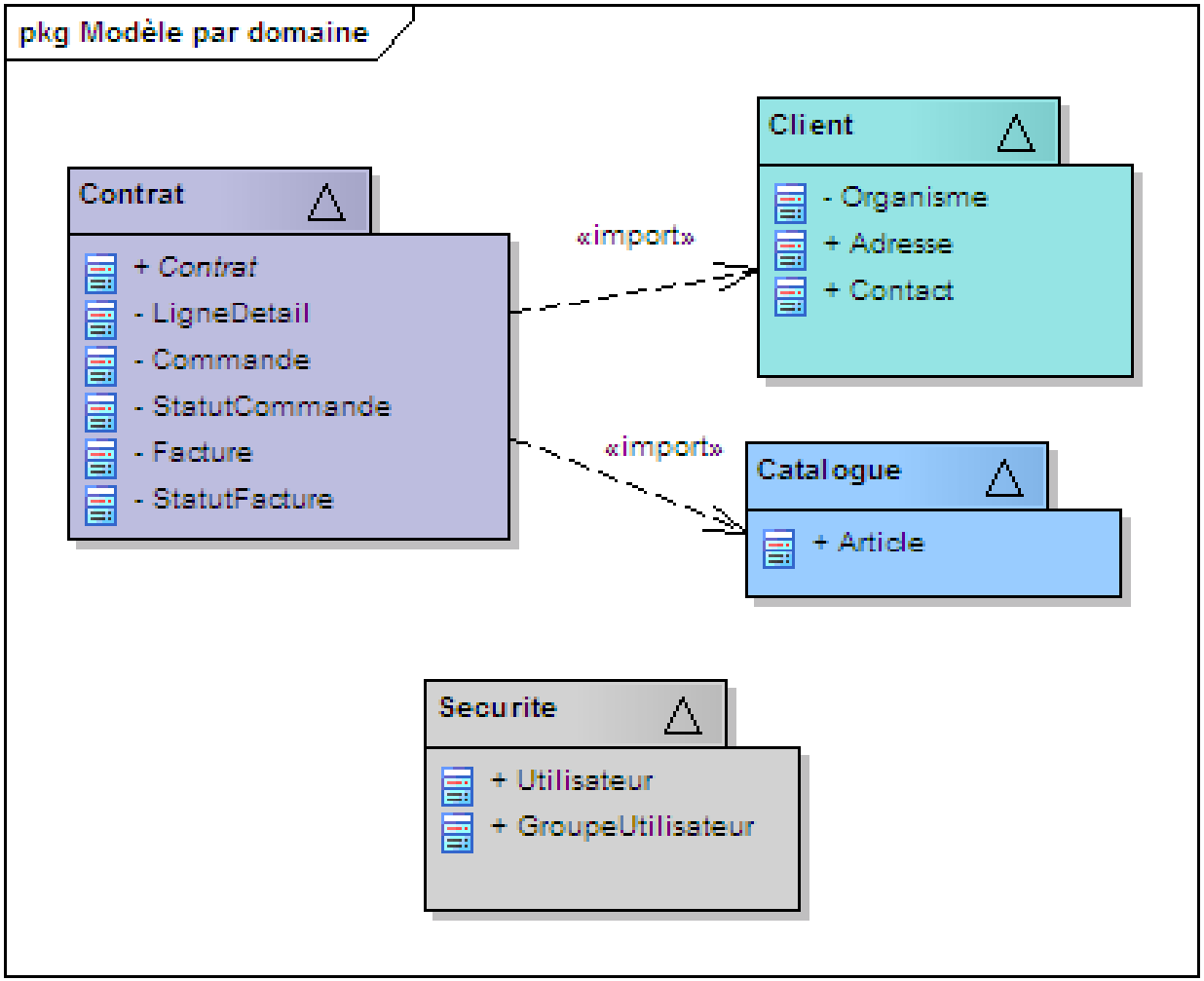
IX.II. Eléments de base

Paquet / Classe

Dépendance

IX. Diagramme de paquet

IX.III. Exemple



IX. Diagramme de paquet

IX.IV. Elaboration

- Analyser le besoin métier
- Identifier les paquets
  - Opérations et Visibilités
- Identifier les liens entre les paquets
  - Dépendances

X. Diagramme de structure composite

- X.I. Description
- X.II. Eléments de base
- X.III. Exemple
- X.IV. Elaboration

X. Diagramme de structure composite

X.I. Description

- Expose la structure interne d'une classe ainsi que les *collaborations* que cette dernière rend possible.

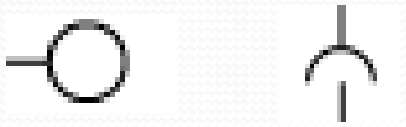
- Les éléments de ce diagramme sont les *parties* (en anglais *parts*), les *ports* par le biais desquels les parties interagissent entre elles, avec différentes instances de la classe ou encore avec le monde extérieur, et enfin les *connecteurs* reliant les parties et les ports.

X. Diagramme de structure composite

X.II. Eléments de base

Classe

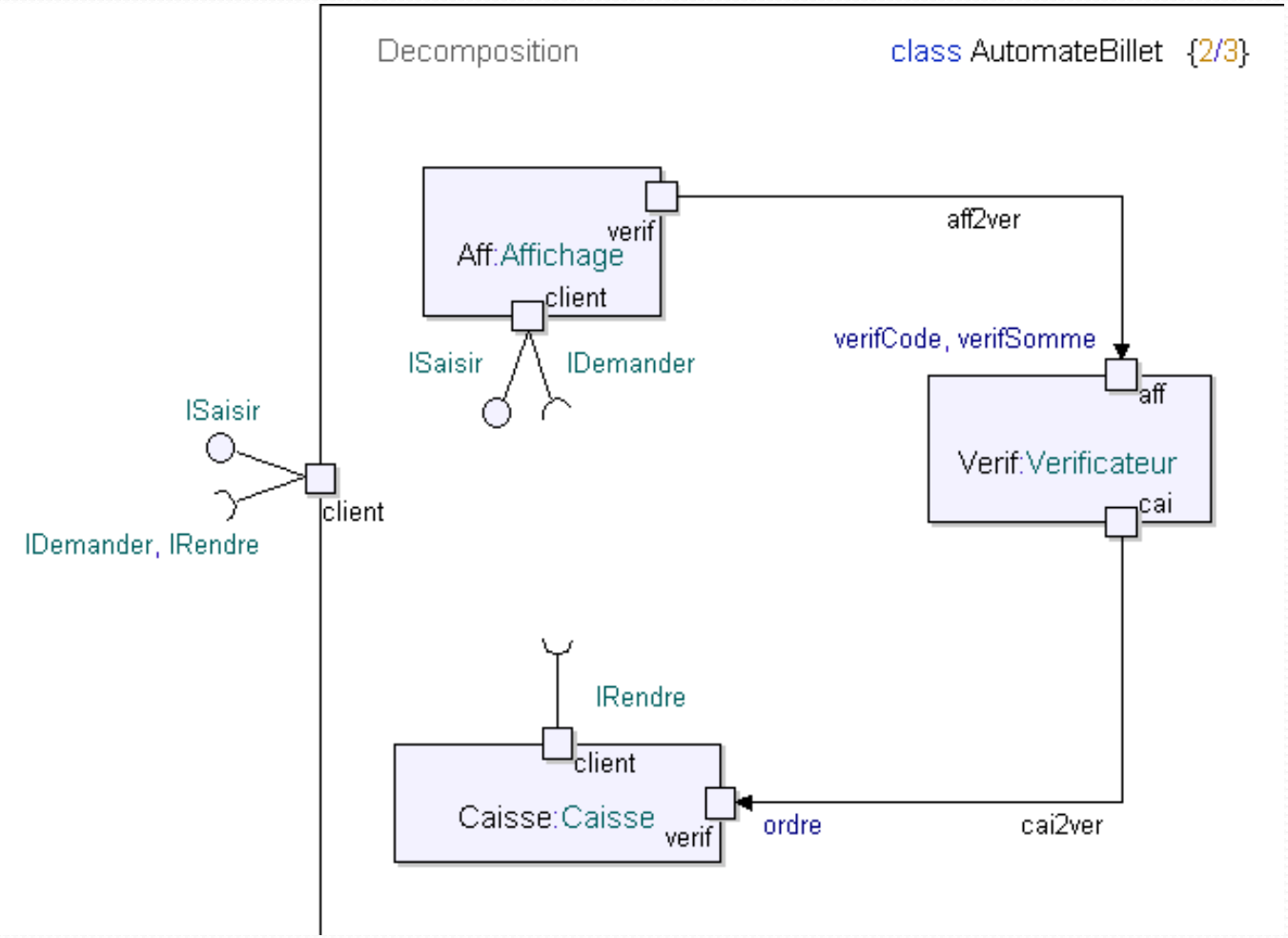
Interface Fournis / Requis



Port

X. Diagramme de structure composite

X.III. Exemple



X. Diagramme de structure composite

X.IV. Elaboration

- Analyser le besoin métier
- Identifier les composants et les sous-composants (modules)
  - Ports
  - Interfaces
  - Connecteurs
- Identifier les liens entre les sous-composants
  - Dépendances
  - Appels des interfaces externes
- Identifier les interfaces externes
  - Ports
  - Interfaces

XI. Diagramme de profil

- XI.I. Description
- XI.II. Eléments de base
- XI.III. Exemple
- XI.IV. Elaboration

XI. Diagramme de profil

XI.I. Description

- Vue statique qui permet de décrire un mécanisme d'extension léger par rapport à UML pour répondre à un domaine particulier par exemple Java ou #Net.
  - Par exemple en Java l'héritage multiple n'existe pas donc dans notre diagramme de profil nous allons obliger cette restriction.

XI. Diagramme de profil

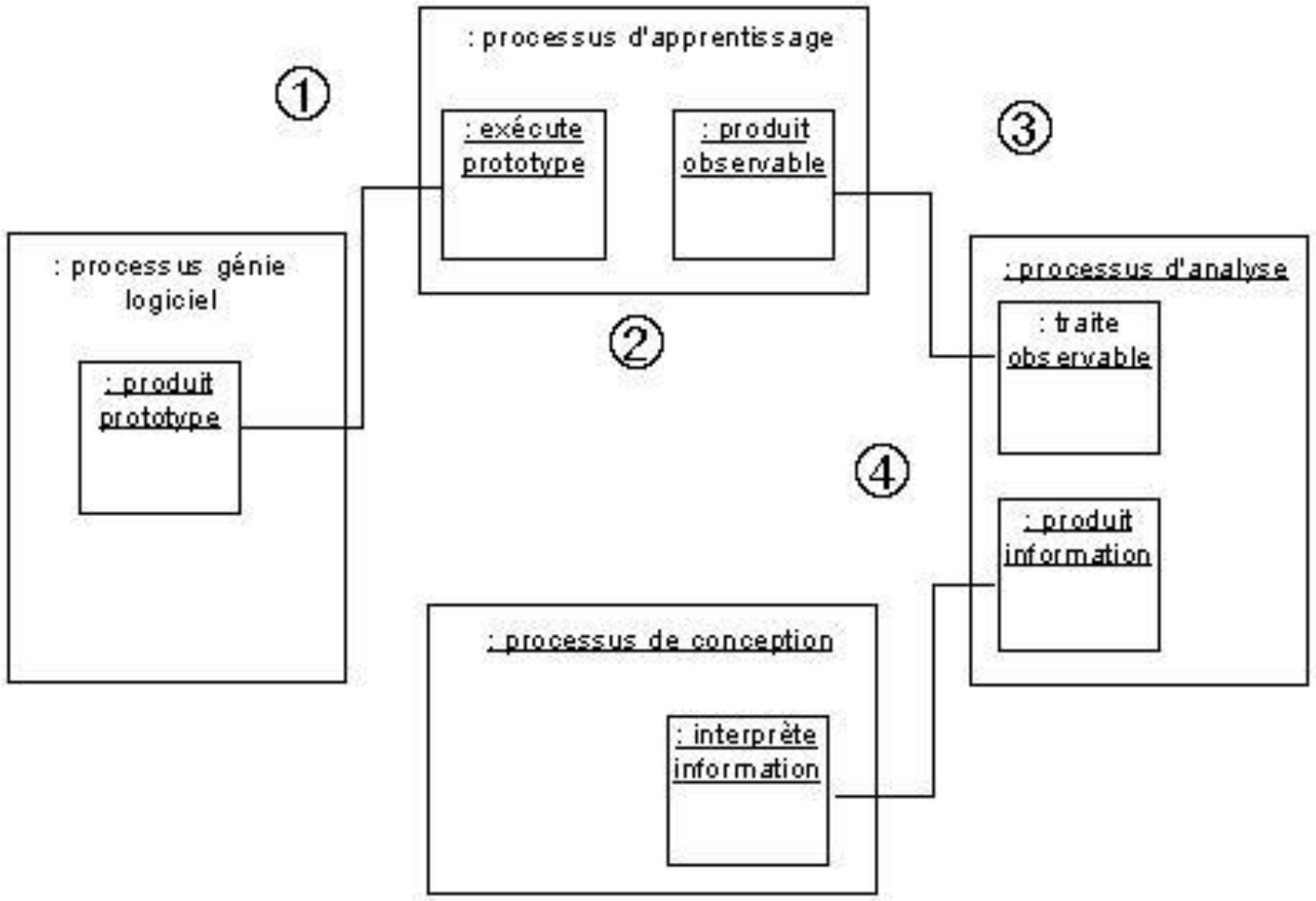
XI.II. Eléments de base

Nouveau Concept

Association

XI. Diagramme de profil

XI.III. Exemple



XI. Diagramme de profil

XI.IV. Elaboration

- Analyser le besoin métier
- Identifier les nouveaux concepts
  - Nom
  - Contraintes
- Identifier les liens entre les nouveaux concepts
  - Association

XII. Diagramme de cas d'utilisation

- XII.I. Description
- XII.II. Eléments de base
- XII.III. Exemple globale
- XII.IV. Exemple par sous système
- XII. V. Elaboration
- XII.VI. Description textuelle

XII. Diagramme de cas d'utilisation

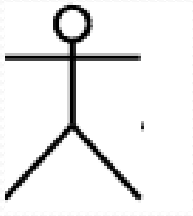
XII.I. Description

- Donne une vision globale du comportement fonctionnel d'un système logiciel.
- Ils sont utiles pour des présentations auprès de la direction ou des acteurs d'un projet,
- Un cas d'utilisation représente une unité discrète d'interaction entre un utilisateur (humain ou machine) et un système. Il est une unité significative de travail.
- Dans un diagramme de cas d'utilisation, les utilisateurs sont appelés acteurs (actors), ils interagissent avec les cas d'utilisation (use cases).

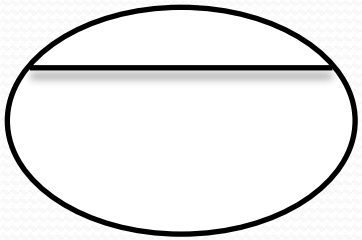
XII. Diagramme de cas d'utilisation

XII.II. Eléments de base

Acteur

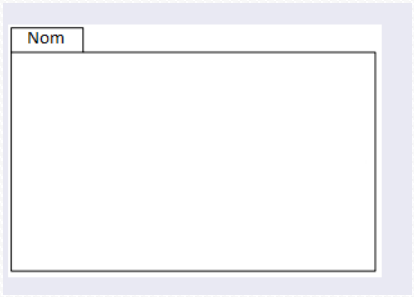


Opération



Association

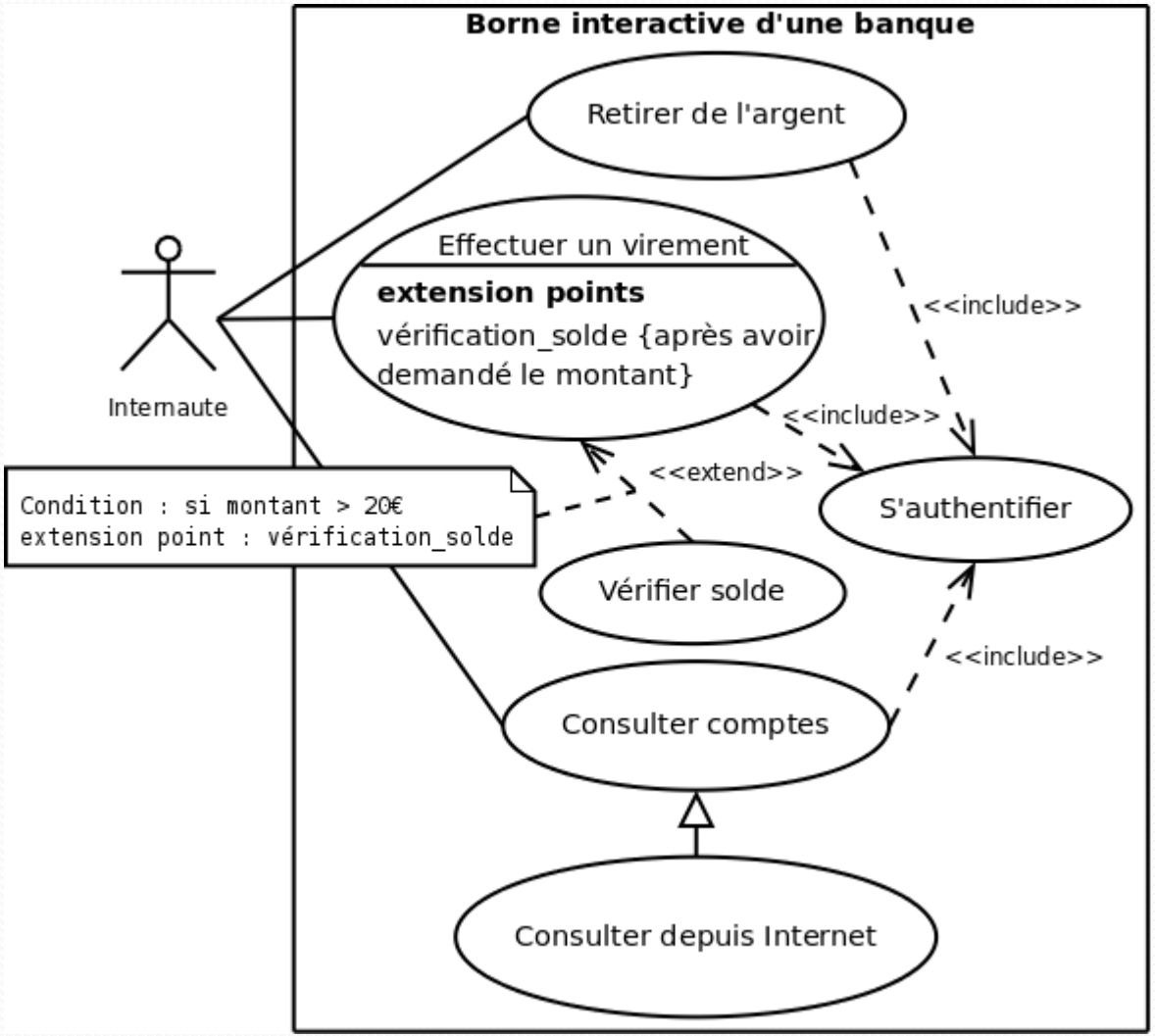
Dépendance



Paquetage

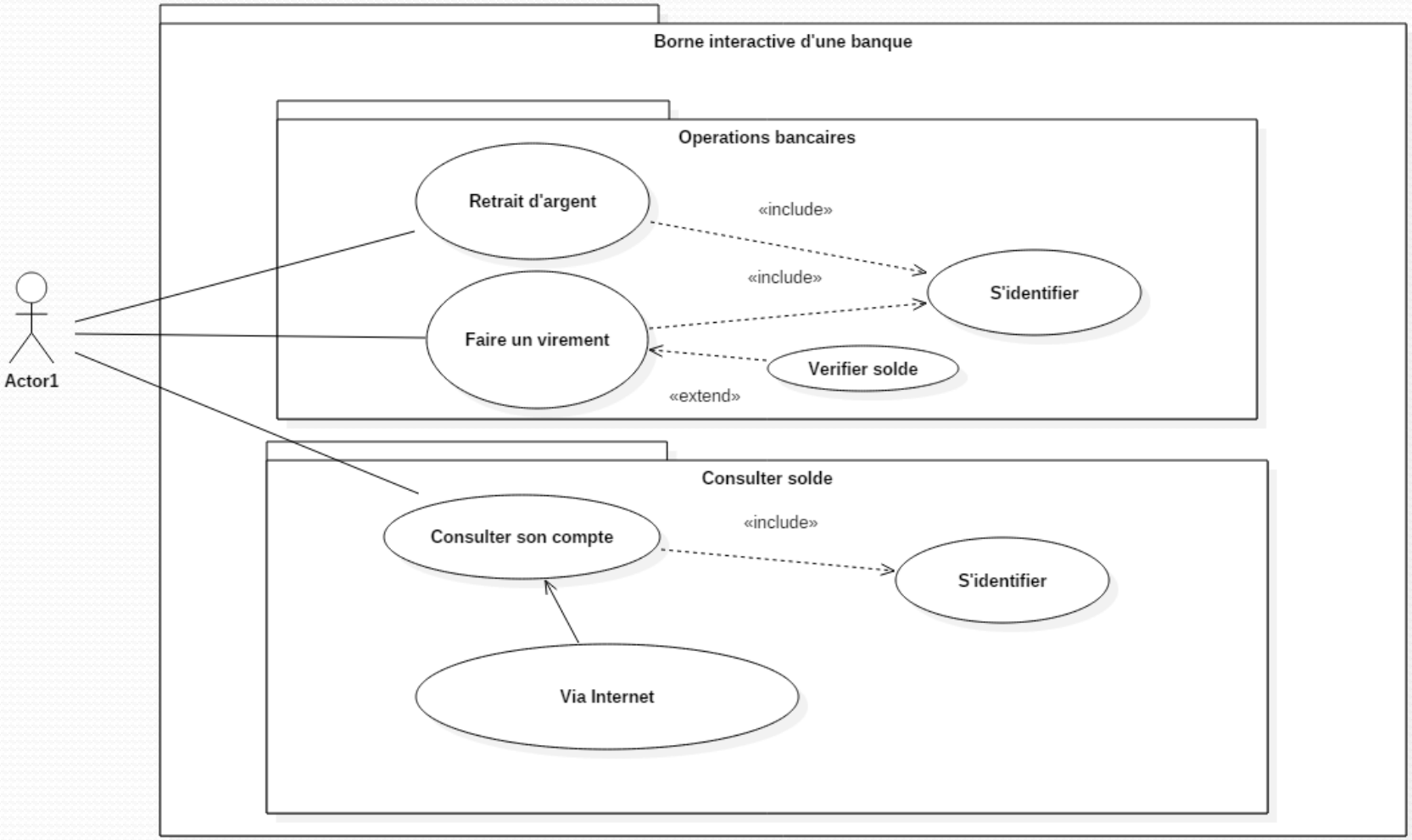
XII. Diagramme de cas d'utilisation

XII.III. Exemple globale



XII. Diagramme de cas d'utilisation

XII.IV. Exemple par sous système



XII. Diagramme de cas d'utilisation

XII.V. Elaboration

- Analyser le besoin métier
- Identifier l'acteur
  - Nom
  - Rôle
- Identifier les opérations
  - Nom
  - Description
- Identifier les liens entre les opérations
  - Dépendances
  - Contraintes
- Définir le cadre
  - Pré-conditions
  - Post-conditions

XII. Diagramme de cas d'utilisation

XII.VI. Description textuelle (1)

Cas d'utilisation : Nom du cas d'utilisation

- **Acteur** : rôle concerné
- **Parties prenantes et intérêts :**
  - Pour chaque rôle : définir son objectif
- **Portée** : Nom du système ou sous-système concerné
- **Pré-conditions** : Les conditions nécessaires pour la réalisation du cas d'utilisation
- **Post-conditions** : Le résultat attendu après la réalisation du cas d'utilisation
- **Scénario nominal**: La description du scénario nominal étape par étape
- **Extensions** : Les cas non nominaux et les cas de sortie
- **Contraintes** : Les contraintes imposées

XII. Diagramme de cas d'utilisation

XII.VI. Description textuelle (2)

- Il peut y avoir d'autres propriétés selon les règles de votre organisation
  - **Description** : description courte du cas d'utilisation
  - **Version** : version de l'application
  - **Dates** : dates de création et de mise à jour du cas d'utilisation
  - Etc.

- Il peut y avoir les mêmes propriétés avec des noms différents
  - **Extension < == > Scénarii non nominaux**
  - Etc.

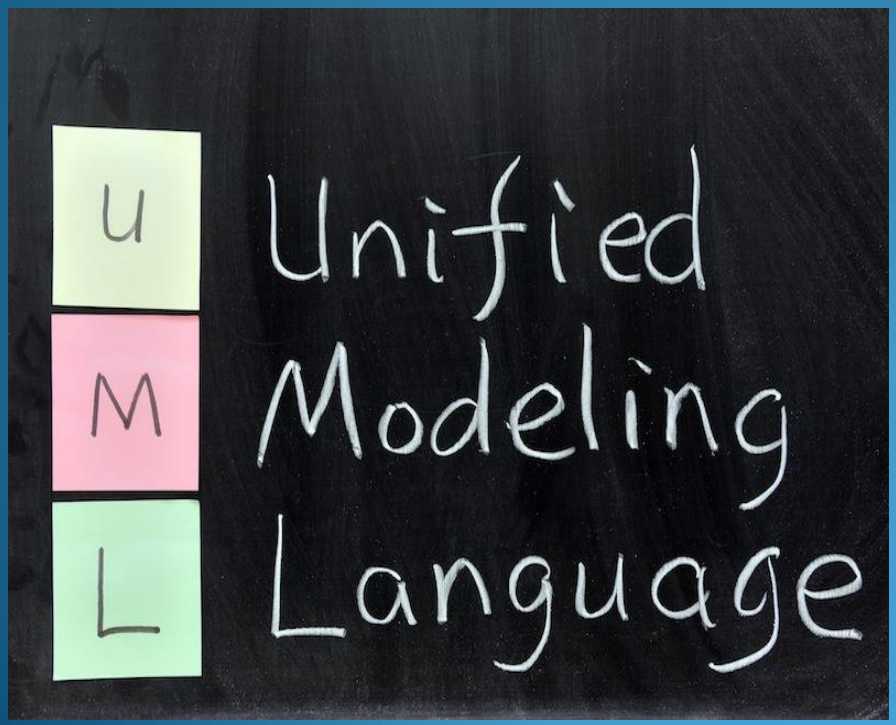
XII. Diagramme de cas d'utilisation

XII.VI. Description textuelle (3)

Cas d'utilisation : Effectuer un retrait depuis une borne libre service

- **Acteur** : client
- **Parties prenantes et intérêts :**
  - Client : retirer de l'argent de son compte
- **Portée** : sous-système des opérations bancaires
- **Pré-conditions** : disposer d'une carte valide *et du code d'identification correspondant à cette dernière*
  - *Cela permet d'alléger la description et d'éviter des détails inutiles*
- **Post-conditions** : le client dispose du montant souhaité
- **Scénario nominal**:
  - Le client entre sa carte
  - Il choisit l'option retrait et le montant souhaité
  - Il saisit son code et confirme l'opération
- **Extensions** :
  - Vérifier que le solde client
- **Contraintes** :
  - La carte doit être bloquée si 3 échecs de saisie du code
  - Le client ne peut pas retirer plus de 200€ / jour

Conception et UML
Travaux Pratiques



Naby Daouda Diakite

XIII. Diagramme d'état-transition

- XIII.I. Description
- XIII.II. Eléments de base
- XIII.III. Exemple
- XIII.IV. Elaboration

XIII. Diagramme d’état-transition

XIII.I. Description

- Représentation du comportement d'un objet sous la forme d'un automate à états
- Ne permet pas de comprendre globalement le fonctionnement du système, mais est directement transposable en algorithme.
- Tous les automates d'un système s'exécutent parallèlement et peuvent donc changer d'état de façon indépendante.

XIII. Diagramme d'état-transition

XIII.II. Eléments de base

Début

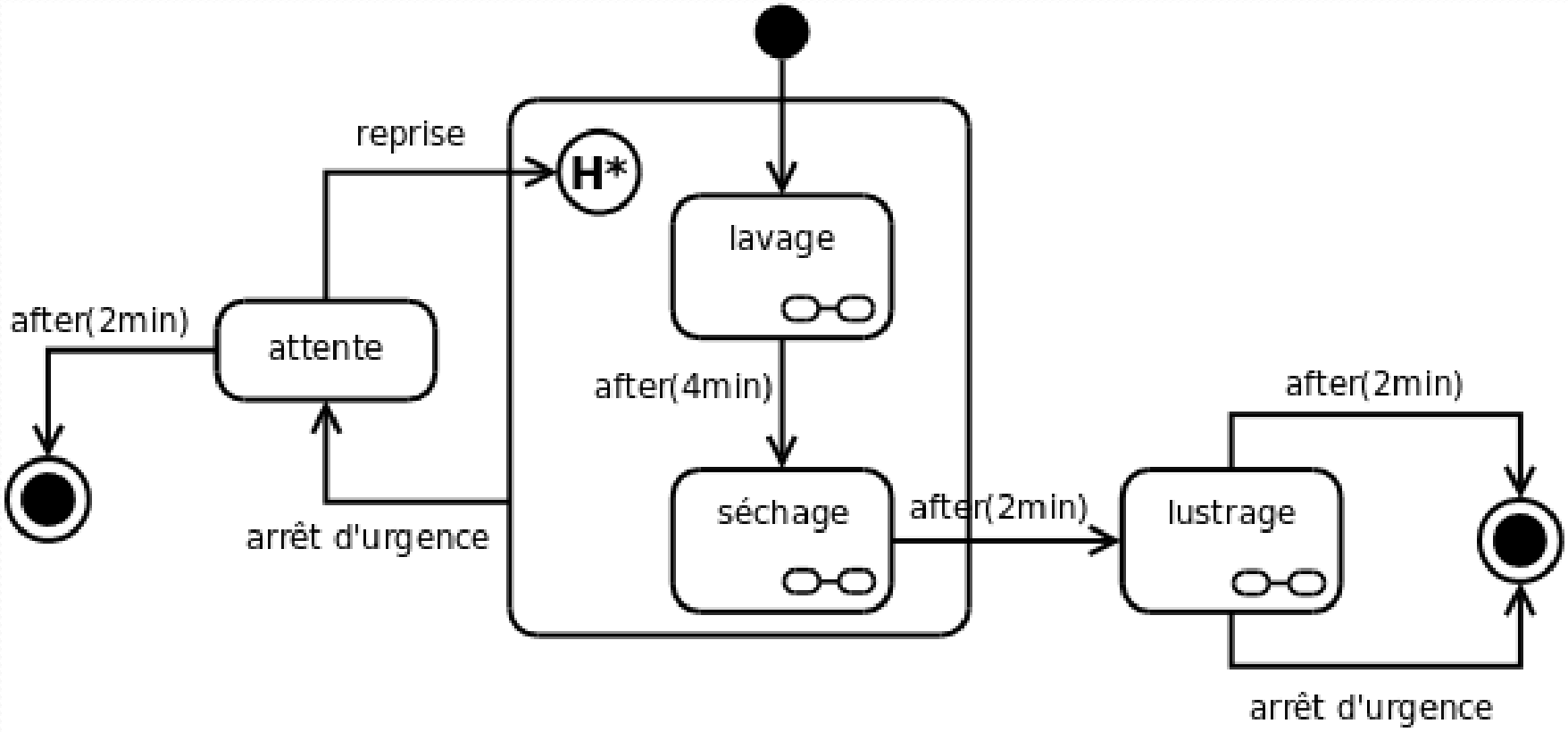
Fin

Etat

Association

XIII. Diagramme d'état-transition

XIII.III. Exemple



XIII. Diagramme d'état-transition

XIII.IV. Elaboration

- Analyser le besoin métier
- Identifier les points d'entrée et de sortie
- Identifier les états
  - Nom
- Identifier les liens entre les états
  - Associations et Descriptions
  - Conditions