Base de la Conception Orientée Objet

Iza Marfisi
iza.marfisi@univ-lemans.fr
Bureau au fond à gauche

Cours DUT 2ème année

Organisation

- 5CM de 1h
  - Concepts théoriques
  - Définitions des diagrammes
- 5TD de 1h30
  - Exercices papier sur les diagrammes
- 5TDm de 3h
  - Logiciel Modelio (gratuit)
  - Logiciel Omondo pour reverse engineering avec Java
- 3TP de 3h
  - Projet en corrélation avec le cours de Programmation Orienté Objet
- Évaluation
  - 1 TDm noté (coeff 1/6)
  - 1 projet noté (coeff 1/6)
  - 1 examen final (coeff 2/3) aucun document autorisé sauf 1 feuille taille A4 recto verso avec vos notes personnelles

Plan du cours

1. Logique de la programmation orientée objet
2. Un besoin de modélisation
3. Modéliser la structure avec UML
4. Modéliser le comportement avec UML
5. Modéliser les fonctionnalités avec UML
6. Développer avec UML

Références

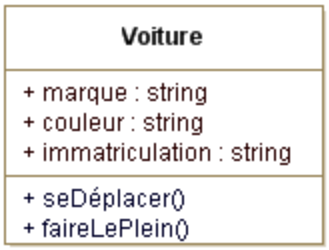
- Cours :
  - Pierre Laforcade – OMGL et UML avancés – IUT de Laval
  - Christine Solnon – Modélisation UML – INSA de Lyon
  - Eric Cariou – UML 2 – Université de Pau et des Pays de l'Adour
  - Shebli Anvar – Introduction à UML - CEA Saclay
  - Tarak Chaar – Atelier UML – l'institut supérieur d'électronique et de communication de Sfax
- Livres :
  - Xavier Blanc et Isabelle Mounier (2006), *UML 2 pour les développeurs*, Eyrolles, Paris, 202 p.

1. Logique de la programmation orientée objet

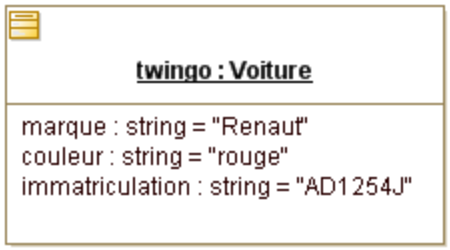
Vision objet (1/3)

- Système d'information = un ensemble d'objets qui collabore entre eux
  - Facilite la réutilisation et la maintenance du système

- Un objet = une instance du type abstrait « classe » et une classe est caractérisé par :
  - Des frontières précises
  - Une identité
  - Un ensemble d'attributs (propriétés) décrivant son état
  - Un ensemble d'opérations (méthode, fonction) définissant son comportement



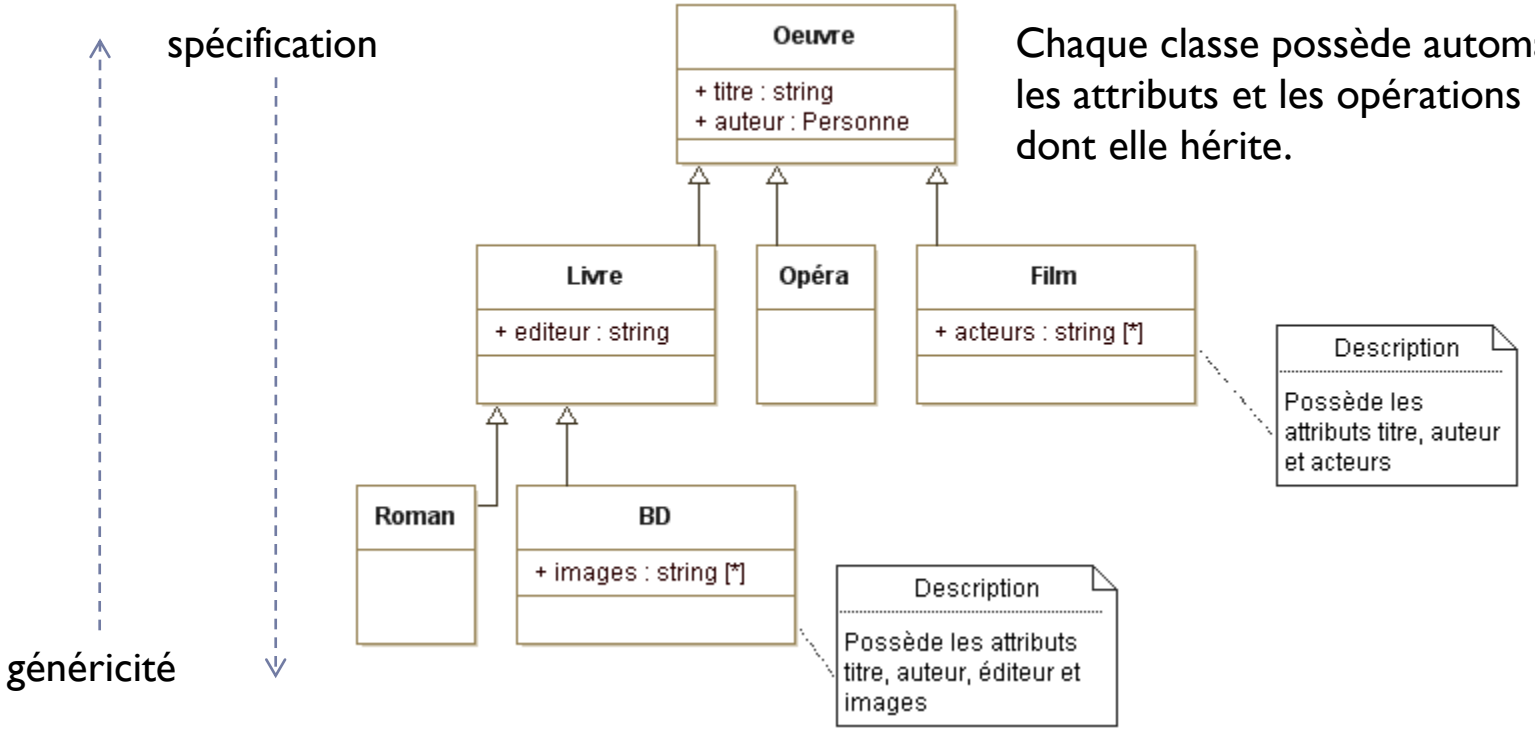
classe = regroupement de données et de traitement



objet = instance d'une classe

Vision objet (2/3)

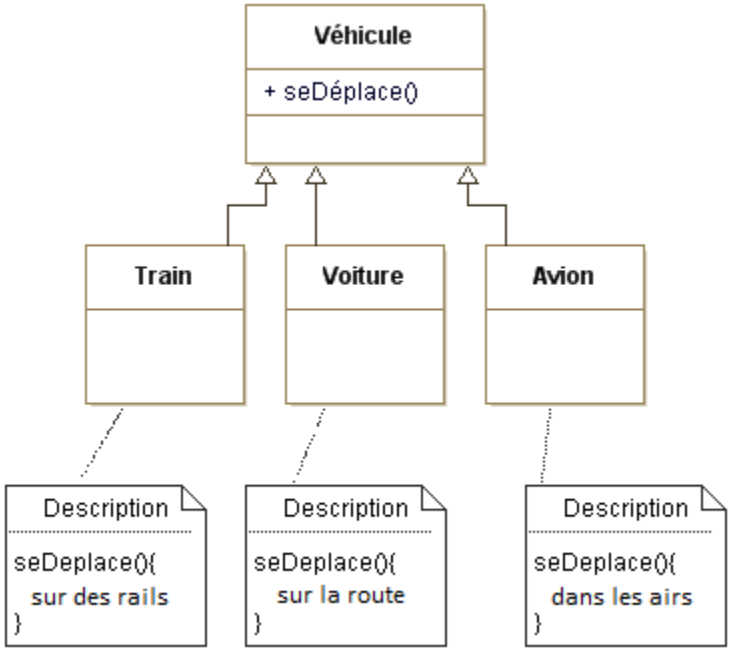
- Héritage : transmission des propriétés (attributs et opérations) d'une classe à une autre classe
  - évite la duplication de code
  - encourage la réutilisation



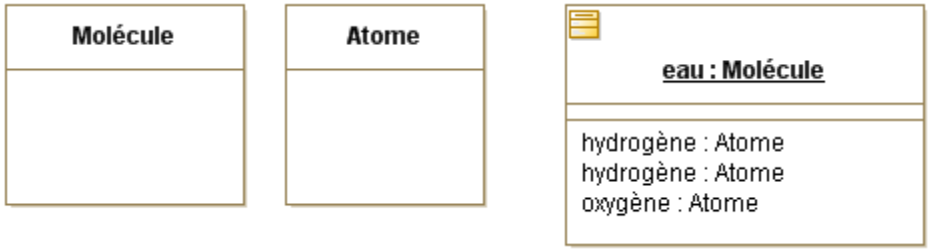
Chaque classe possède automatiquement les attributs et les opérations des classes dont elle hérite.

Vision objet (3/3)

- Polymorphisme : factorisation de comportement (opérations) commun d'objet
  - regroupe les comportements communs
  - garde les variantes de comportement spécifique à chaque objet



- Agrégation : un objet peut être composé de plusieurs objets
  - composition d'objet complexe avec des objets simples



Exercice 1

1. Avec une logique orientée objet, listez les principaux objets qui constituent le cours dans lequel vous vous trouvez.
2. Donnez quelques uns de leur attributs et opérations.
3. Trouvez des relations d'agrégation entre ces objets.
4. Trouver des relations d'héritage entre ces objets.

Solution

Exercice 2

- Vous avez tous une feuille différente devant vous avec une description textuelles d'un système informatique. Vous avez 5 minutes pour :
  - Lire cette description
  - Retournez la feuille et, sans recopier le texte, représentez ce système avec tous les moyens que vous voulez (dessin, code, texte) afin qu'un puisse être compris par quelqu'un d'autre.
  - Donner votre feuille à la personne derrière vous
- Vous recevez une feuille
  - Essayez de comprendre le système à l'aide des explications dessinées
  - Retournez la feuille et comparez avec la version textuelle. Aviez-vous compris la même chose ?
- Gardez bien cette feuille ! Elle vous servira par la suite.

Exercice 3

- Pour le prochain TD, lisez le code Java qui vous a été distribué. Des questions vous seront posées pour vérifier votre degré de compréhension du système.

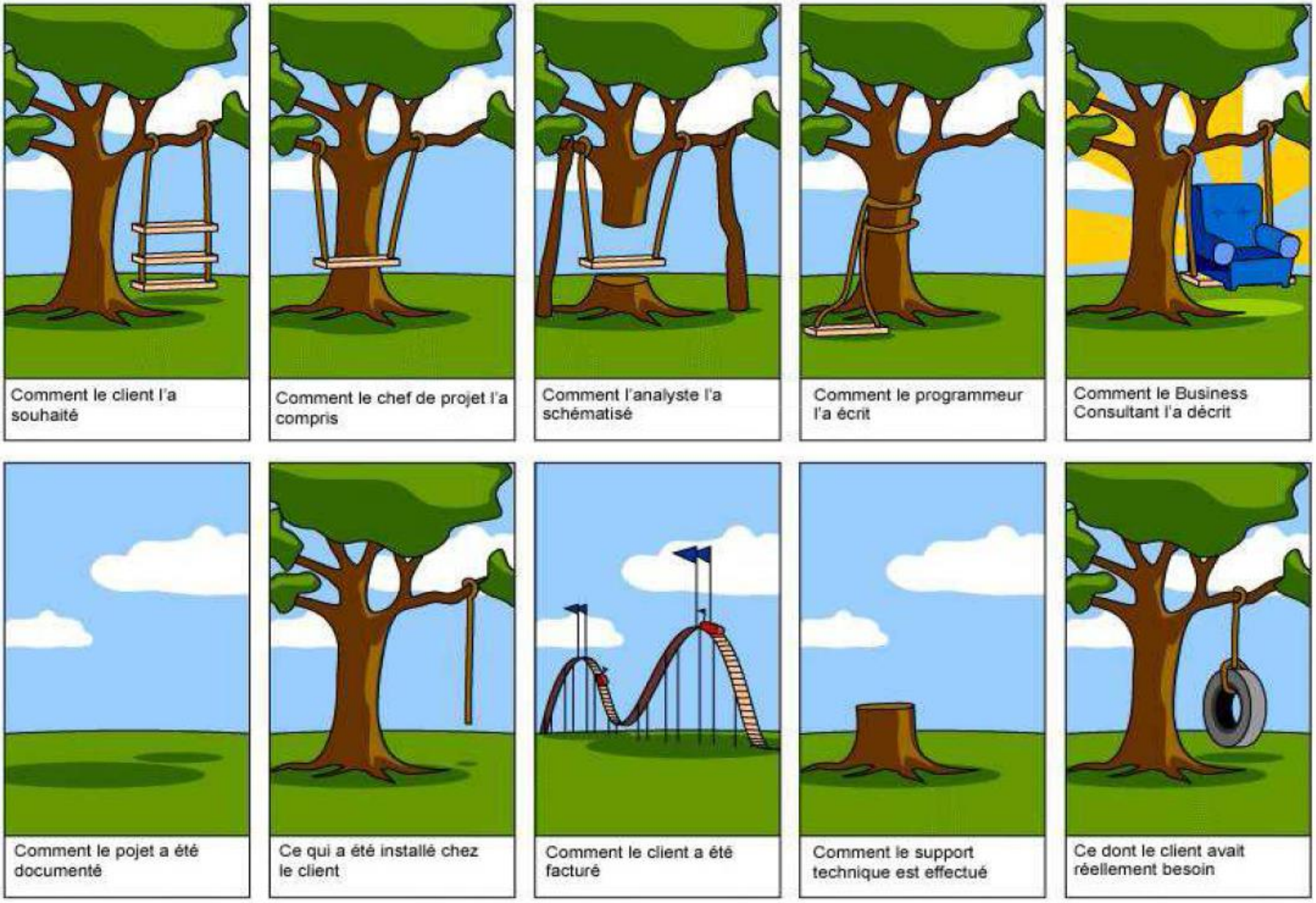
2. Un besoin de modélisation

Qu'est ce qu'un modèle ?

- Un modèle est une représentation partielle de la réalité
  - Abstraction de ce qui est intéressant pour un contexte donné
  - Vue subjective et simplifiée d'un système
  - Dans ce cours, on s'intéresse aux modèles d'applications informatiques

- Utilité des modèles
  - Faciliter la compréhension d'un système
  - Faciliter la communication entre développeurs, chef de projet et client
  - Définir et simuler le fonctionnement d'un système

Pourquoi un modèle ?



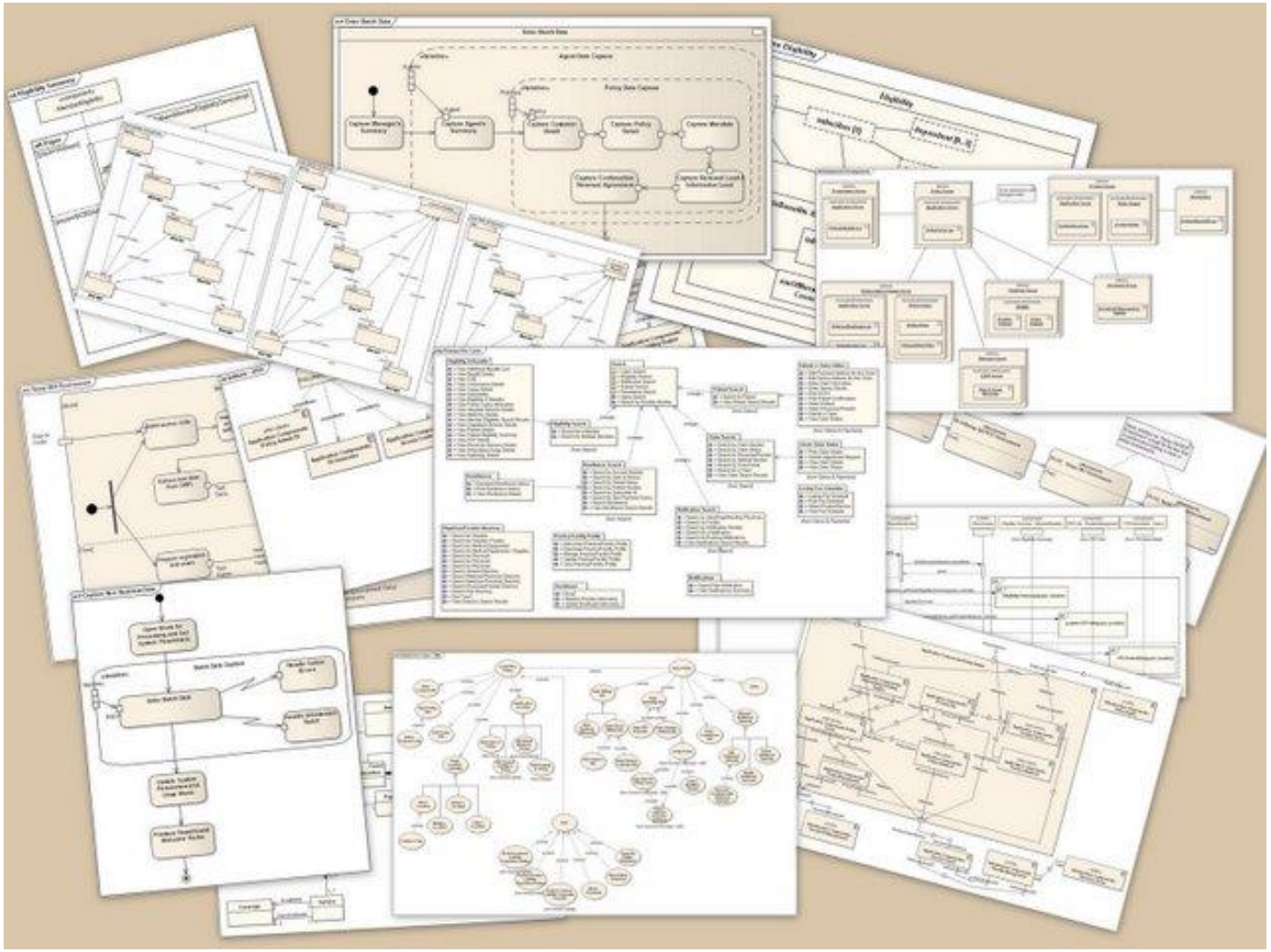
Historique d'UML

- Conçu par l'Object Management Group www.omg.org
  - Version 1.0 d'UML en 1997
  - Définition d'UML selon l'OMG : « Langage visuel dédié à la spécification, la construction et la documentation des artéfacts d'un système logiciel »
  - Améliorations continuelles
    - Nouvelles versions (version 2.5 en 2012)
    - Nouveaux outils pour faciliter modélisation et lien avec le code

Caractéristiques d'UML

- UML est un langage :
  - Vocabulaire spécifique
  - Règles de syntaxe
  - Notation spécifique
- UML cadre pour l'analyse d'objet en offrant :
  - Différentes vues (perspectives) complémentaires d'un système
  - Différents niveaux d'abstraction
  - La possibilité de modéliser le système complet avec sont environnement
- UML est un support de communication
  - Sa notation graphique permet d'exprimer visuellement une solution
  - L'aspect formel de sa notation limite les ambigüités et les incompréhensions
- UML est lisible par des humains et des machines

13 types de diagrammes



Dans ce cours, nous ne présentons que les 5 plus utilisés (en gras dans la prochaine diapo)

13 diagrammes pour 3 points de vue complémentaires :

- **Diagramme de classes (class diagram)**
- **Diagramme de paquetages (package diagram)**
- Diagramme de déploiement (deployment diagram)
- Diagramme d'objet (object diagram)
- Diagramme de structure composite (composite structure diagram)
- Diagramme de composants (component diagram)

Structure

- **Diagramme de séquence (sequence diagram)**
- **Diagramme d'états-transition (state diagram)**
- Diagramme de communication (communication diagram)
- Diagramme d'activité (activity diagram)
- Diagramme de vue générale d'interaction (interaction overview diagram)
- Diagramme de temps (timing diagram)

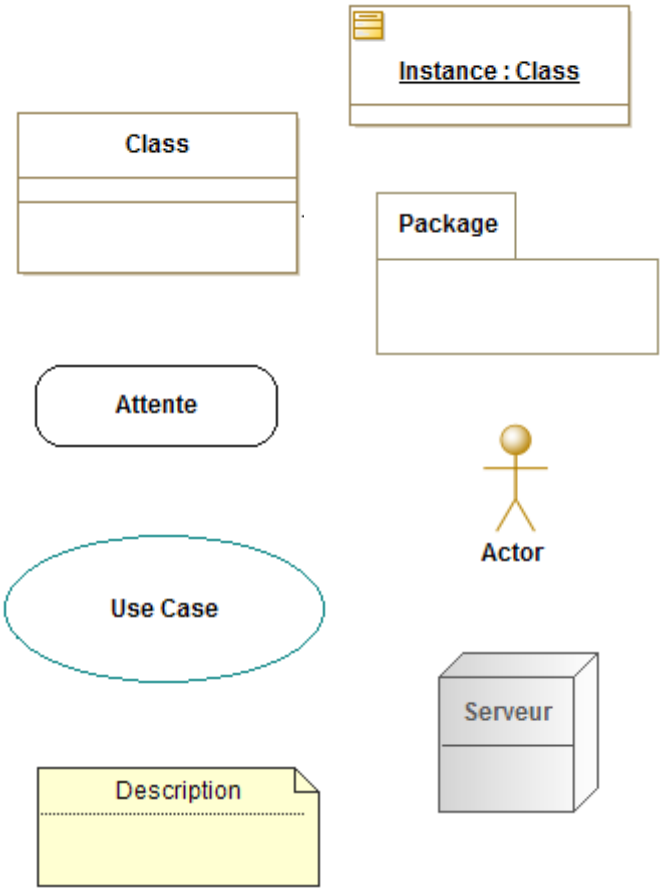
Comportement

- **Diagramme de cas d'utilisation (use case diagram)**

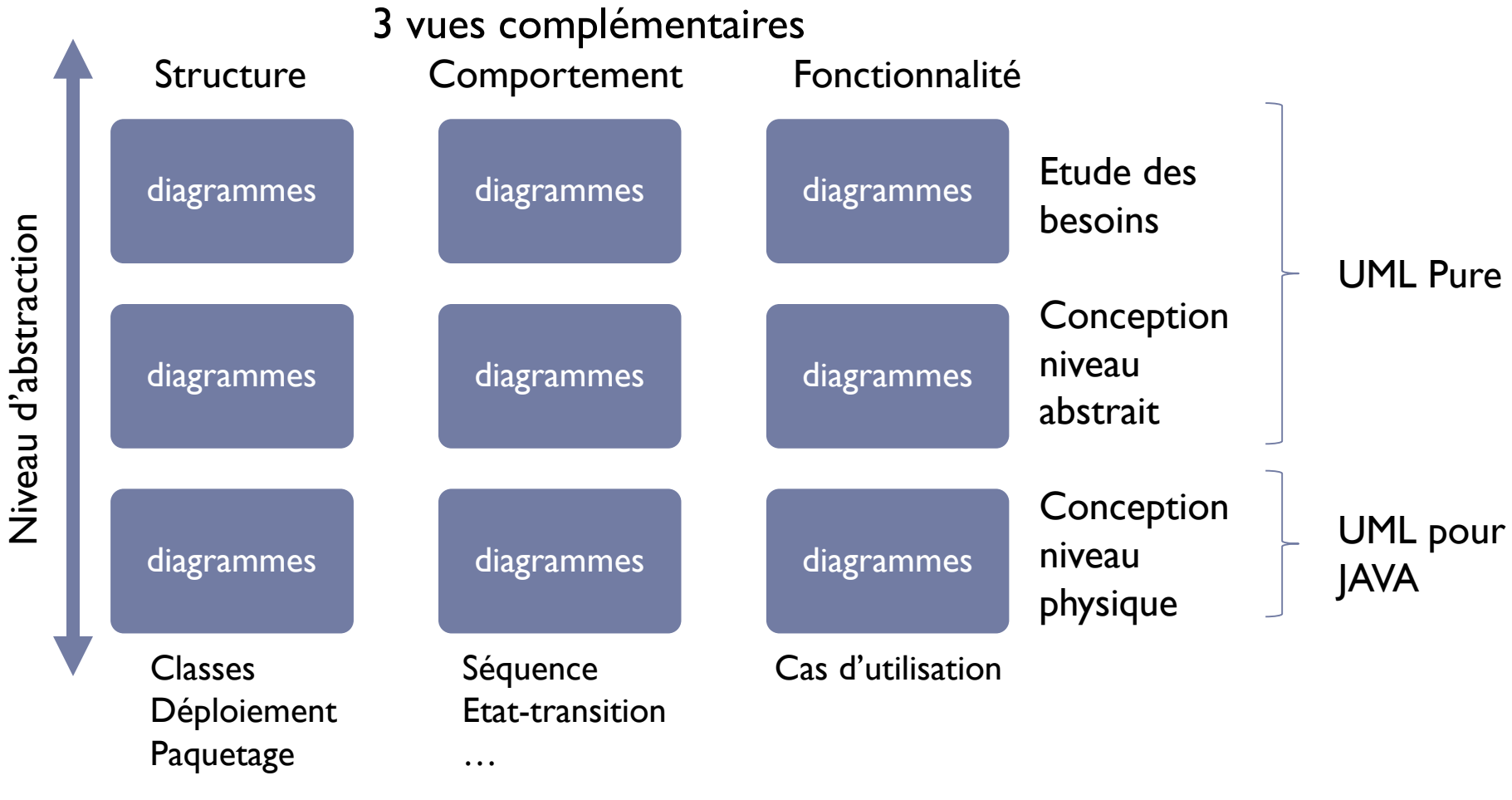
Fonctionnalité

Les éléments de modélisation

- Objet : entité réelle ou virtuelle (instance d'une classe)
- Classe : ensemble d'objets (regroupement de données et de traitement)
- Paquetage : ensemble de classe
- États : étape de la vie d'un objet
- Acteur : utilisateurs du système
- Cas d'utilisation : utilisation possible du système
- Nœud : dispositif matériel
- Note : commentaire, remarque ou explication
- Message et Relation : communication entre entités



Niveau d'abstraction de la modélisation



Comment modéliser avec UML ?

- **UML est un langage qui permet de représenter des modèles, mais il ne définit pas le processus d'élaboration des modèles !**

- Cependant, dans le cadre de la modélisation d'une application informatique, les auteurs d'UML préconisent d'utiliser une démarche :
  - itérative et incrémentale,
  - guidée par les besoins des utilisateurs du système
  - centré sur l'architecture logicielle

- D'après les auteurs d'UML, un processus de développement qui possède ces qualités devrait favoriser la réussite d'un projet.

Démarche itérative et incrémentale

- L'idée est simple : pour modéliser (comprendre et représenter) un système complexe, il vaut mieux s'y prendre en plusieurs fois, en affinant son analyse par étapes.

- Réaliser des diagrammes avec différent niveau d'abstraction :
  - Diagrammes abstraits pour l'étude des besoins
  - Diagrammes pour la conception niveau abstrait
  - Diagrammes pour la conception niveau physique

- Cette démarche devrait aussi s'appliquer au cycle de développement dans son ensemble, en favorisant le prototypage.

3. Modéliser la structure avec UML

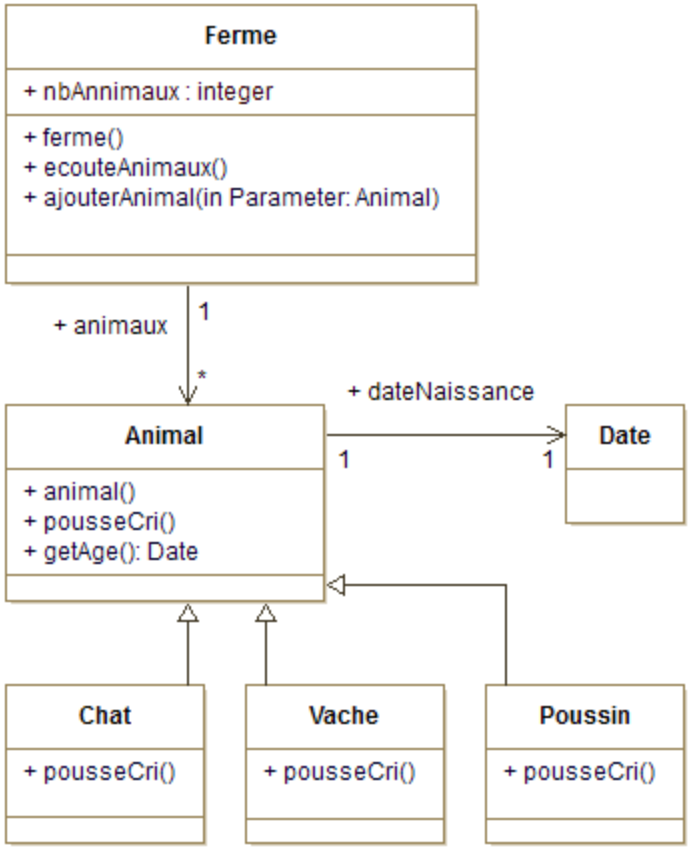
Diagramme de classes

Introduction au diagramme de classes

- Éléments représentés
  - Classes et Interfaces avec leurs attributs et opérations
  - Relations entre les classes (héritage, association)

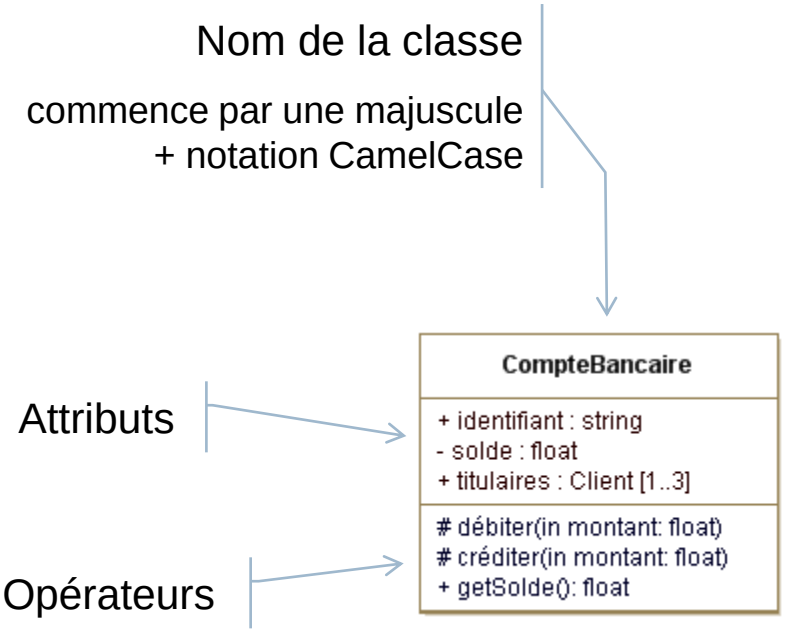
- Utilisation
  - Pendant l'étude des besoins (niveau 1)
    - modélisez le domaine
  - Pendant la conception niveau abstrait (niveau 2)
    - modéliser la structure du système (sans être reliés aux spécificités du langage ou de la plateforme de développement)
  - Pendant la conception niveau physique (niveau 3)
    - modéliser la structure du système en Java, tel qu'il est développé,
    - facilité la compréhension du système pour sa maintenance

Exemple


Ferme
+ nbAnnimaux : integer
+ ferme()
+ ecouteAnimaux()
+ ajouterAnimal(in Parameter: Animal)
+ animaux
1
*
+ dateNaissance
Animal
+ animal()
+ pousseCri()
+ getAge(): Date
1
1
Date
Chat
+ pousseCri()
Vache
+ pousseCri()
Poussin
+ pousseCri()

Class

- Classe : entité avec un ensemble d'attributs et d'opérations




```java
public class CompteBancaire{
  public String identifiant;
  private Float solde;
  public ArrayList titulaires;

  protected void debiter(Float montant) {}
  protected void credit(Float montant) {}
  public Float getbalance () {}
}
```

Attributs

- Format de description d'un attribut :

  [visibilité] Nom [Multiple] [:Type] [=ValDef]

  - **Visibilité** :
    - - Private - accessible uniquement depuis sa propre classe
    - # Protected - accessible depuis sa propre classe et des classes qui héritent de cette classe
    - + Public - accessible de partout
  - **Nom** commençant avec une minuscule + notation CamelCase
  - **Multiple** : [nbElément] ou [min..max]
  - **Type** : Types primitifs (integer, string, bollean, char ou float) ou Nom d'une autre classe
  - **ValDef** : valeur par défault
  - **Propriété** : {gelè}, {variable}, {ajoutUniquement}…
  - **Attribut statique :** description soulignée

Opérateurs

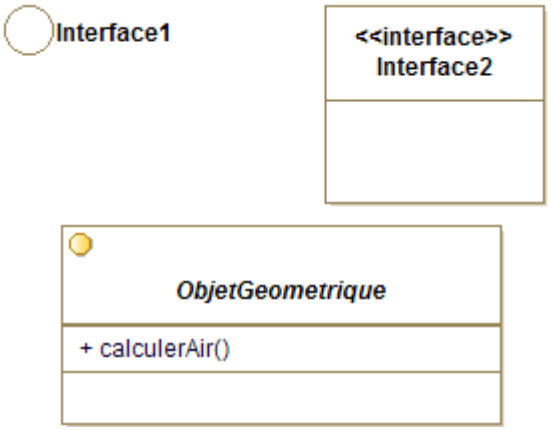
- Format de description d'un opérateur :
  [visibilité] Nom [(Arg)] [: TypeRetour]

  - **Visibilité** : - Private, # Protected ou + Public (comme les attributs)
  - **Nom** commençant avec une minuscule + notation CamelCase
  - **Arg** : liste des arguments selon le format [Dir] NomArgument : TypeArgument
    - Dir : in (par défault) out ou inout
  - **TypeRetour** : Types primitifs ou classe (comme les attributs)
  - **Opération statique** : description soulignée

Interface

- Interface : classe sans attribut dont toutes les opérations sont abstraites (sans le contenu du code)
  - Une classe qui « implémente » une Interface doit définir le code pour chacune des opérations héritées
  - Pourquoi les interfaces : en Java une classe ne peut hériter que d'une seule classes mais réaliser plusieurs interfaces

3 représentations UML possibles



Java

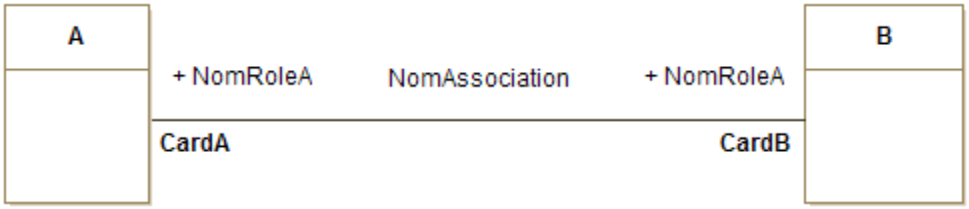
```
public interface ObjectGeometrique{
  public Point pts;
  public float air;
  public float calculerAir() {}
}
```

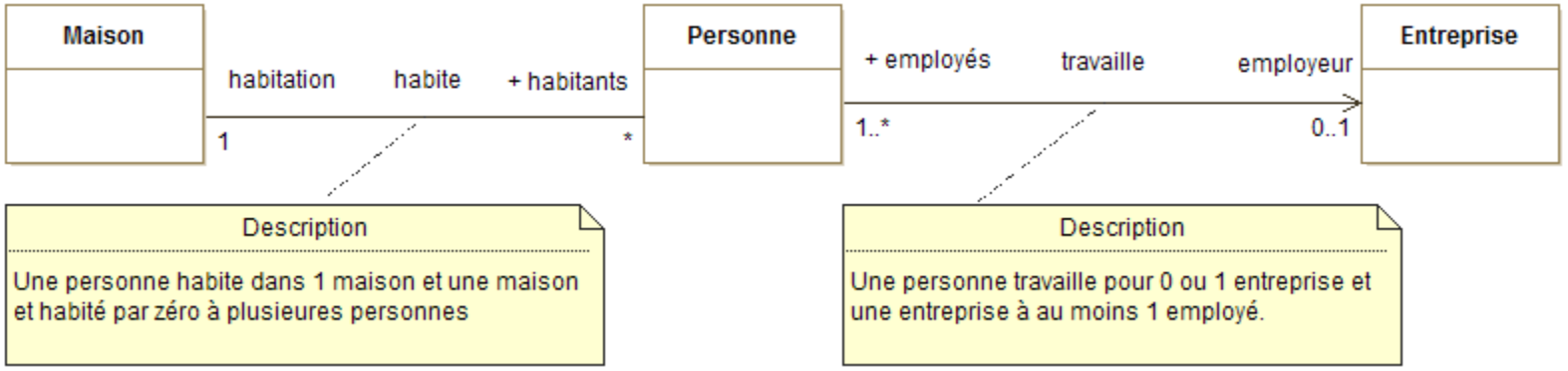
Relations entre classes et interfaces

- 6 types de relation:
  - Association
    - Agrégation 
    - Composition 
  - Héritage 
  - Implémentation 
  - Dépendance 
- Navigation
  - Représenté par une flèche sur le bout d'une relation
  - Signifie qu'une classe peut accéder aux attributs et opérations d'une autre classe
  - Si aucune flèche, la relation est navigable dans les deux sens

Association 1/3

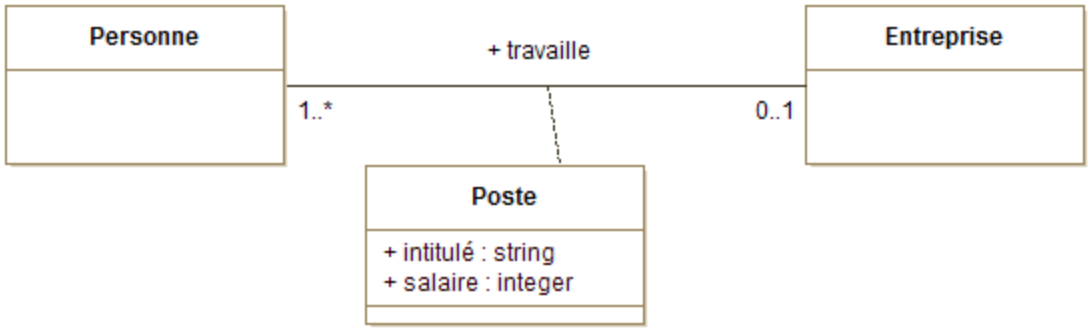
- Association entre classe : relation entre instance
  - Nom de l'association
  - Nom des rôles (optionnels)
  - Cardinalités de l'association
    - 0..1 : zéro ou 1 instance
    - n : exactement n instances de le classe
    - m..n : de m à n instance
    - * : zéro à plusieurs instances (maximum non fixé)
    - n..* : plus de n instances (maximum non fixé)





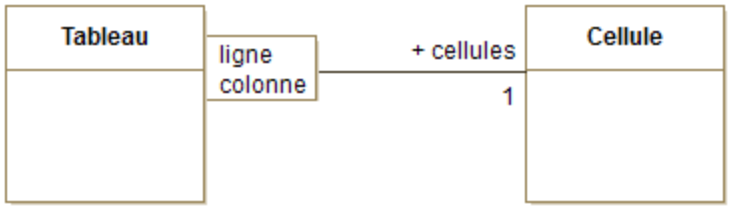
Association 2/3

- Classe d'association : association entre deux classes qui contient des attributs propres



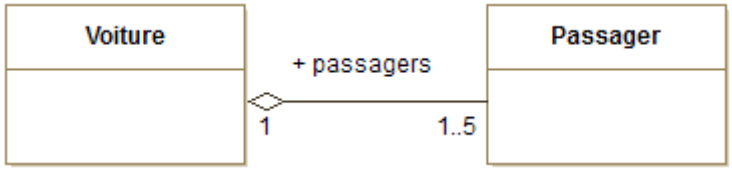
Association 3/3

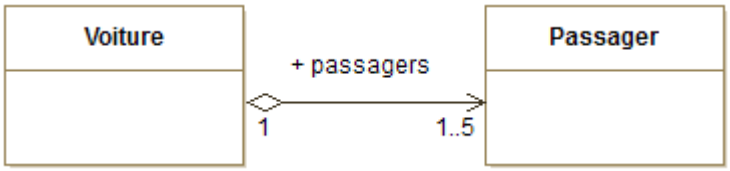
- Association qualifiée :
  - Possibilité de sélection d'un sous-ensemble d'objet associé à l'aide d'un ou plusieurs attributs qualificatifs (appelés clés)
  - En Java, ce type d'association se traduit souvent par une table d'indexage.



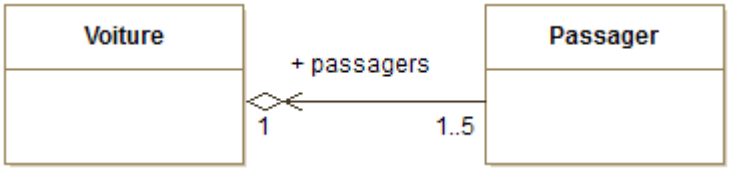
Agrégation

- Agrégation : association + les éléments existent toujours quand l'association est détruite





En connaissant une voiture, on peut accéder à la liste de ses passagers.

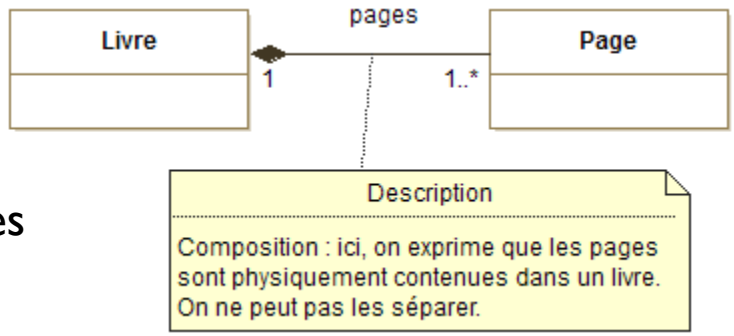


En connaissant un passager, on peut accéder à la voiture dans laquelle il est.

Composition

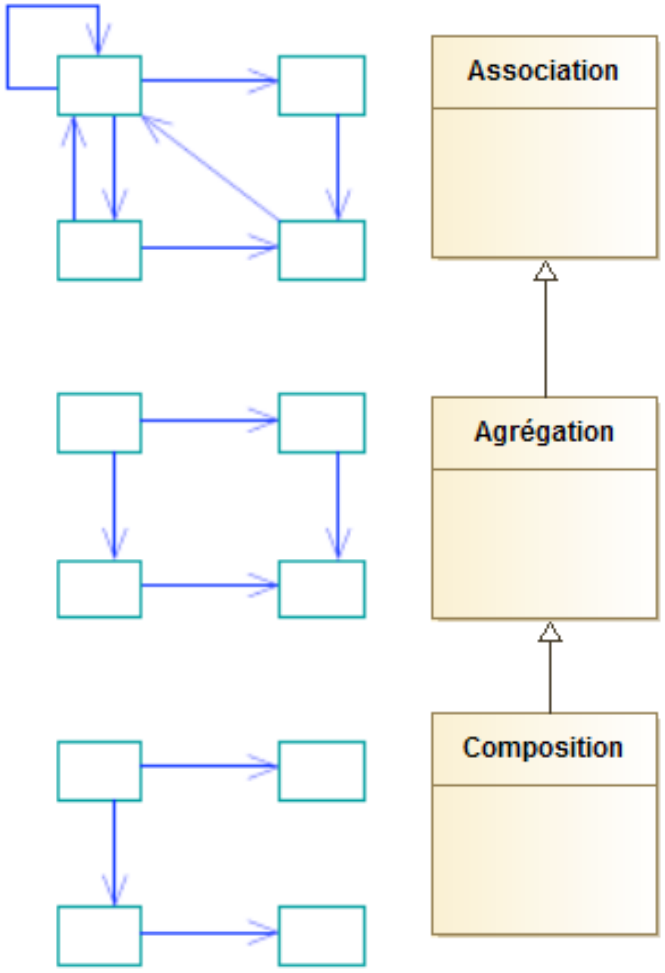
- Composition : les éléments disparaissent quand l'association est détruite

Si l'objet Livre est effacé, ses pages sont aussi effacées.



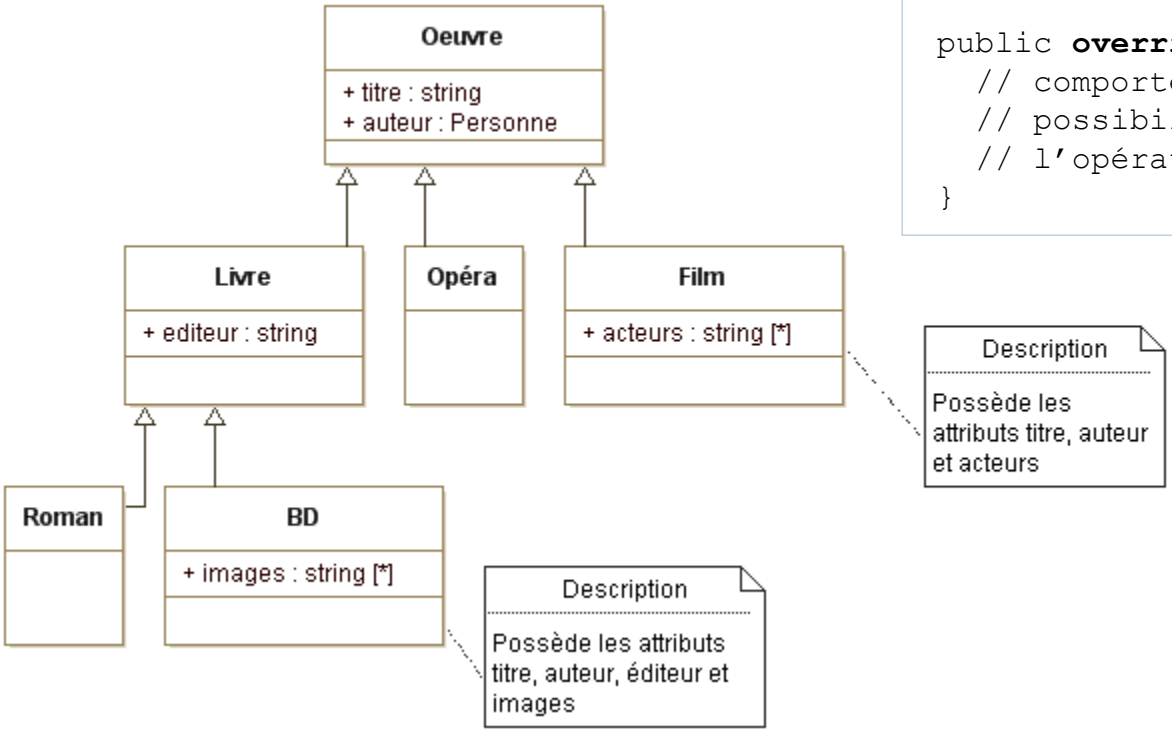
Différences entre les 3 types d'Association

- **Association** : A est relié à B
  - Type de la relation non spécifié
  - Aucune contrainte sur les liens
- **Agrégation** : A contient un ou plusieurs B
  - Association + éléments partageables
- **Composition** : À est composé d'un ou plusieurs B
  - Agrégation + contrainte de durée de vie + composants non partageables



Héritage

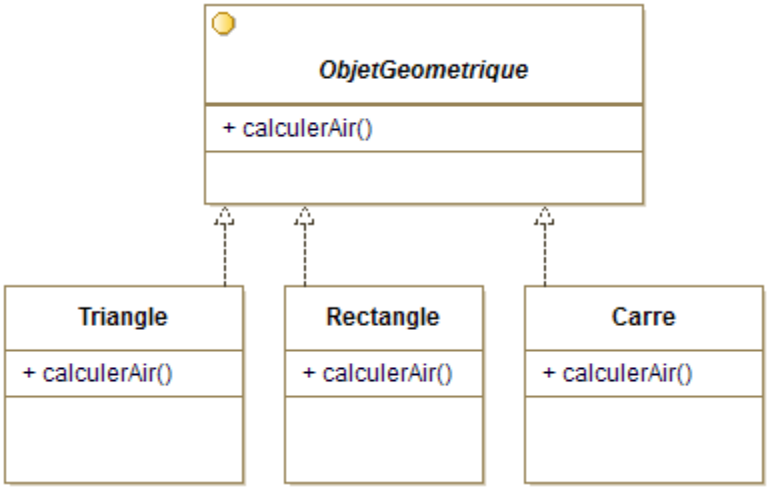
- Héritage : une classe hérite des propriétés (attributs et opérations) d'une autre classe.
  - Les opérations peuvent être substitué et spécifié




```java
public override int operationHerite() {
  // comportement spécifique
  // possibilité de faire appel à
  // l'opération héritée avec super()
}
```

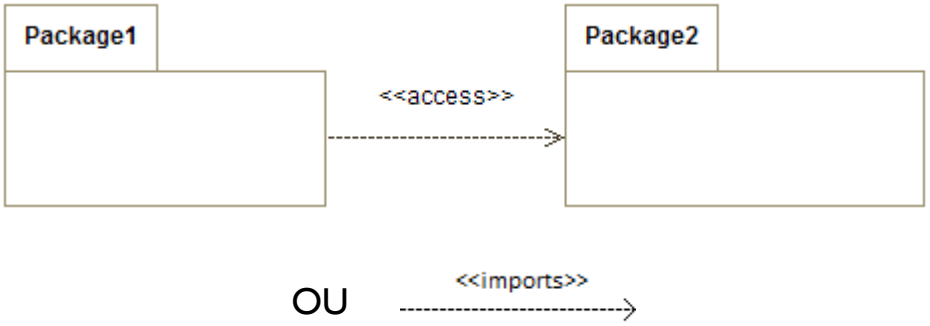
Implémentation

- Implémentation : une classe peut implémenter une interface. Pour que l'implémentation soit valide, cette classe doit spécifier le contenu de chaque opération définie par l'interface.



Dépendance

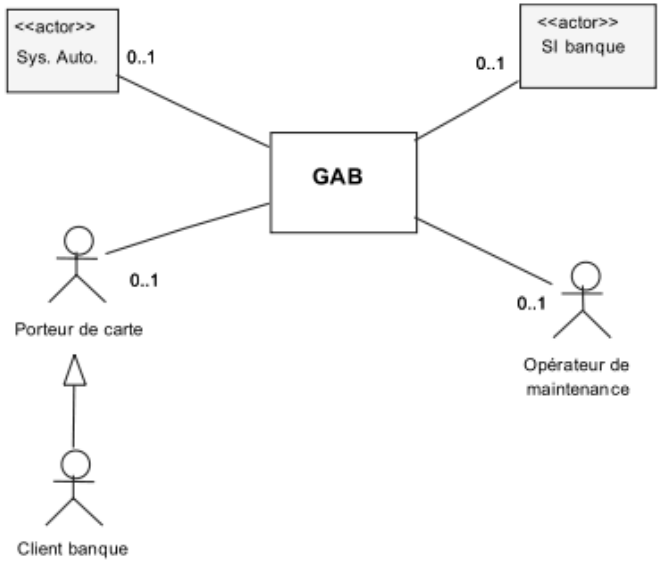
- Importation : un paquetage ou une classe peut « accéder » un autre paquetage ou classe.



- Utilisation : un paquetage ou classe peut utiliser un autre paquetage ou classe.

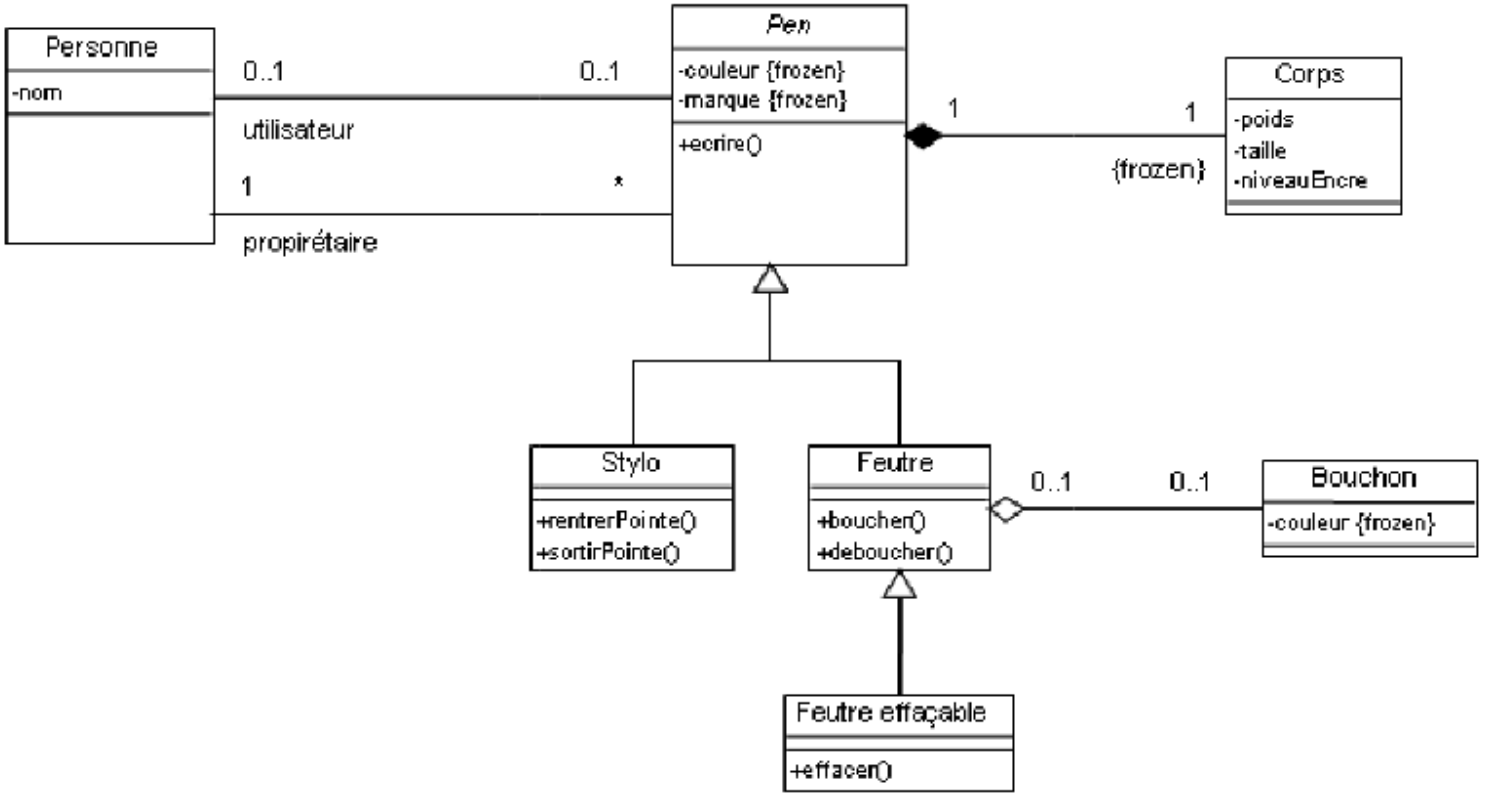
Exemples

- Diagramme de classes pour l'étude des besoins (niveau 1)



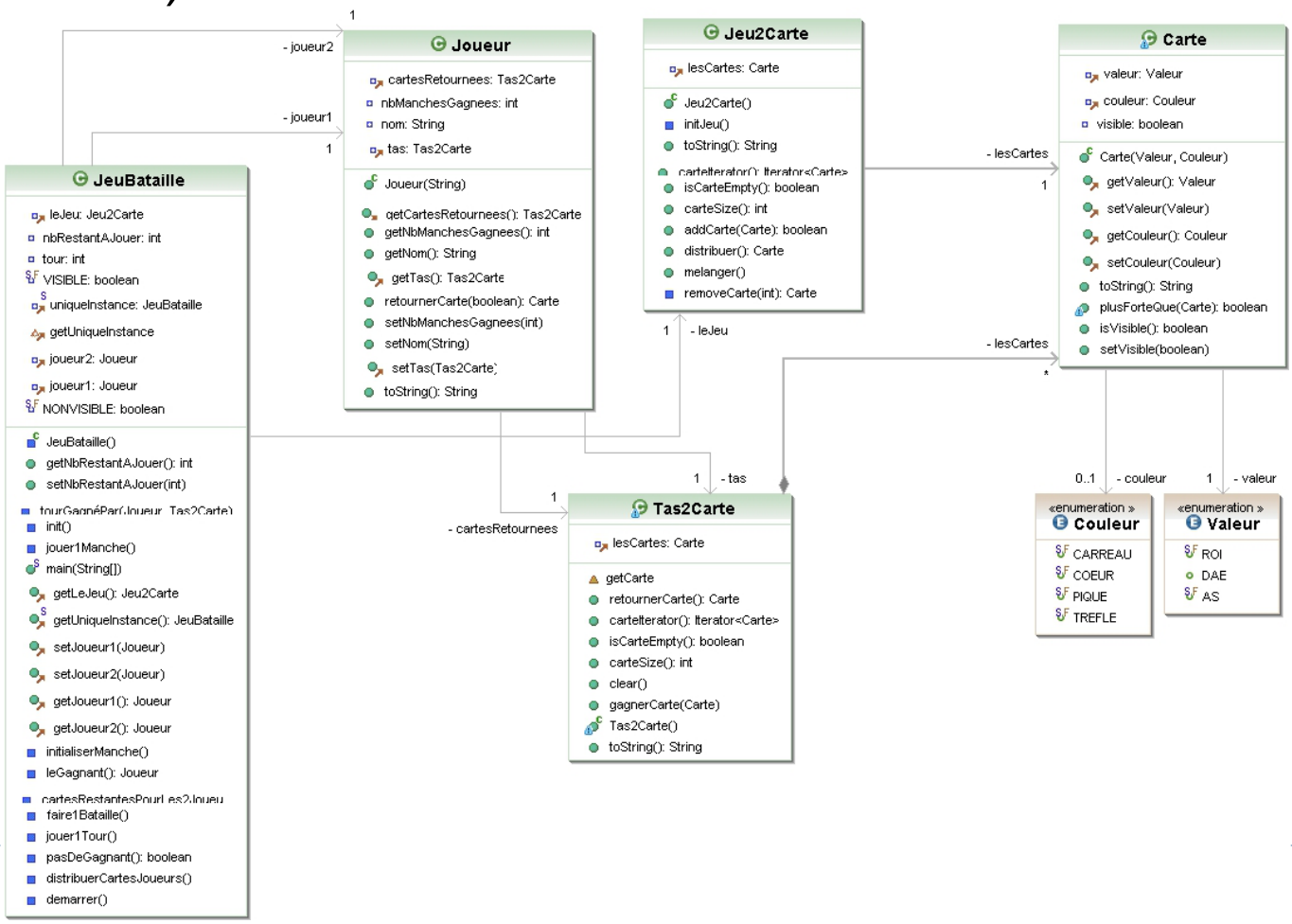
Exemples

- Diagramme de classes pour la conception niveau abstrait (niveau 2)



Exemples

- Diagramme de classes pour la conception niveau physique (niveau 3)



Exemples

- Diagramme de classes pour les bases de données (niveau 2 ou 3)

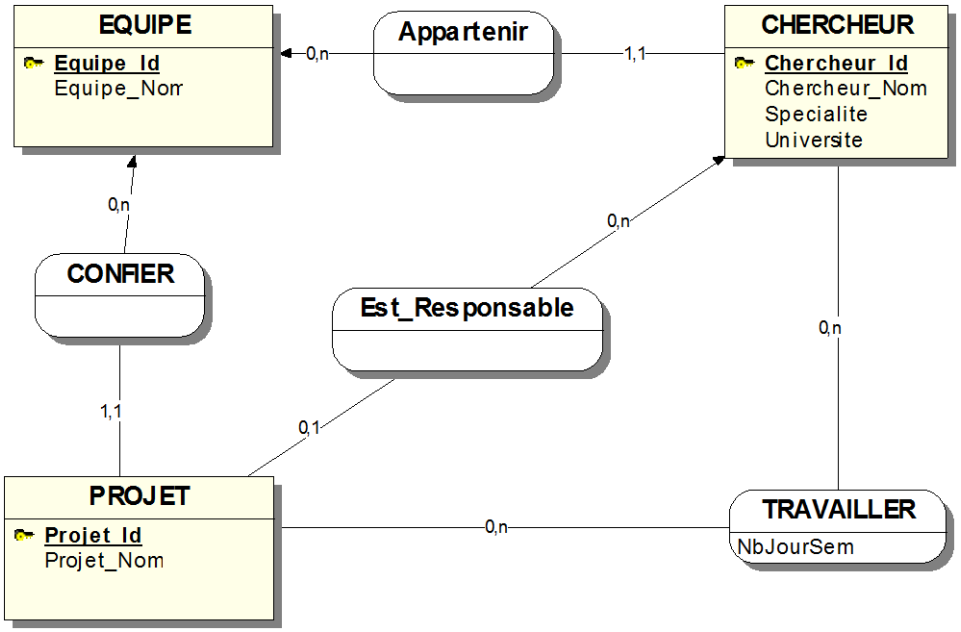


Table => Classes
Colonnes => Attributs

Les associations avec une cardinalité 0 ou 1.. 1 sont traduites en clé secondaire.

Les associations avec une cardinalité multiples sont traduites par une nouvelle table avec 2 clés secondaires.

Exercice 4

- Reprenez la feuille avec la description textuelle d'un système informatique.
  - Modéliser la partie structurelle du système avec un diagramme de classe.

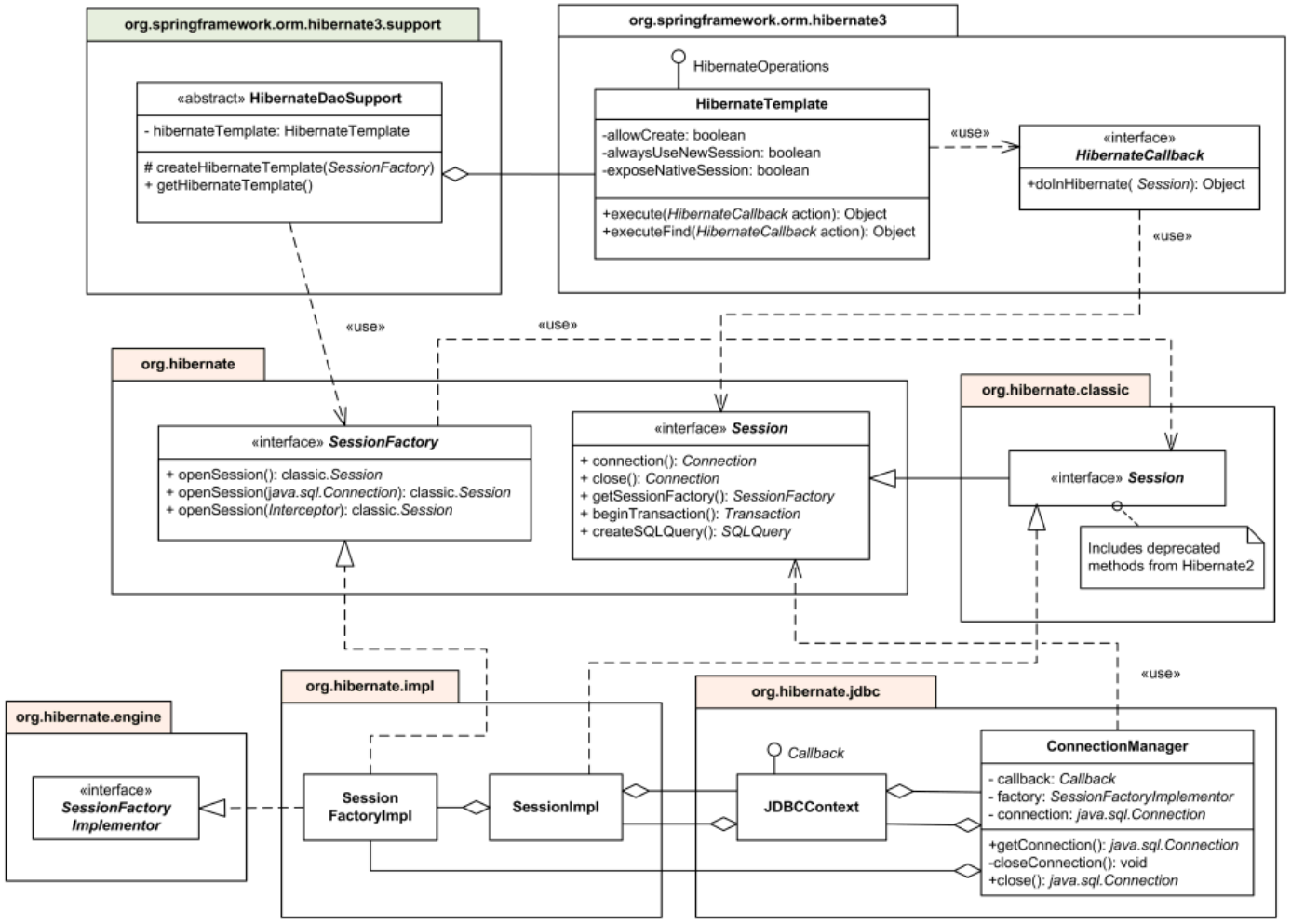
Diagramme de paquetage

Introduction au diagramme de paquetage

- Éléments représentés
  - Paquetages contenant des Classes et Interfaces
  - Relations entre les paquetages (importation)

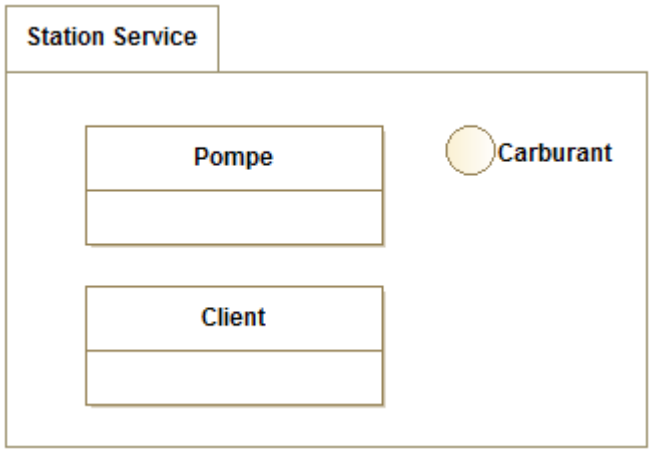
- Utilisation
  - Pendant la conception niveau abstrait (niveau 2)
    - modéliser les paquets cohérents qui composent le système (sans être reliés aux spécificités du langage ou de la plateforme de développement)
  - Pendant la conception niveau physique (niveau 3)
    - modéliser comment les fichiers sont physiquement organisés dans le projet Java
    - facilité la compréhension du système pour sa maintenance

Exemple


org.springframework.orm.hibernate3.support
«abstract» HibernateDaoSupport
- hibernateTemplate: HibernateTemplate
createHibernateTemplate(SessionFactory)
+ getHibernateTemplate()
org.springframework.orm.hibernate3
HibernateOperations
HibernateTemplate
-allowCreate: boolean
-alwaysUseNewSession: boolean
-exposeNativeSession: boolean
+execute(HibernateCallback action): Object
+executeFind(HibernateCallback action): Object
«use»
«interface»
HibernateCallback
+doInHibernate(Session): Object
«use»
«use»
«use»
org.hibernate
«interface» SessionFactory
+ openSession(): classic.Session
+ openSession(java.sql.Connection): classic.Session
+ openSession(Interceptor): classic.Session
«interface» Session
+ connection(): Connection
+ close(): Connection
+ getSessionFactory(): SessionFactory
+ beginTransaction(): Transaction
+ createSQLQuery(): SQLQuery
org.hibernate.classic
«interface» Session
Includes deprecated methods from Hibernate2
«use»
org.hibernate.engine
«interface»
SessionFactory
Implementor
org.hibernate.impl
Session
FactoryImpl
SessionImpl
org.hibernate.jdbc
Callback
JDBCContext
ConnectionManager
- callback: Callback
- factory: SessionFactoryImplementor
- connection: java.sql.Connection
+getConnection(): java.sql.Connection
-closeConnection(): void
+close(): java.sql.Connection

Paquetage

- Package : collection de classes et d'interface

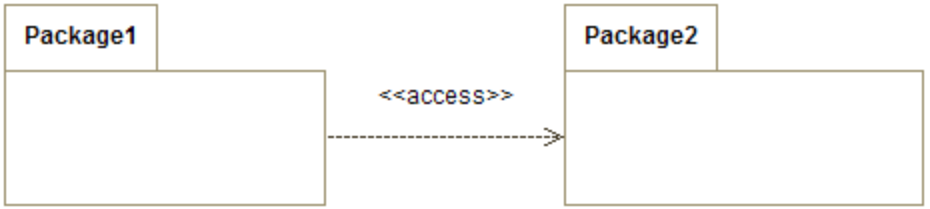


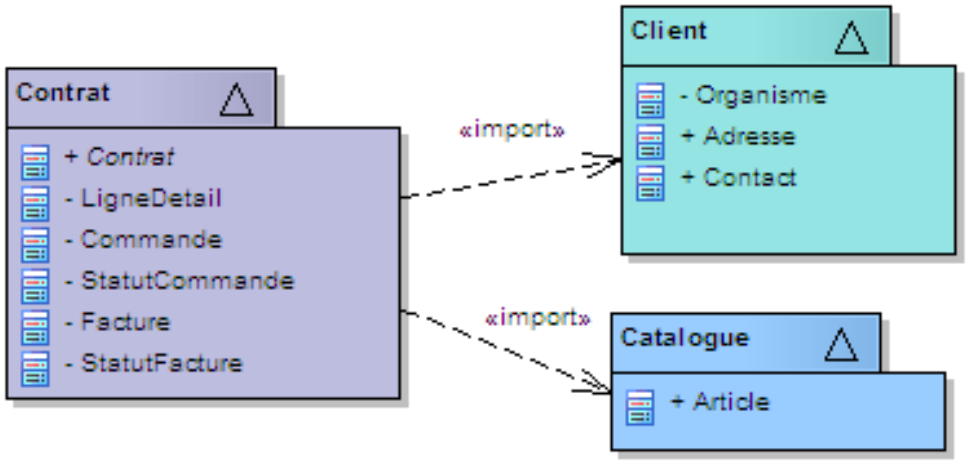
Java

```
En java, cela correspond au dossier
dans lequel se trouvent les fichiers
de description des classes et interfaces
```

Relation

- Relation de dépendance





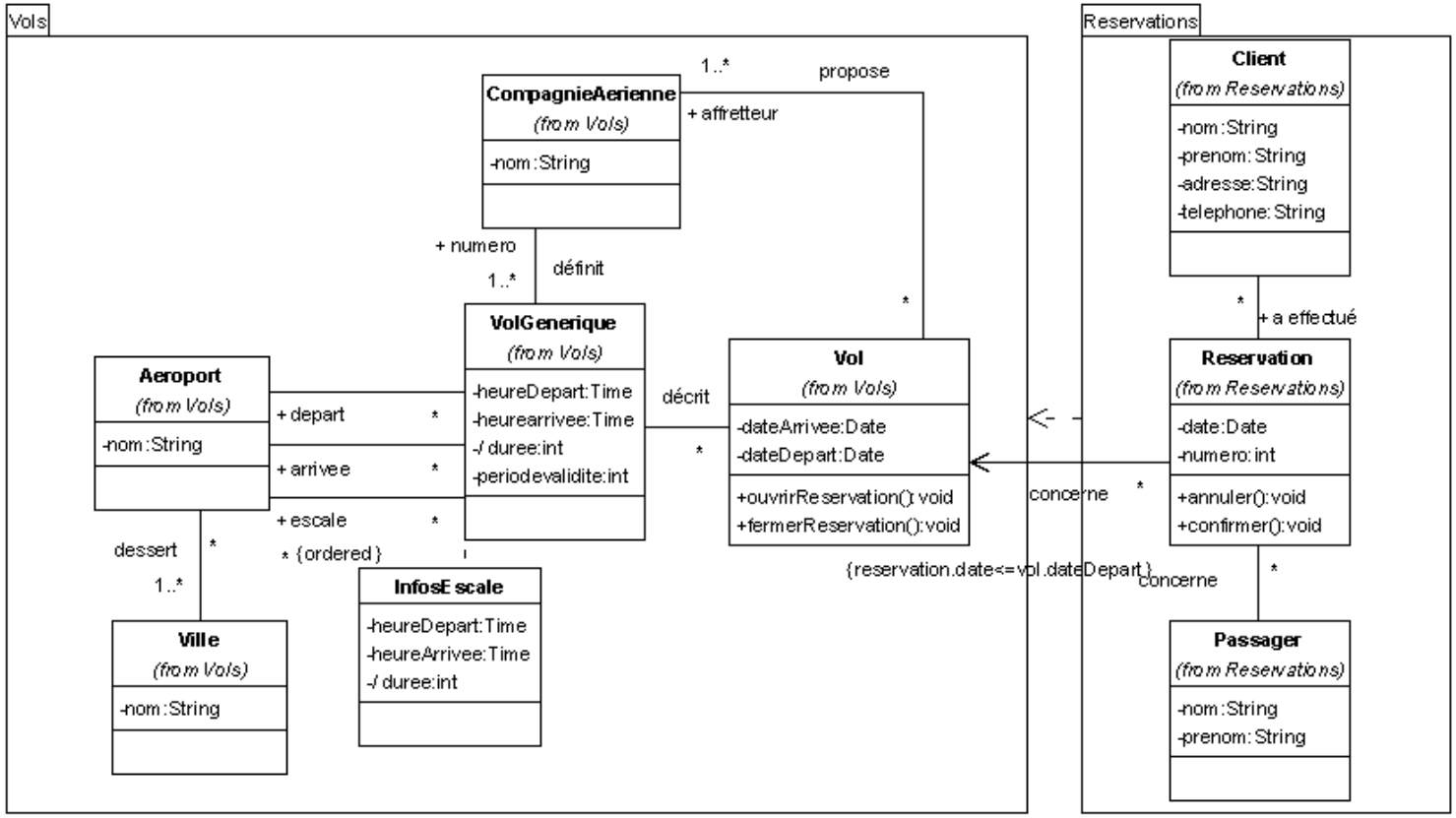
En java, cela correspond à l'importation du dossier destination dans toutes les classes du dossier initiale.

Java

```
package Contrat;

import Catalogue.*;
import Client.*;
```

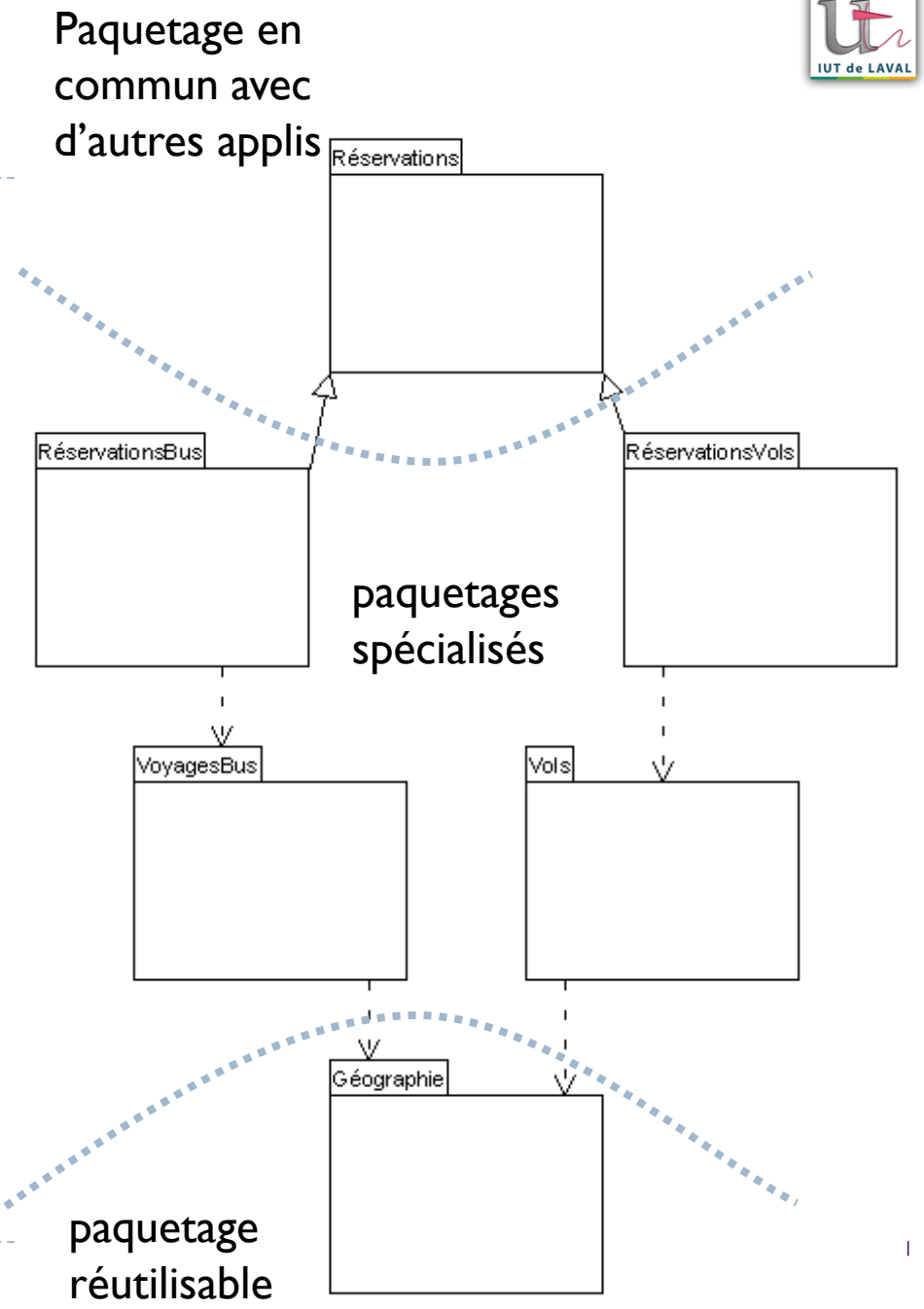
Exemple

- Diagramme de paquetage pour la conception niveau physique (niveau 3)



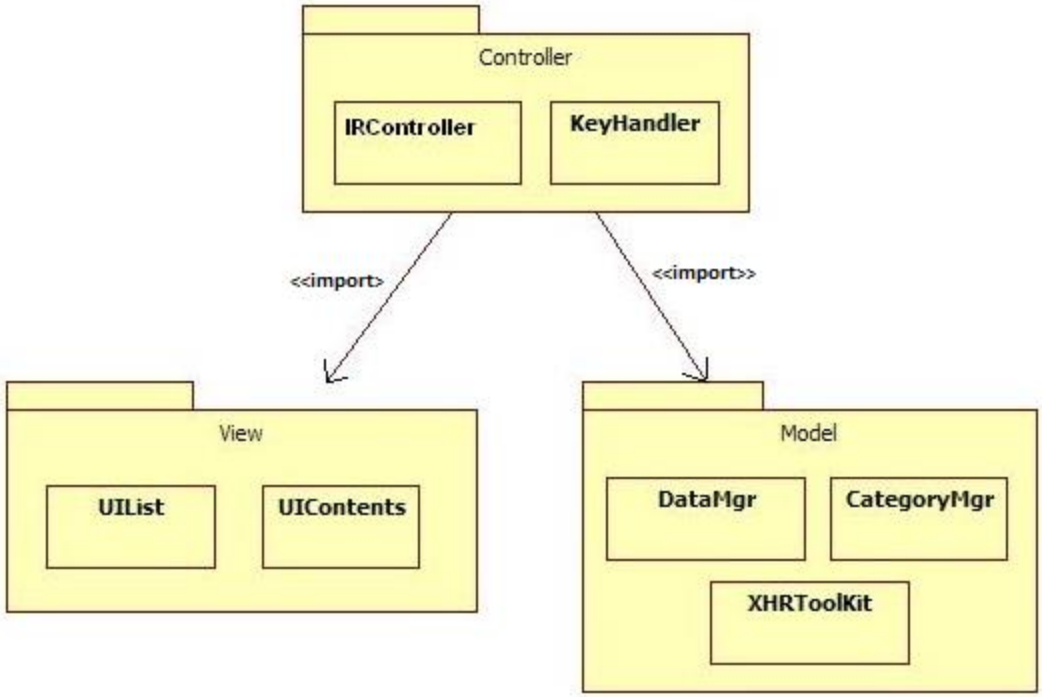
Exemple

- Diagramme de paquetage pour la conception niveau physique (niveau 3)
  - Chaque paquetage doit contenir une collection logique de classes
  - Le moins de relation possible entre paquetages



Exemple

- Diagramme de paquetage pour la conception niveau abstrait (niveau 2)
  - architecture MVC (Modèle - Vue - Contrôleur à voir l'année prochaine)



Du code au modèle et vice versa

Relation entre le code et le modèle

- Du code au modèle : reverse engineering
- Du modèle au code : génération de code automatique aussi appelée forward engineering
- Utilisation des deux pour concevoir un système de façon efficace et maintenir une cohérence entre le code et sa documentation

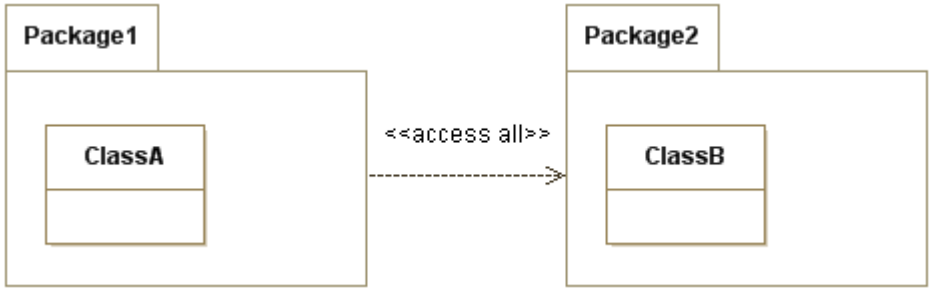
Règles du reverse Engineering

1. classe Java ⇨ classe UML

2. interface Java ⇨ interface UML

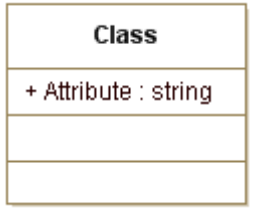
3. dossier Java ⇨ paquetage UML

4. classe Java A importe la classe B ⇨ dossier UML contenant la classe A importe toutes les classes du dossier contenant la classe B

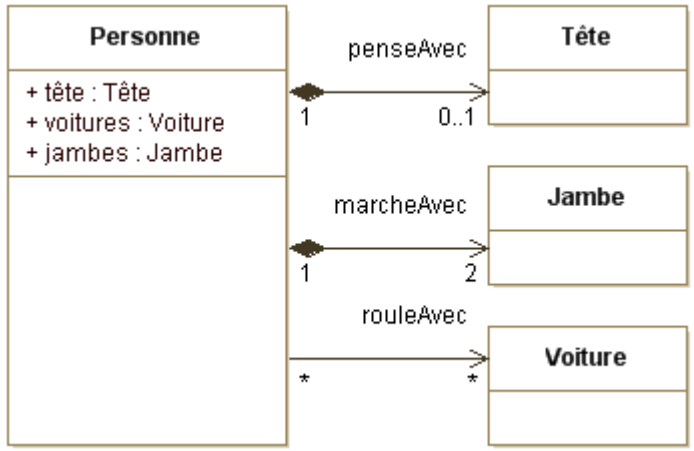


Règles du reverse Engineering

5. attribut de classe Java
   - Si attribut de type primitif (byte, short, int, long, float, double, boolean, char) ⇨ propriété de classe UML avec le type correspondant (integer, string, boolean, char, real)

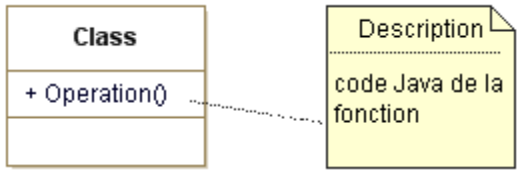


   - Si attribut de type autre classe java ⇨ association navigable (choisir le bon type) entre la classe UML parent et la classe UML correspondant à l'attribut. Si l'attribut est un tableau, utilisé l'association multiple, sinon 0..1

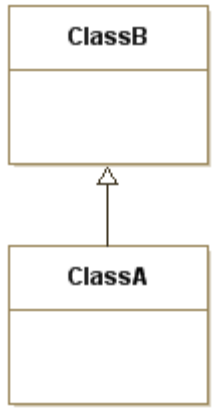


Règles du reverse Engineering

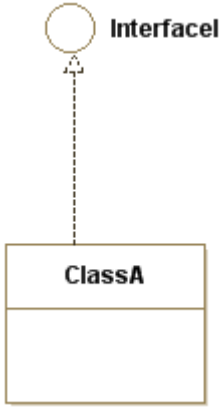
6. fonction de classe Java ⇨ opération de classe UML lié à une note UML contenant le code interne de la fonction



7. classe Java A hérite de classe B ⇨ classe UML A hérite de classe B



8. classe Java A réalisé l'interface I ⇨ classe UML A réalisé l'interface I



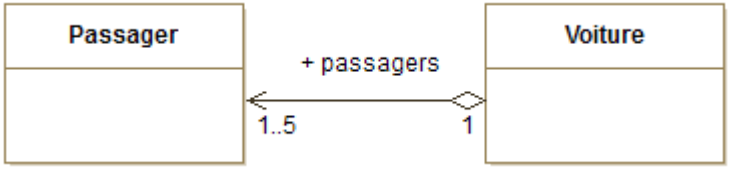
Génération de code automatique

- Les règles sont essentiellement inversent de celles du reverse engineering
- Peut être réalisé par certain plugin Java comme Omondo ou eUML2
- Le modèle doit être complet, cohérent et respecter les règles de l'UML pour JAVA

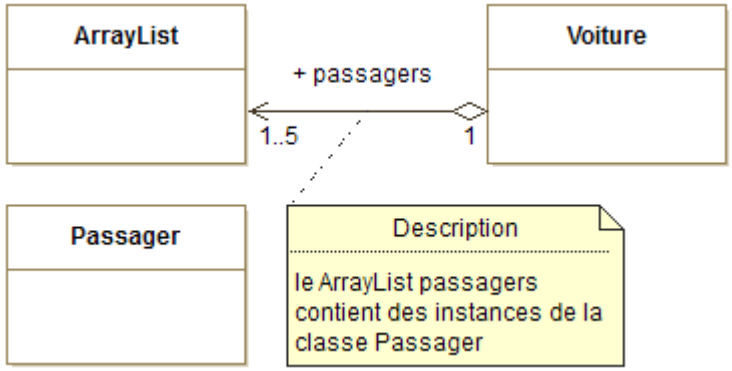
UML pour Java

- UML pour Java = UML + contraintes pour permettre génération de code automatique
  - Les relations doivent toutes avoir un sens de navigation
  - Le code Java pour chaque opération doit être dans une Note attachée à celle-ci
  - Une classe ne peut hériter que d'une seule classe (plusieurs interfaces possibles)
  - Utilisation des classes particulières à l'API Java (ArrayList, Stack, JPanel, EventListener…)

Exemple : les associations multiples



UML pure



UML pour Java

3. Modéliser le comportement avec UML

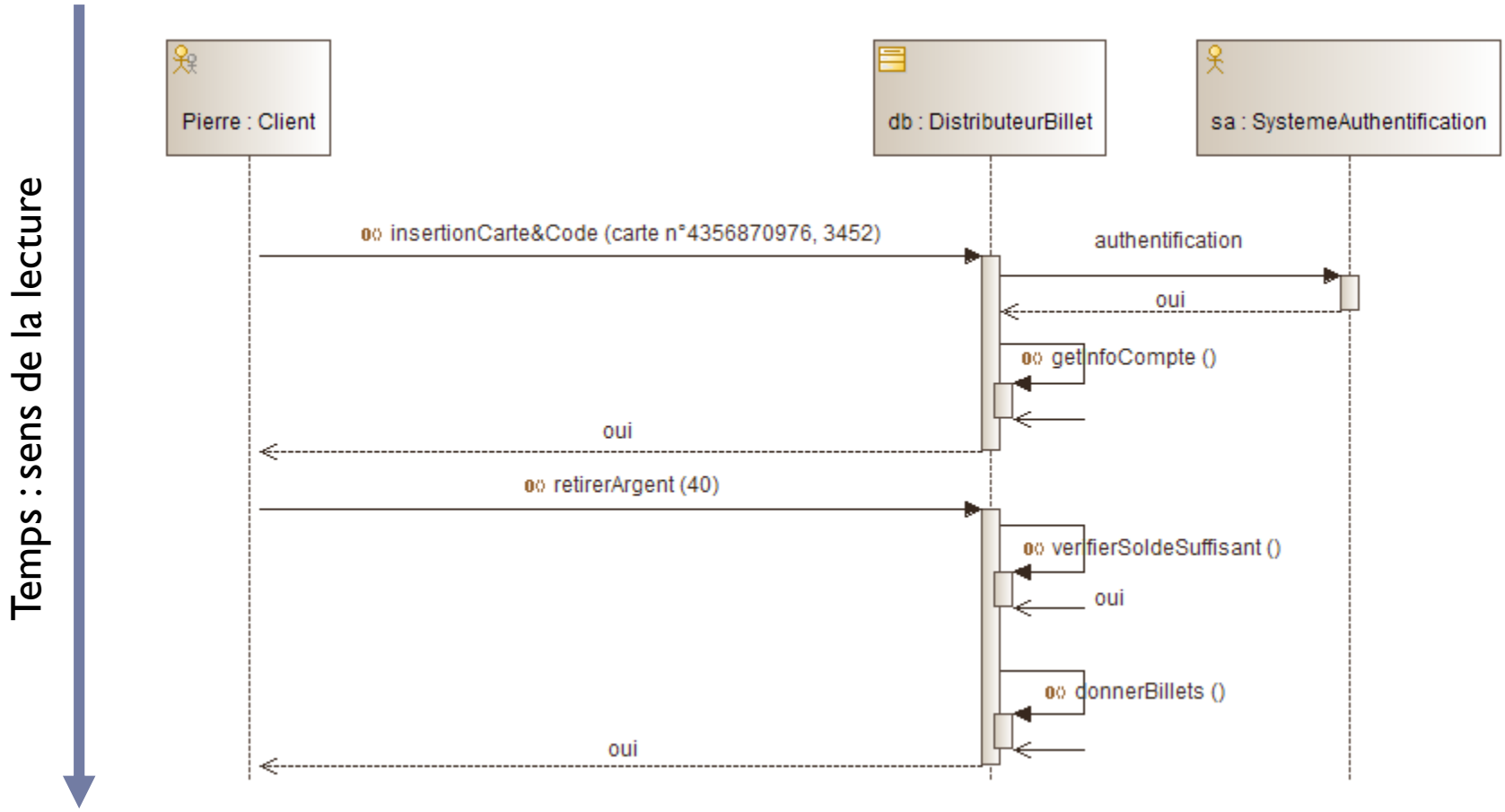
Diagramme de séquence

Introduction au diagramme de séquence

- Éléments représentés
  - Objets : acteurs externes ou instances de classes
  - Interactions entre ces objets par rapport au temps
  - Frame : ensemble d'interactions

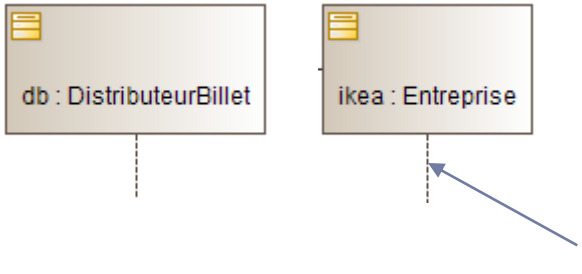
- Utilisations
  - Pendant l'étude des besoins (niveau 1)
    - identifier les scénarios d'interaction type entre les acteurs principaux et le système
  - Pendant la conception niveau abstrait (niveau 2)
    - comprendre comment les objets du système interagissent entre eux
  - Pendant la conception niveau physique (niveau 3)
    - décrire les appels/retours de méthodes entre instances pour illustrer un comportement entre différents objets
    - spécifier des tests unitaires

Exemple


Temps : sens de la lecture
Pierre : Client
db : DistributeurBillet
sa : SystemeAuthentification
insertionCarte&Code (carte n°4356870976, 3452)
authentification
oui
getInfoCompte ()
oui
retirerArgent (40)
verifierSoldeSuffisant ()
oui
donnerBillets ()
oui

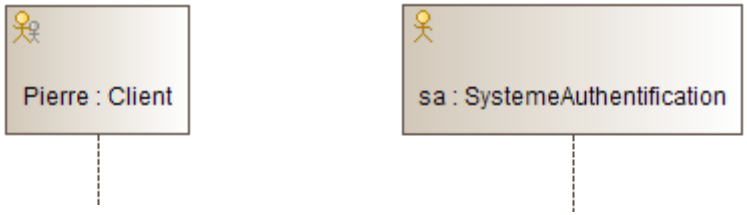
Objet

- Objet
  - Instance du système ou d'un classe du système



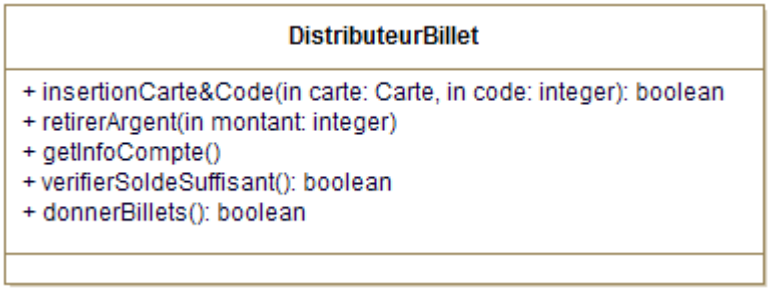

```java
var db:DistributeurBillet = new DistributeurBillet();
var ikea:Entreprise = new Entreprise();
```

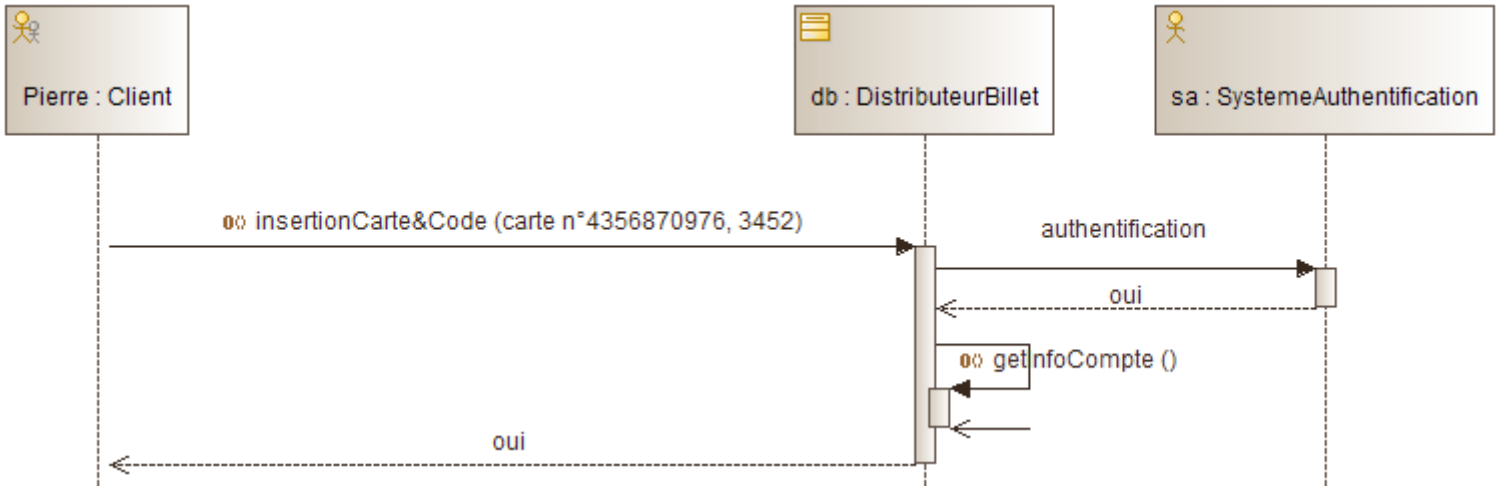
  - Instance d'un acteur externe (humain ou un autre système)



Interactions 1/2

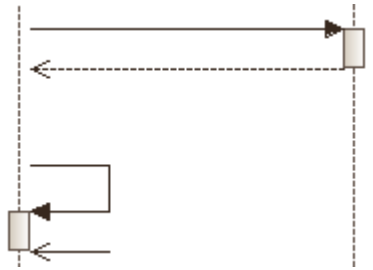
- Interactions :
  - Message : appelle à une opération de la classe destination avec des paramètres
  - Réponse : type de retour de l'opération

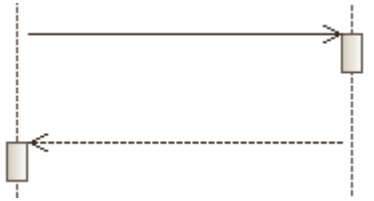




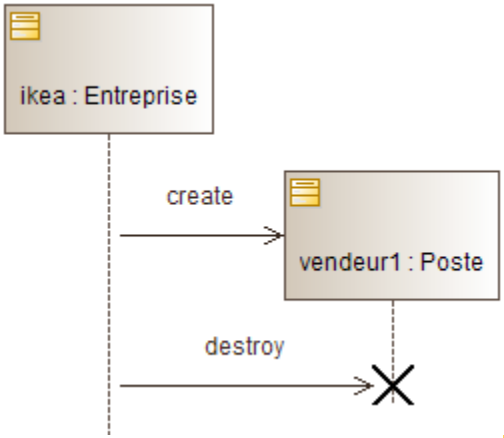
Interactions 2/2

- Type de messages
  - Message synchrone : l'émetteur est bloqué jusqu'au traitement effectif du message. La réponse arrive tout de suite après l'appel de l'opération. Représentation habituelle des opérations en JAVA.





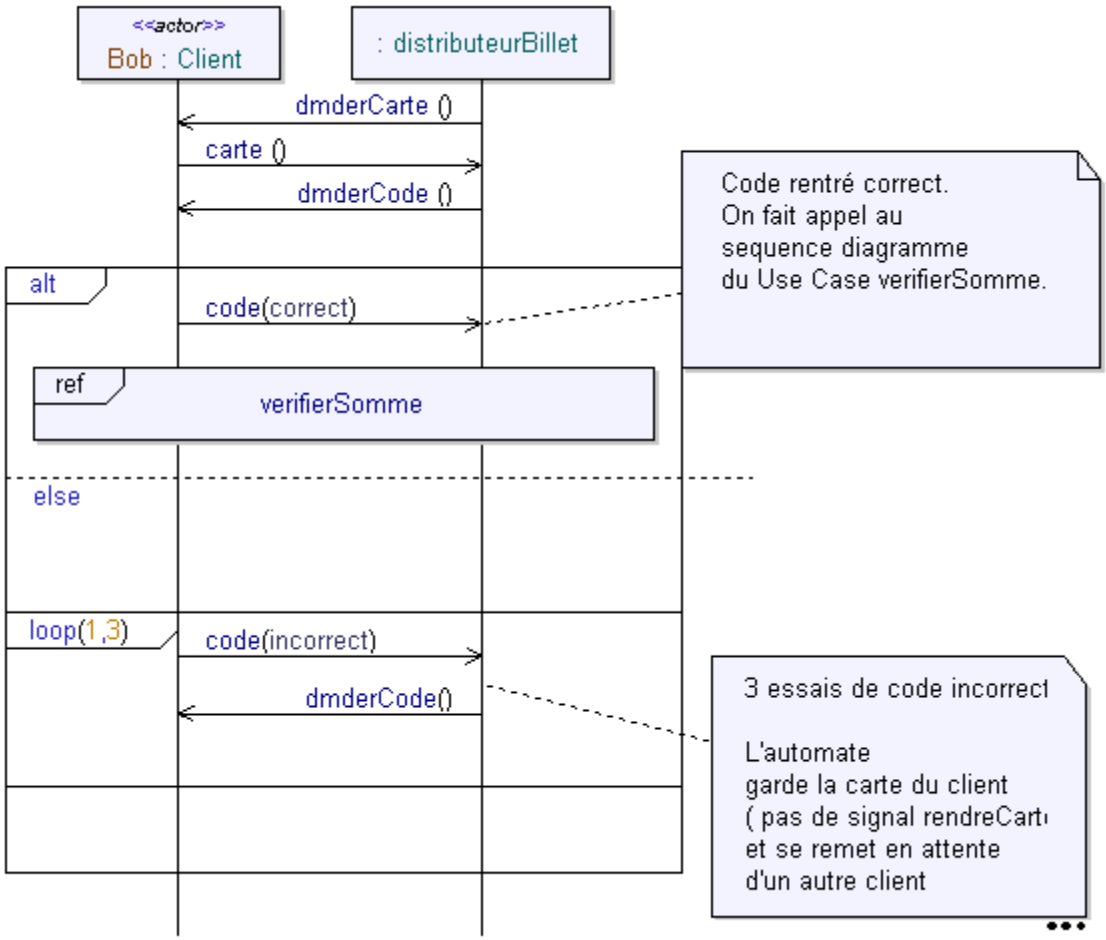
  - Message asynchrone : l'émetteur n'est pas bloqué et il peut poursuivre son exécution. La réponse arrive plus tard.



  - Message de création et de destruction d'un objet (en JAVA, il s'agit de l'appel au constructeur d'une classe).

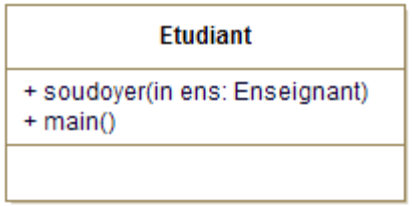
Frame

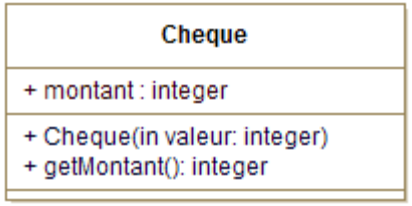
- Frame (aussi appelé fragment compilé) représentent des articulations d'interactions.
- Types les plus courants :
  - loop : boucle
  - alt : choix de l'utilisateur
  - opt : si… alors
  - par : interactions qui s'exécutent en parallèle
  - ref : référence à un autre diagramme de séquence

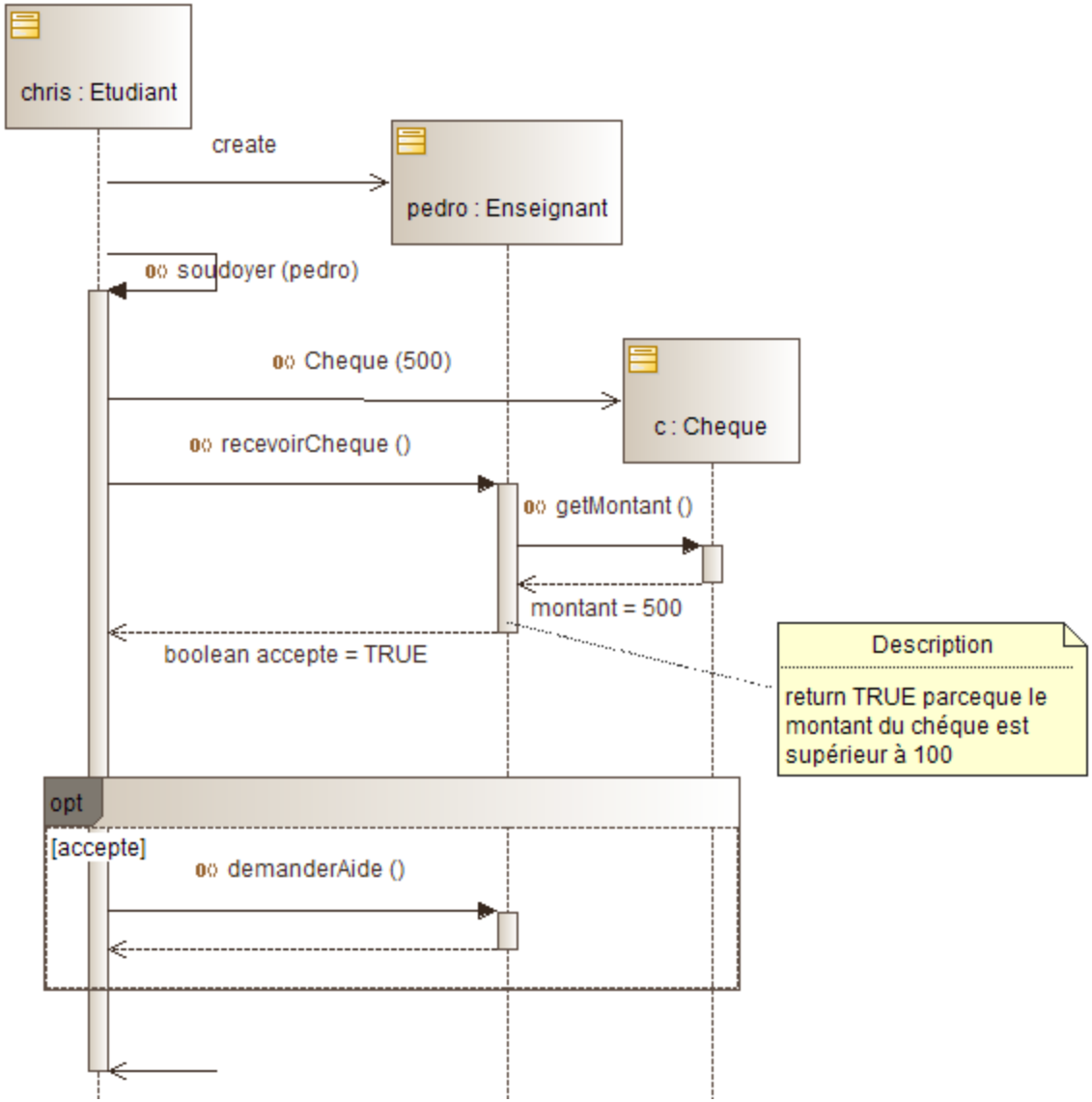


Cohérence avec le diagramme de classe

- Classes
- Opérations dans la classe destination avec le bon type de paramètres entrée en sortie







Cohérence avec le code

- Classes, attributs et opérations
- Séquence d'opération dans l'ordre

Etudiant.java | Enseignant.java | Cheque.java

```
package cours;

public class Etudiant {


    Etudiant (){

        Enseignant pedro = new Enseignant();
        soudoyer(pedro);
    }


    public void soudoyer(Enseignant pEnseignant){

        Cheque newCheque = new Cheque(500);
        boolean accepte = pEnseignant.recevoirCheque(newCheque);

        if(accepte){
            pEnseignant.demanderAide();
        }
    }

}
```

Etudiant.java | *Enseignant.java | Cheque.java

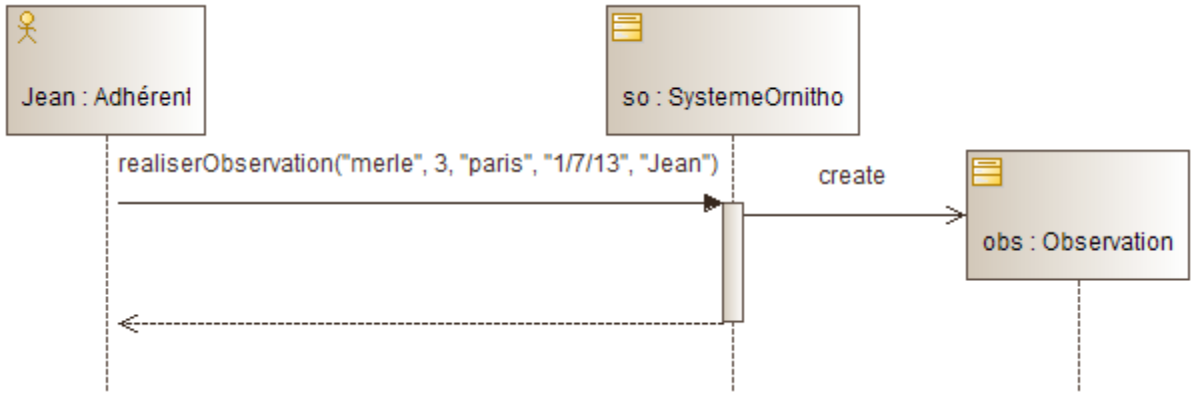
```
package cours;

public class Enseignant {

    public void demanderAide(){

    }

    public boolean recevoirCheque(Cheque pCheque){
        if(pCheque.getMontant() > 100){
            return true;
        }else{
            return false;
        }
    }
}
```

Etudiant.java | *Enseignant.java | Cheque.java

```
package cours;

public class Cheque {

    public int montant;

    Cheque(int pMontant){
            montant = pMontant;
        }

    public int getMontant(){
            return montant;
        }
}
```

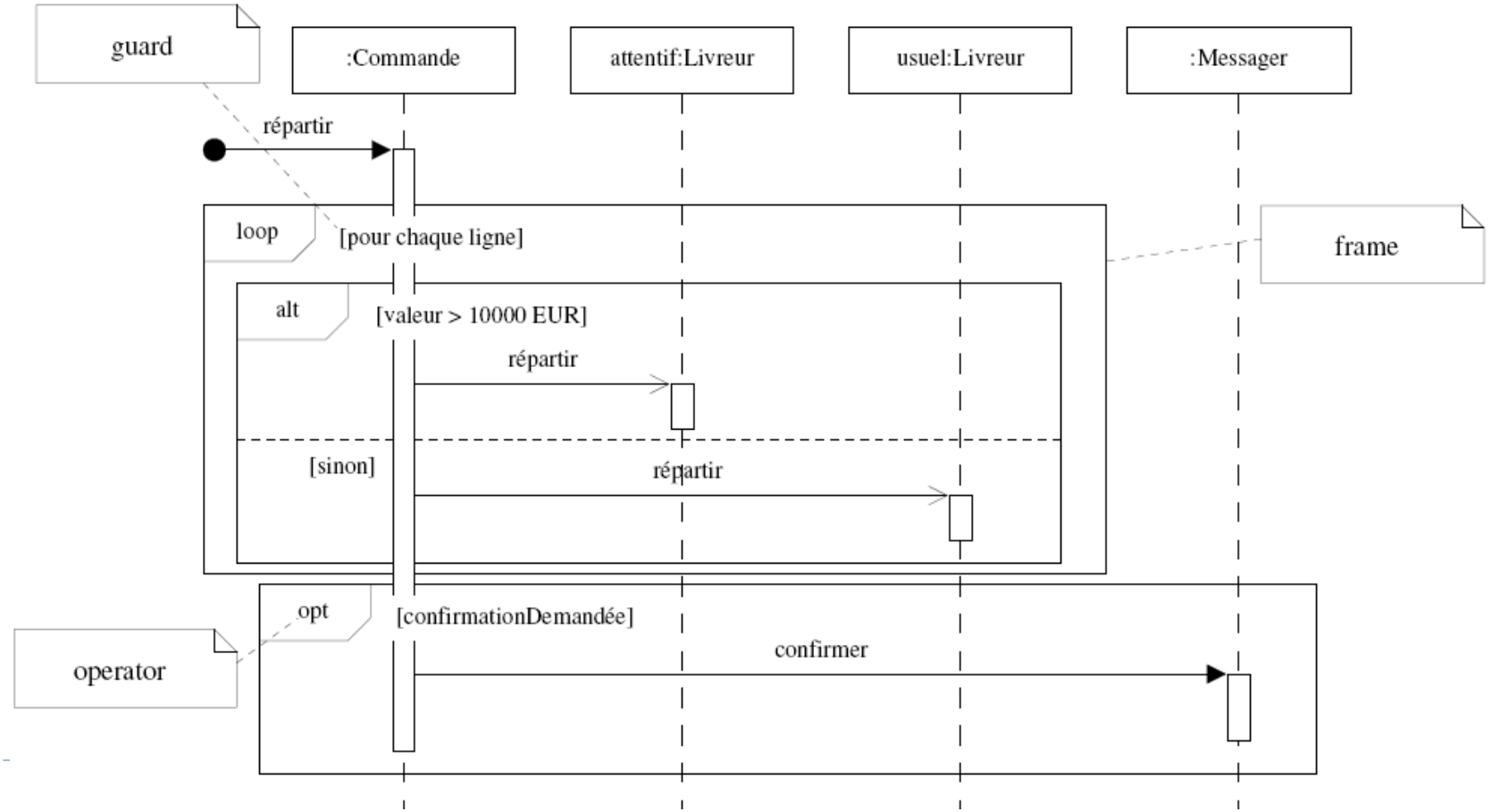
Exemples

- Diagramme de séquence pour l'étude des besoins (niveau 1)
  - Objets : les acteurs externes + le système (représenté comme 1 seul objet)
  - Interactions : les fonctionnalités système utilisées par les acteurs externes



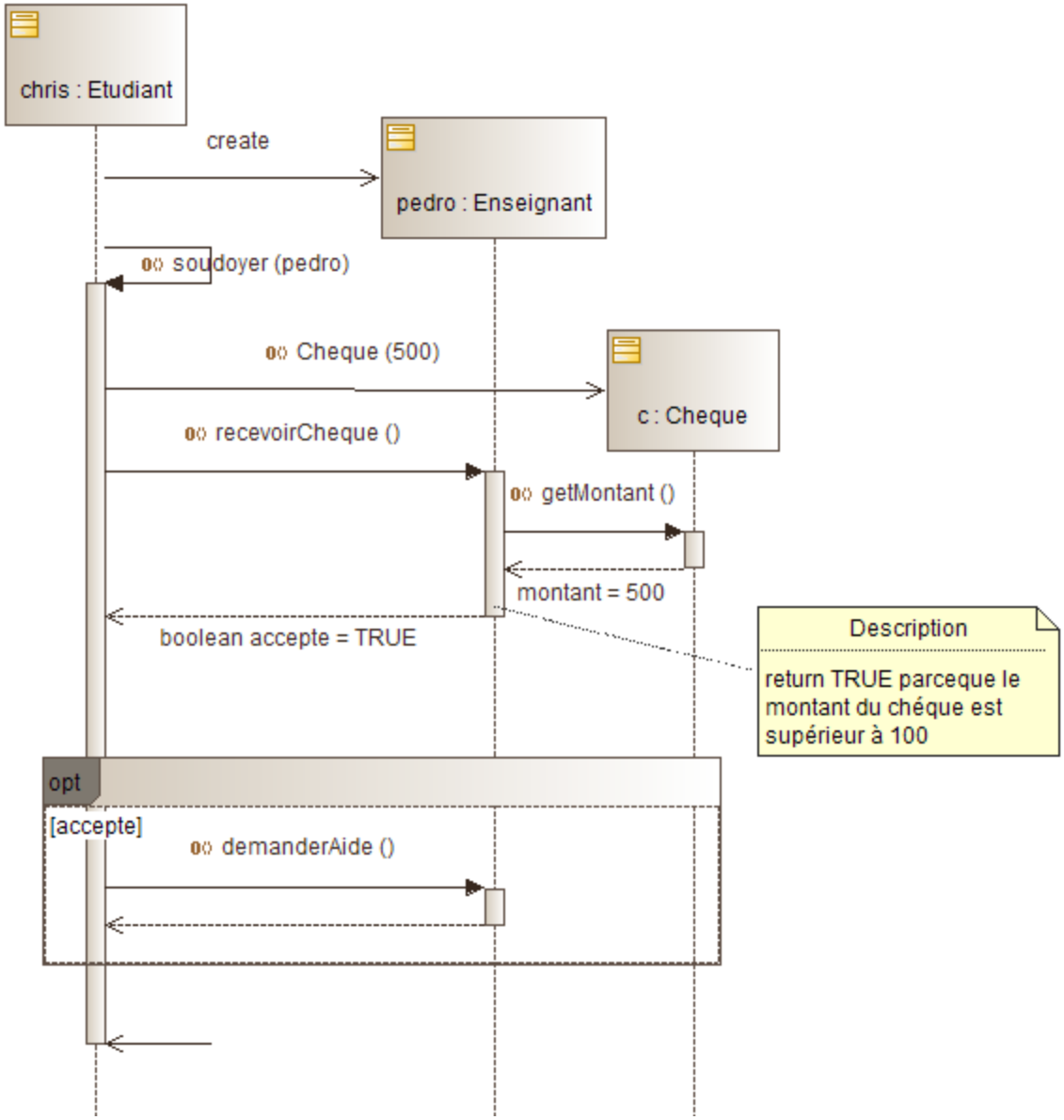
Exemples

- Diagramme de séquence pour la conception niveau abstrait (niveau 2)



Exemples

- Diagramme de séquence pour la conception niveau physique (niveau 3)
  - Objets = classes du système
  - Interactions = opérations des classes avec l'envoi de leurs paramètres et leur type de retour



Exemples

- Diagramme de séquence pour la génération de teste de validation du code
  - Objets = 1 objet « Testeur » + classes du système
  - Interactions = 1 scénario possible
  - Note en fin de séquence pour décrire le comportement attendu
- Pour chaque opération du système, prévoir au moins 1 teste sans erreur et 1 test avec erreur pour décrire le comportement souhaité.

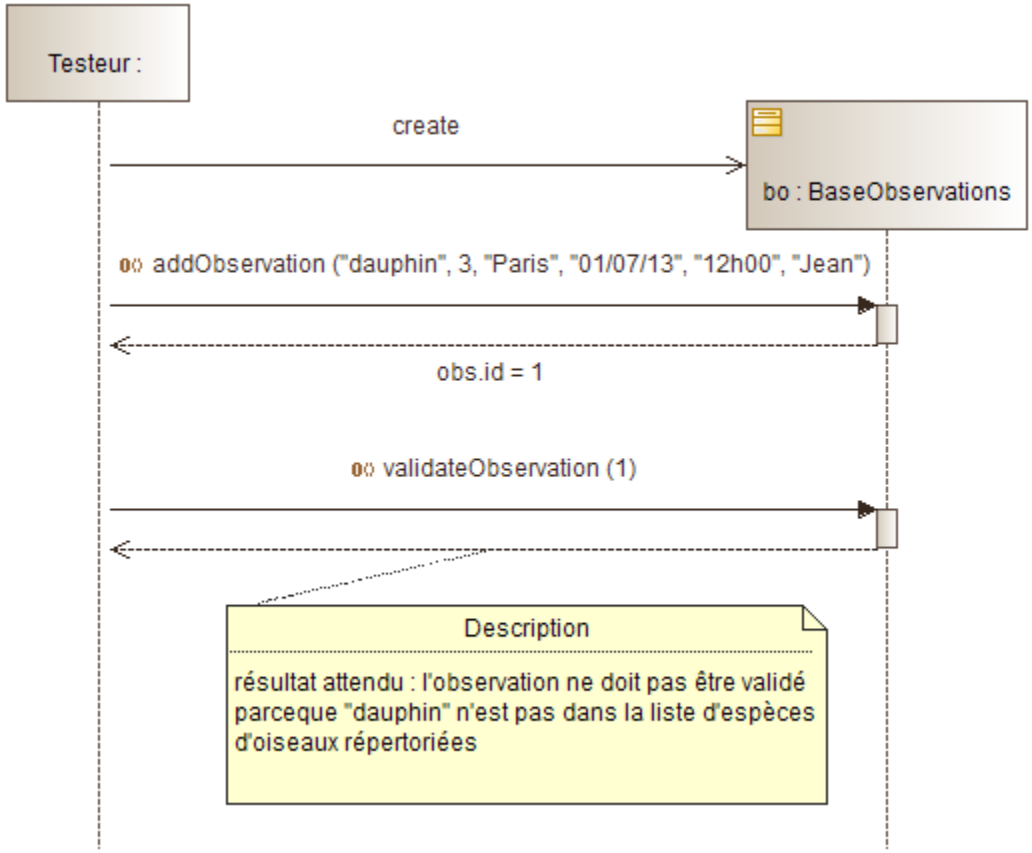


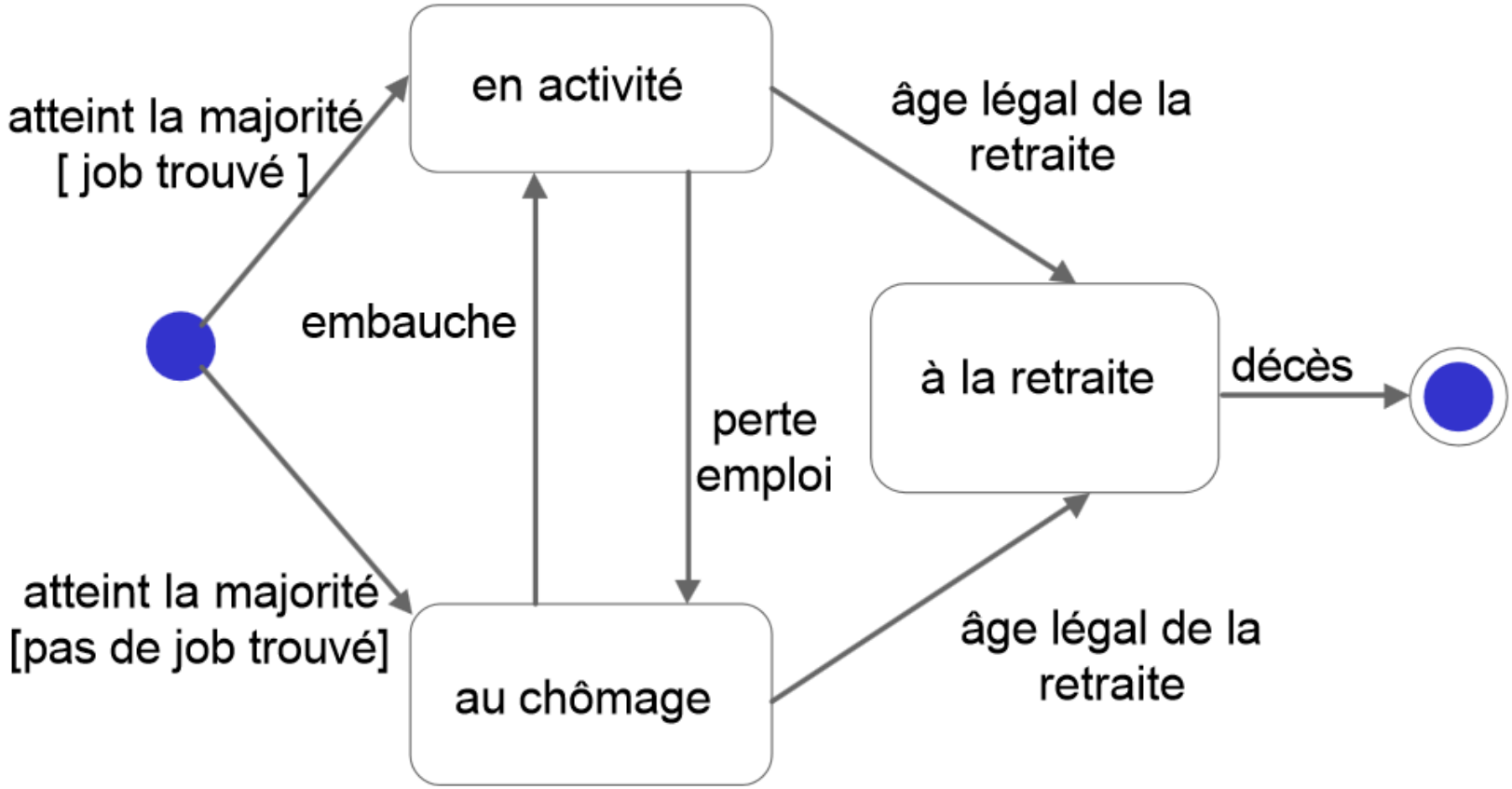
Diagramme d’état transition

Introduction au diagramme d'état transition

- Éléments représentés
  - État : configuration particulière des valeurs des attributs d'un objet pendant une certaine durée
  - Transition : passage d'un état une autre
  - Évènement : occurrence d'un stimulus susceptible d'entrainer le déclenchement d'une réaction au sein du système

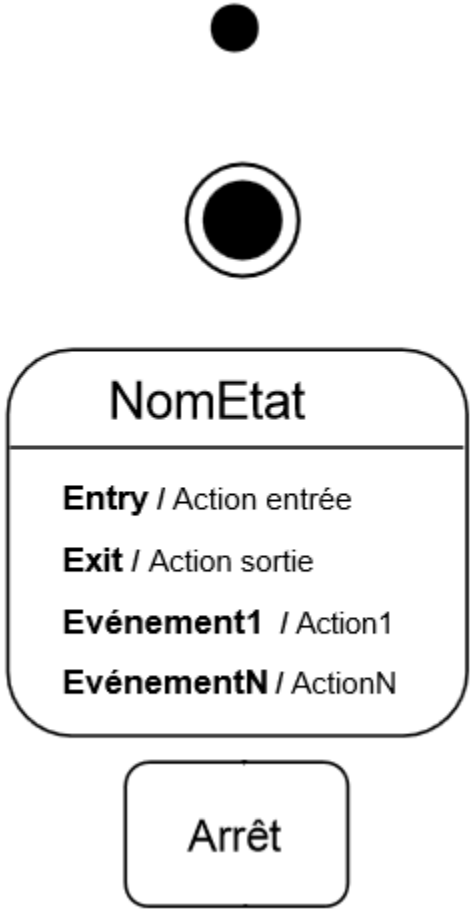
- Utilisations
  - À tous les niveaux
    - spécifier les états et les évolutions possibles d'objets qui ont un comportement complexe

Exemple


en activité
atteint la majorité
[job trouvé]
âge légal de la
retraite
embauche
à la retraite
décès
perte
emploi
atteint la majorité
[pas de job trouvé]
au chômage
âge légal de la
retraite

État

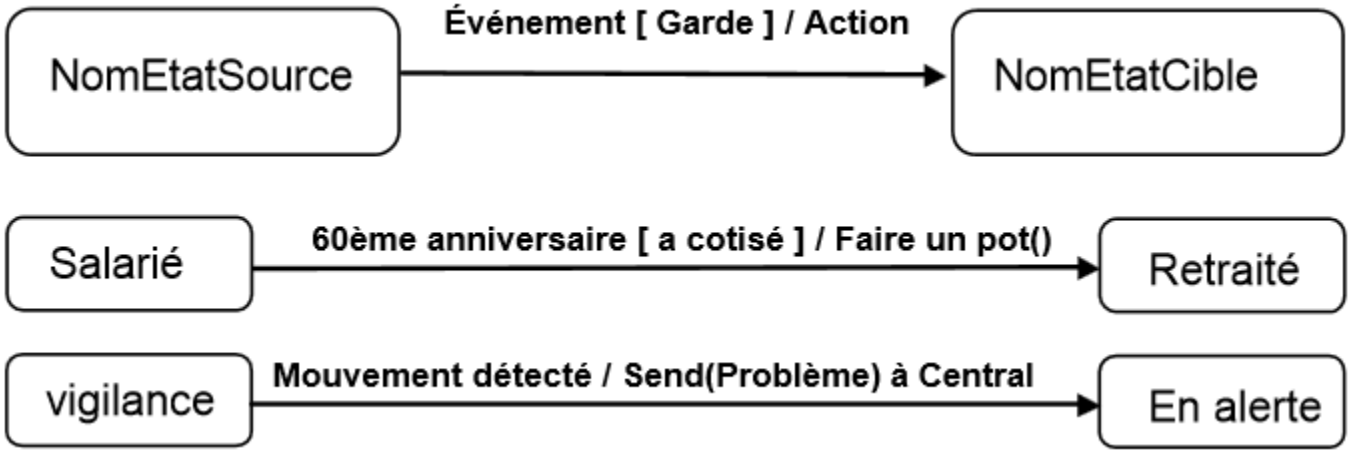
- État
  - État initial (obligatoire) : point de départ par défaut du diagramme
  - État final (possibilité d'en avoir plusieurs) : indique que le cycle de vie de l'instance s'achève
  - État simple (avec détails ou non)



- Le diagramme doit contenir tous les états possibles d'un unique objet à travers l'ensemble des cas d'utilisation dans lequel il est impliqué

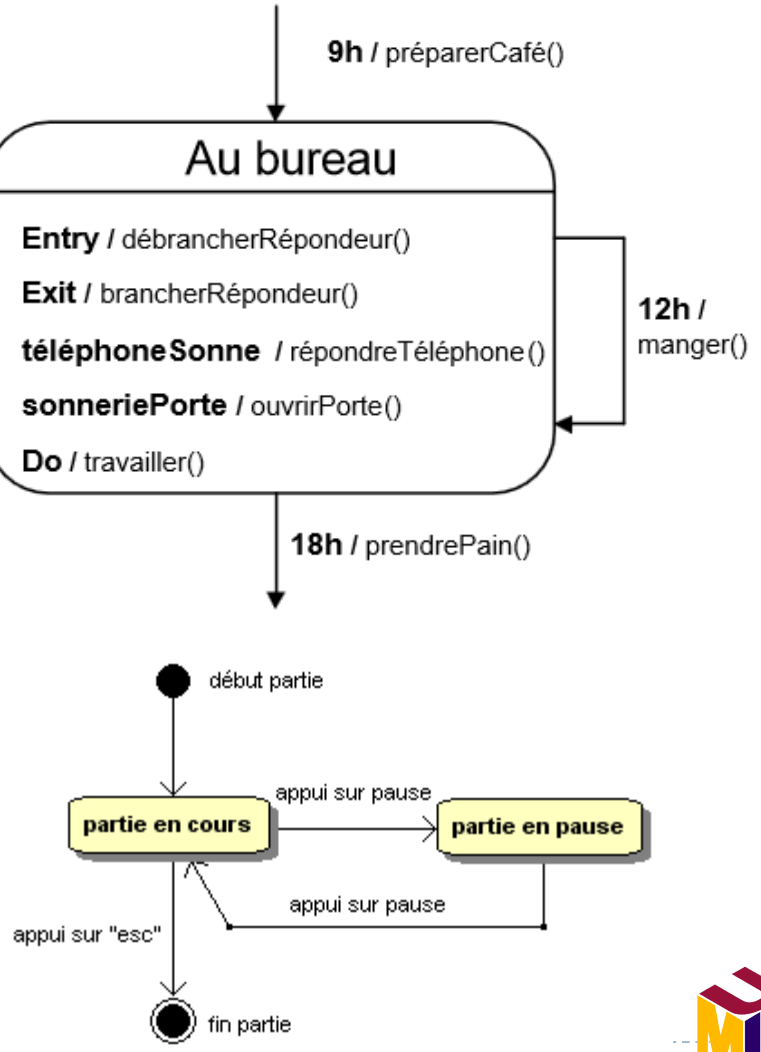
Transition

- Transition
  - Connexion orientée entre deux états, se déclenchant lorsque l'évènement auquel elle est associée est reçu par l'objet
  - Description :
    - Un évènement déclencheur
    - Une condition booléenne de garde qui doit être vérifiée pour que la transition puisse être déclenchée lors de la survenue de l'évènement
    - Une action



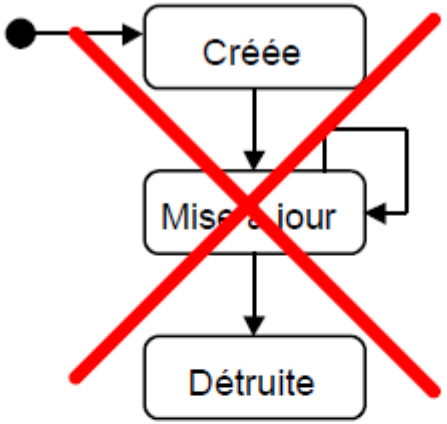
Évènement

- Évènement
  - La provenance de l'évènement peut être aussi bien interne, qu'externe au système
  - Types d'évènement :
    - Évènement sur condition (embauche, défaillance…)
    - Évènement temporel (écoulement d'une période de temps ou date définie)
    - Signal explicitement émis par un objet (clique sur bouton)

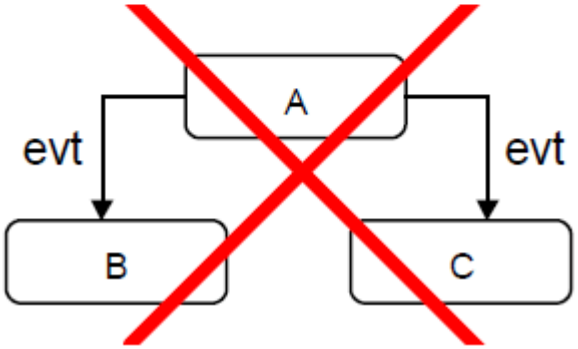


Recommandations

- Faire un diagramme d'état transitions uniquement pour les classes ayant un comportement significatif dans le système

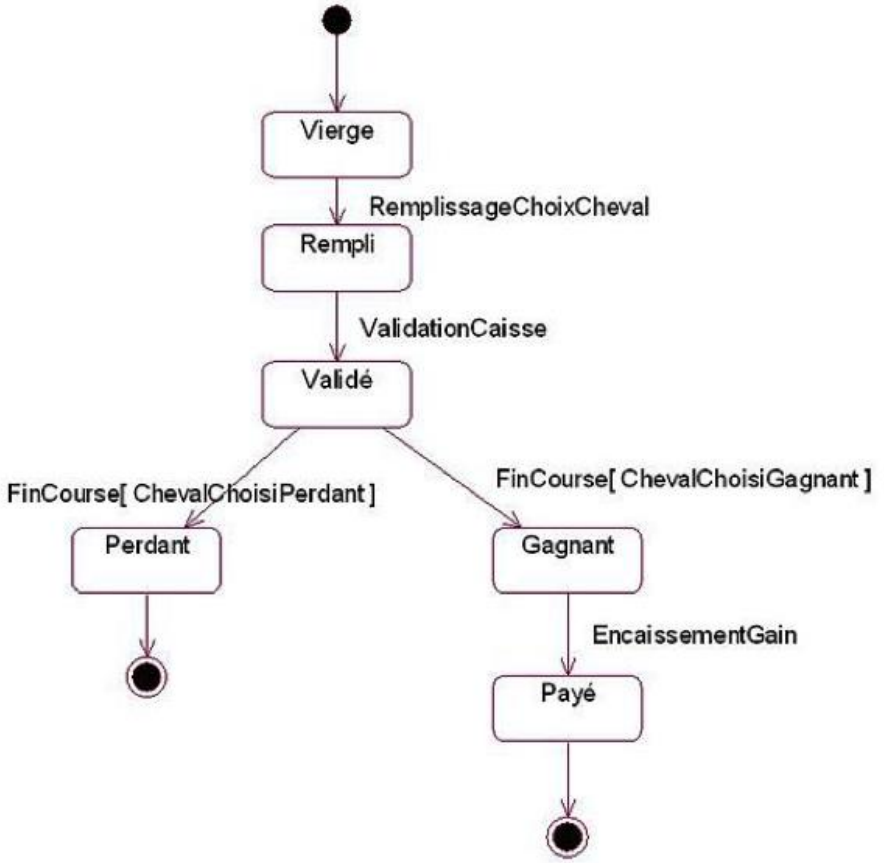


- Le diagramme doit être un automate fini et déterministe
  - 1 seul état initial
  - Dans un état donné, un n'y a qu'une seule transition possible pour le même évènement



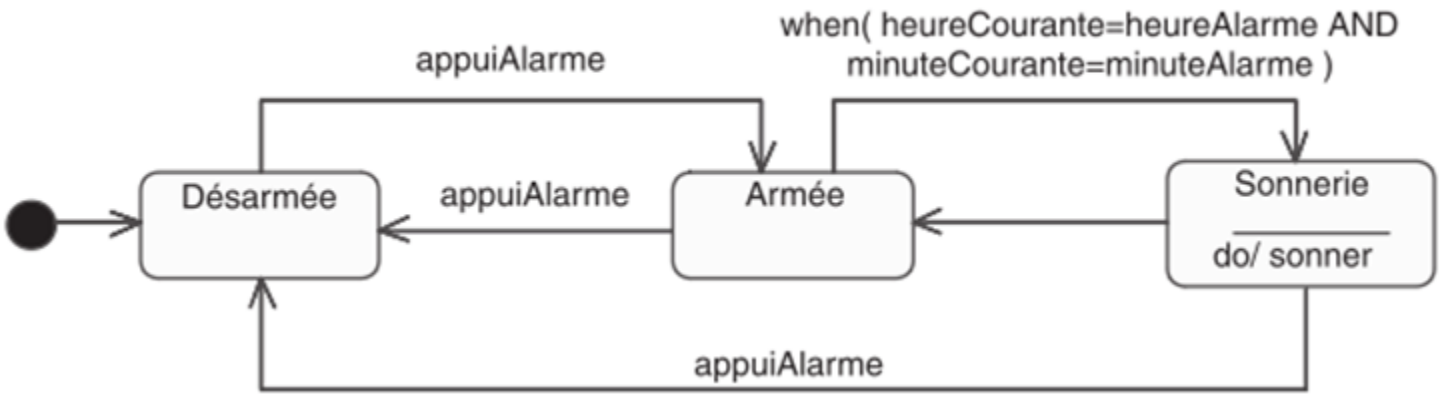
Exemple

- Diagramme d'état transition pour un ticket de course du tiercé



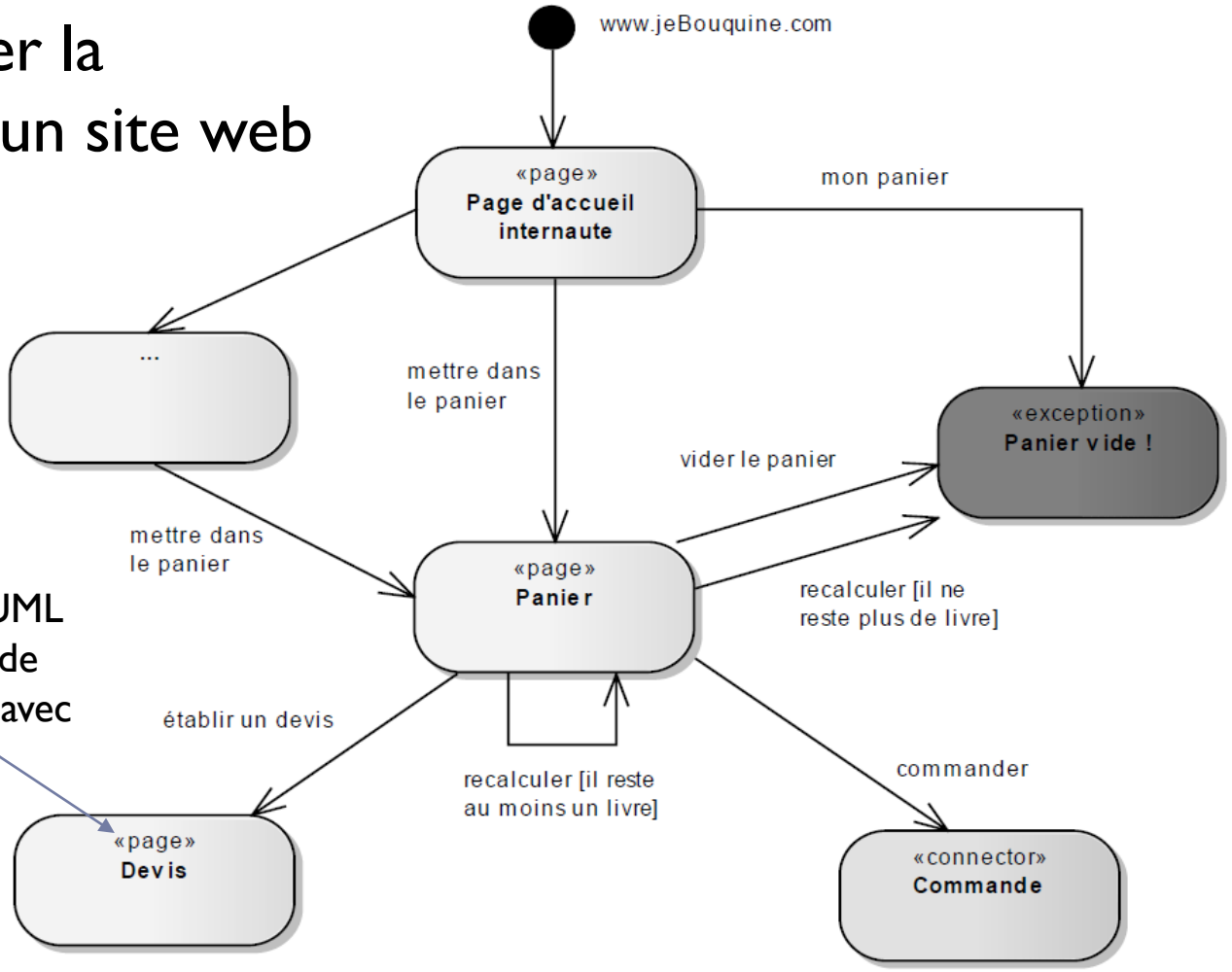
Exemple

- Diagramme d'état transition d'un réveil



Exemple

- Diagramme d'état transition pour représenter la navigation dans un site web



Remarque : ce diagramme contient des **stéréotypes** (« page », « connector », « exception »). Il s'agit d'un mécanisme d'extensibilité d'UML qui permet aux concepteurs de créer de nouveaux éléments avec des propriétés spécifiques

4. Modéliser les fonctionnalités avec UML

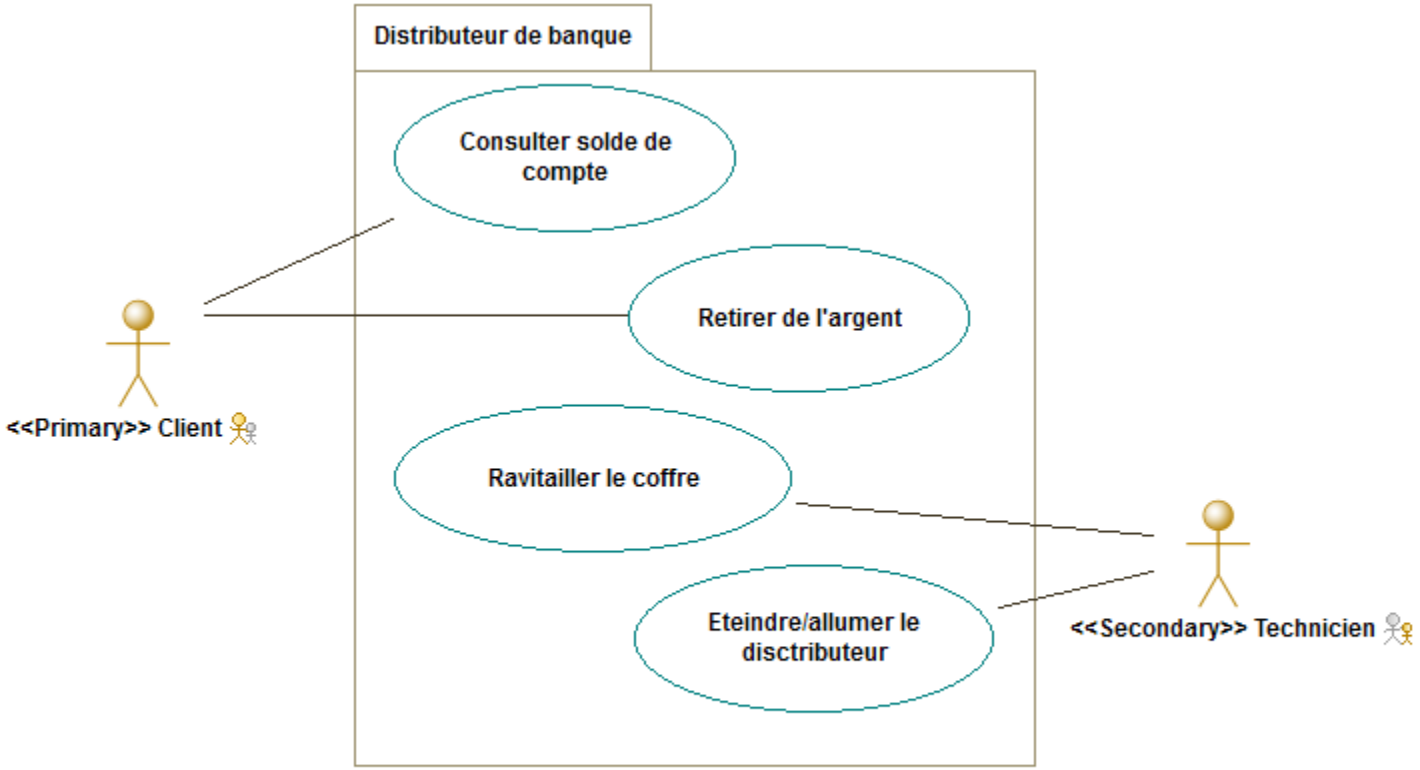
Diagramme de cas d'utilisation

Introduction au diagramme de cas

- Éléments représentés
  - Cas d'utilisation possible du système
  - Acteurs externes qui interagissent avec le système (personne ou autre système)
  - Relations entre acteurs et cas d'utilisation

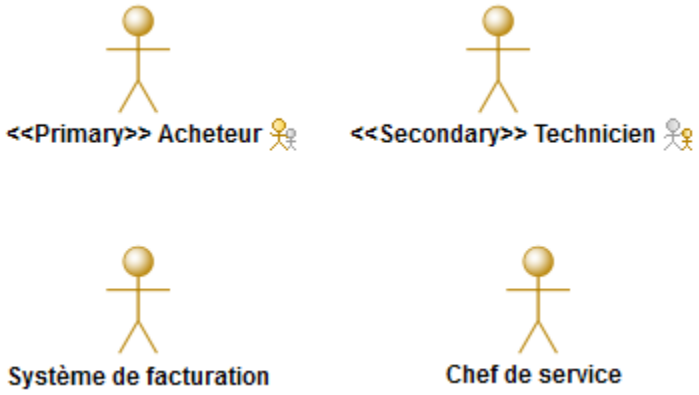
- Utilisation
  - Pendant l'étude des besoins (niveau 1)
    - Identifier ce que le client veut faire avec l'application
    - Comme point d'entrée pour la planification du développement
    - Comme index pour la documentation
  - Pendant la conception niveau abstrait (niveau 2)
    - Préciser comment les acteurs externes interagissent avec le système sans rentrer dans les détails liés au langage de développement ou de la plateforme
  - Pendant la conception niveau physique (niveau 3)
    - Préciser comment les acteurs externes choisis pour de développement (serveur PHP, base de données MYSQL…) interagissent avec le système

Exemple


Distributeur de banque
Consulter solde de compte
Retirer de l'argent
Ravitailler le coffre
Eteindre/allumer le disctributeur
<<Primary>> Client
<<Secondary>> Technicien

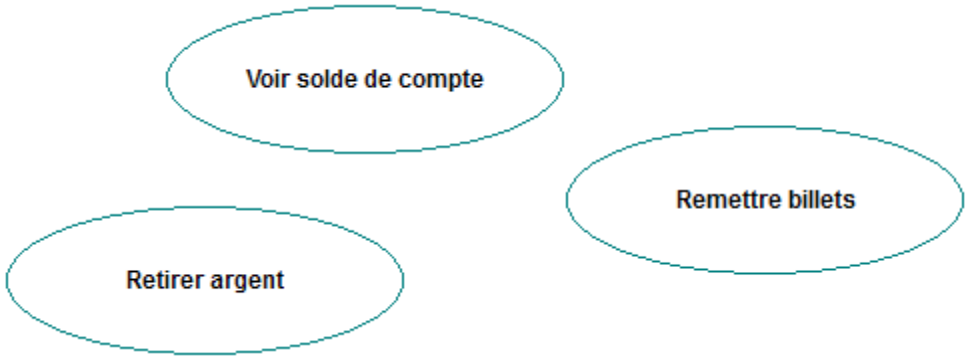
Acteur

- Acteurs externes qui interagissent avec le système pour atteindre un but
  - Acteurs humains ou d'autres systèmes
  - 2 catégories :
    - Acteurs principaux qui utilisent le système
    - Acteurs secondaires qui fournissent un service au système



Cas d'utilisation

- Format de description d'un cas d'utilisation
  - Verbe à l'infinitif + Groupe nominal
  - Entouré d'une ellipse

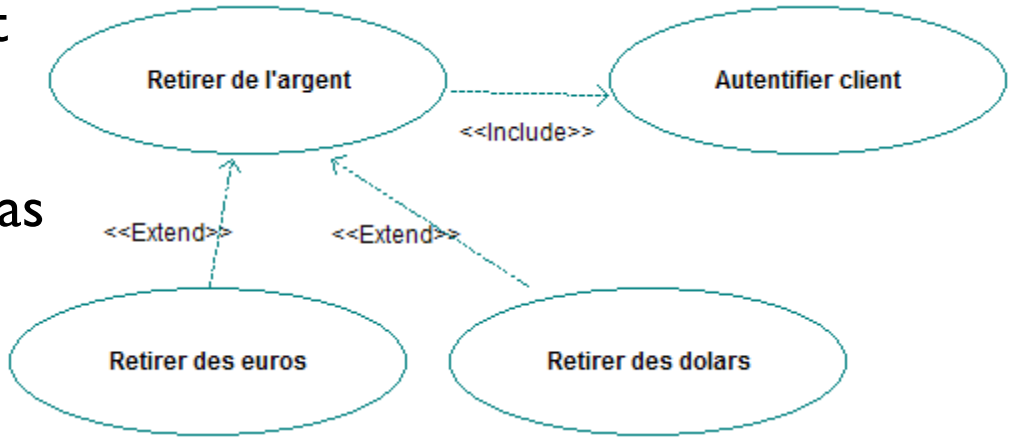


Relation

- Association entre un acteur et ces cas d'utilisation (utilisations qu'il peut faire du système).

Utilisation déconseillée, mais à connaitre quand même

- Utilisation <<include>> : le cas d'utilisation source contient aussi le comportement décrit dans le cas d'utilisation destination.
- Extension <<extend >> : le cas d'utilisation source étend (précise) les objectifs (comportements) du cas d'utilisation destination.

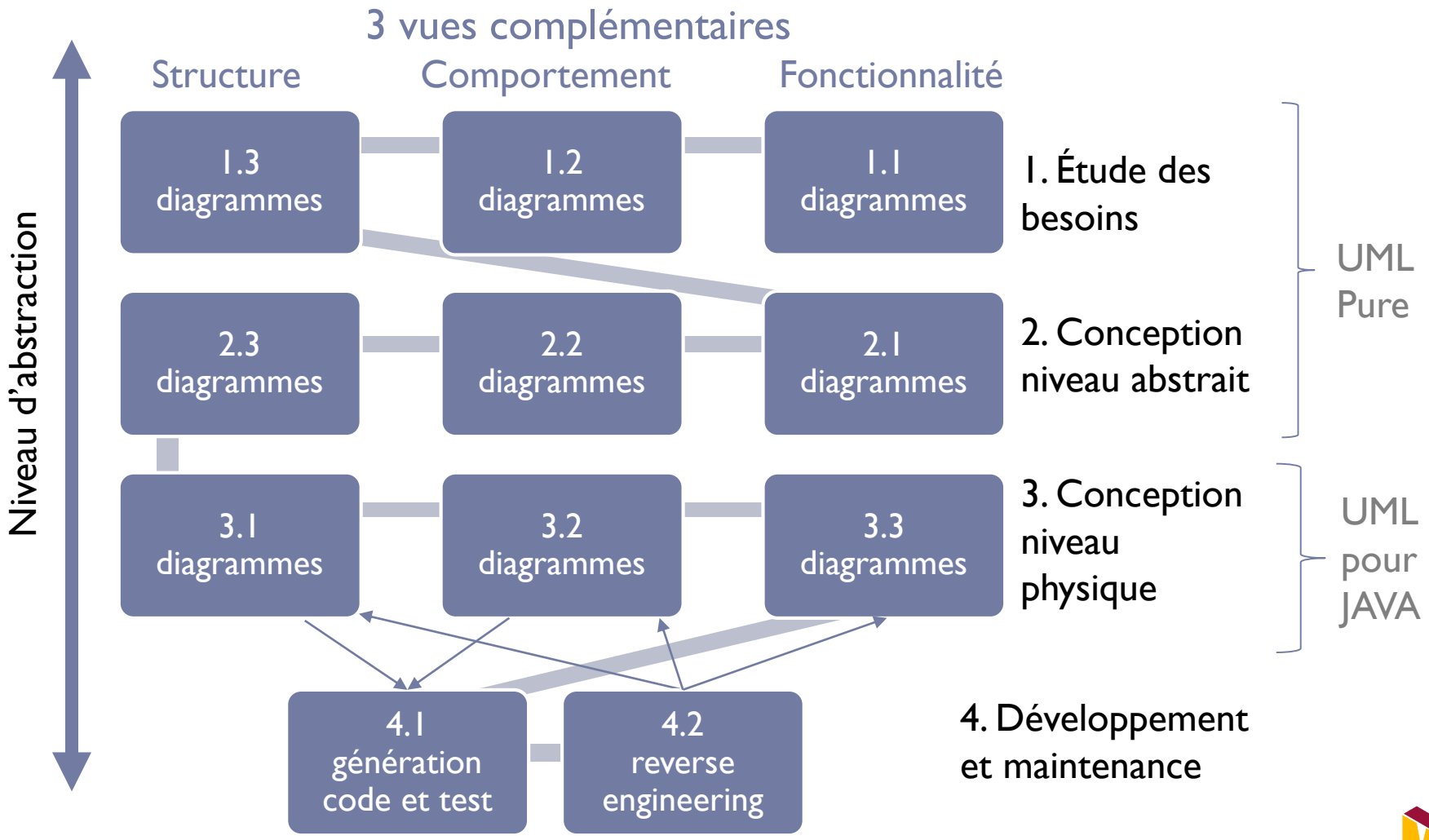


5. Développer avec UML

11 étapes à suivre 1/2

1. Analyse des besoins
   1. Cas d'utilisation
   2. Interactions
   3. Classes
2. Conception niveau abstrait
   1. Cas d'utilisation
   2. Interactions
   3. Classes
3. Conception niveau physique
   1. Classes
   2. Interactions
   3. Cas d'utilisation
4. Développement et maintenance
   1. Génération de code et tests
   2. Modification de l'application

11 étapes à suivre 2/2



1. Analyse des besoins

- Objectif : modéliser les fonctionnalités générales et l'environnement de l'application afin de comprendre les besoins du client.
- Comment : à l'aide de **diagrammes en UML pure** d'un niveau très abstrait contenant uniquement les informations les plus importantes.
- Sous-étapes :
  - 1.1. Cas d'utilisation
  - 1.2. Interaction
  - 1.3. Classes

1.1. Analyse des besoins – cas d'utilisation

- Objectif : modéliser les fonctionnalités et l'environnement de l'application afin de comprendre les besoins du client.
- Comment : à l'aide du client, modéliser les fonctionnalités offertes par l'application et les acteurs (personnes ou d'autre système) qui vont interagir avec à l'aide d'un **diagramme de cas d'utilisation**. Rappelons qu'il est déconseillé d'utiliser des relations <<extends>> et <<includes>>.

Exemple diagramme 1.1

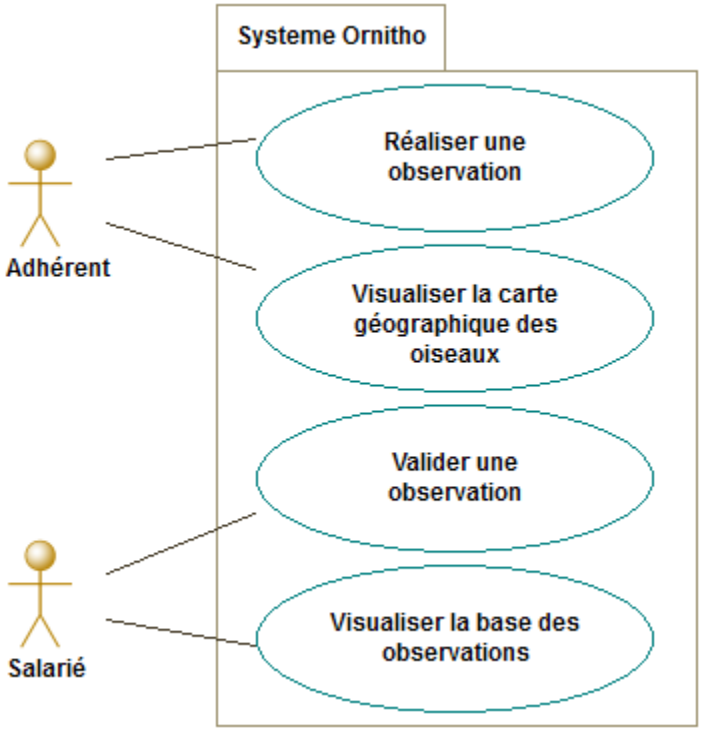
D'après le cahier des charges, le système Ornito doit permettre aux adhérents de :

- réaliser une observation
- visualiser la carte des oiseaux

Le système doit également permettre au salarié de l'association de :

- valider une observation réalisée par un adhérent
- visualiser la base des observations

Ce diagramme fait apparaitre ces 4 fonctionnalités principales et les 2 acteurs principaux qui vont interagir avec le système.



1.2. Analyse des besoins – interactions

- Objectif : modéliser les scénarios d'utilisation type de façon non ambigüe.
- Comment : pour chaque cas d'utilisation identifier à l'étape 1. 1, modéliser les scénarios d'interaction possibles entre l'acteur et le système à l'aide d'un **diagramme de séquence en UML pure**. Pensez à modéliser des scénarios pour lesquels il y aura des erreurs. Réutilisez la même nomenclature pour les éléments et les objets afin de garder la cohérence avec le diagramme de cas.
- Remarque : cette étape peut être faite en même temps de l'étape 1.3 d'analyse des besoins au niveau classe.

Exemple diagramme 1.2

4 cas d'utilisation dans le diagramme de l'étape1.1 => au moins 4 diagrammes de séquence pour décrire le scénario des interactions entre les acteurs et le système

2 Diagrammes pour le cas d'utilisation « réaliser une observation » : 1 sans erreur et 1 avec erreur d'interaction

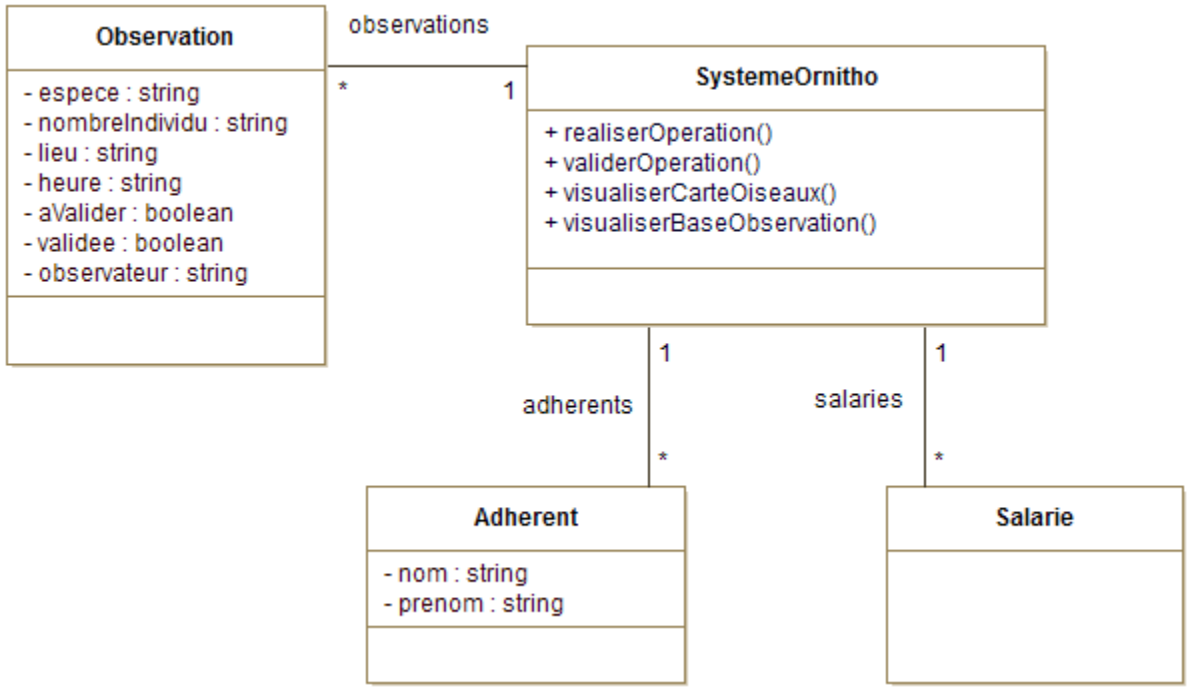
1.3. Analyse des besoins – classes

- Objectif : modéliser les classes représentant les données spécifiées dans le cahier des charges.
- Comment : modéliser toutes les classes nécessaires et leurs associations à l'aide d'un **diagramme de classe en UML pure**. Il est conseillé de rester à un niveau très abstrait en spécifiant uniquement les attributs et les opérations importants et sans spécifier la navigabilité des relations.

Exemple diagramme 1.3

Le cahier des charges et les diagrammes précédant font apparaitre le besoin d'avoir au minimum les 4 classes suivantes : Observation, SystemeOrnito, Adhérent et Salarié.

Ce diagramme fait apparaitre les attributs, les opérations et les associations importants sans donner plus de détails.



2. Conception niveau abstrait

- Objectif : modéliser l'application avec plus de détail en identifiant ses différentes composantes.
- Comment : à l'aide de **diagrammes détaillés en UML pure** contenant toutes les informations nécessaires à la comprehension des fonctionnalités du système.
- Sous-étapes :
  - 2.1. Cas d'utilisation
  - 2.2. Interaction
  - 2.3. Classes

2.1. Conception niveau abstrait – cas d'utilisation

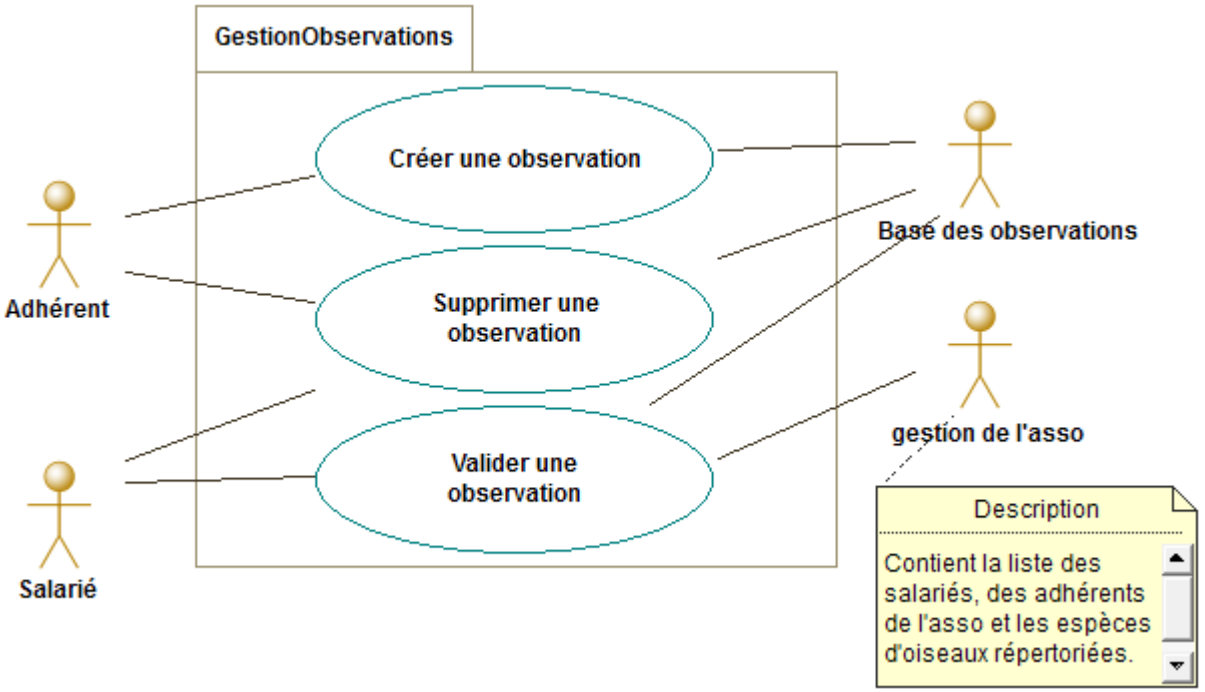
- Objectif : modéliser les fonctionnalités de chaque composant de l'application
- Comment : élaborez, pour chaque composant de l'application, le **diagramme de cas** d'utilisation correspondant. Spécifiez la relation entre ces diagrammes et les diagrammes de cas modéliser lors de la phase d'études des besoins.

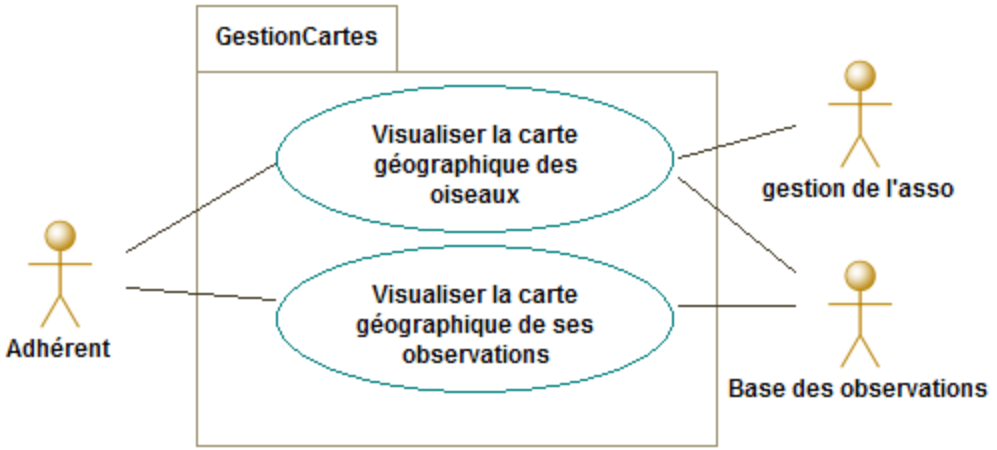
Exemple diagramme 2.1

Nous décomposons le système d'ornithologie en 2 grandes composantes :

- **GestionObservations**
- **GestionCartes**

Les diagrammes représentent les fonctionnalités de ces 2 composantes. Nous avons également fait apparaitre de nouveaux acteurs non humains nécessaires au système. Il est important de vérifier que les cas d'utilisations identifiés lors de l'étape 1.1 sont bien représentés.





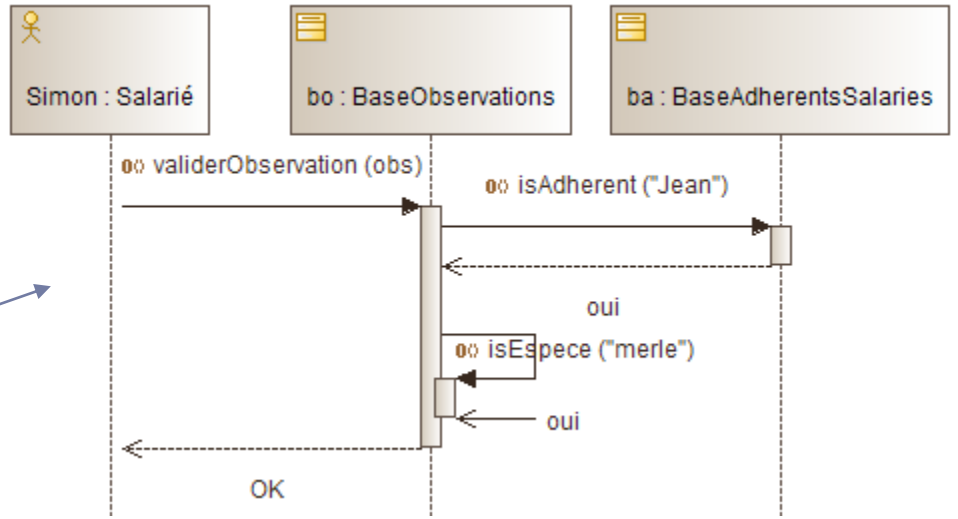
2.2. Conception niveau abstrait – interactions

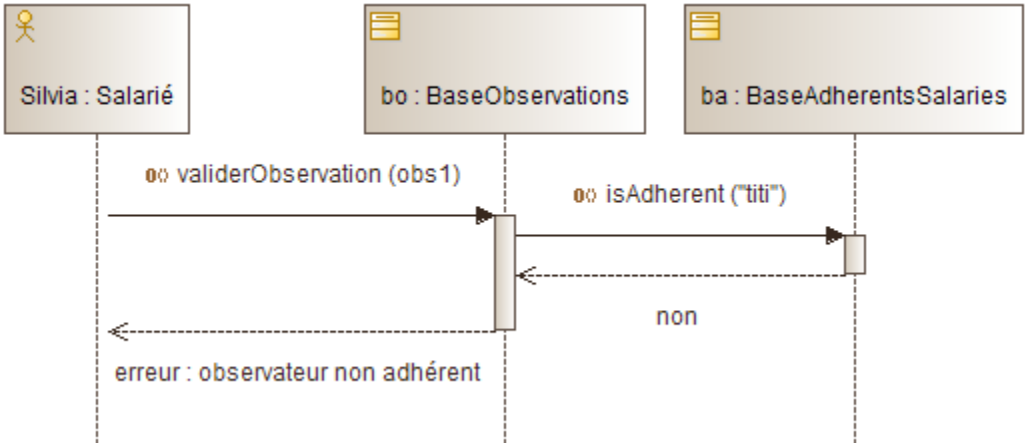
- Objectif : modéliser les scénarios pour les fonctionnalités de chacun des composants de l'application.
- Comment : à partir de la définition des composants de l'étape 2.1. élaborer, pour chacune de leurs fonctionnalités, au moins **2 diagrammes de séquence en UML pure** pour modéliser le cas où tout se passe comme prévu et le cas où il y a une erreur. Afin de maintenir la cohérence avec les autres diagrammes, pensez à typer les objets et utiliser les mêmes noms.
- Remarque : cette étape peut être faite en même temps que l'étape 2.1. de conception niveau abstrait des classes de l'application.

Exemple diagramme 2.2

5 cas d'utilisation dans les diagrammes de l'étape 2.1=> au moins 5 diagrammes de séquence pour décrire le scénario d'interactions entre les acteurs et le système

2 Diagrammes pour le cas d'utilisation « valider une observation » : 1 sans erreur et 1 avec.
Les noms des éléments correspondent aux noms choisis dans les diagrammes précédant.





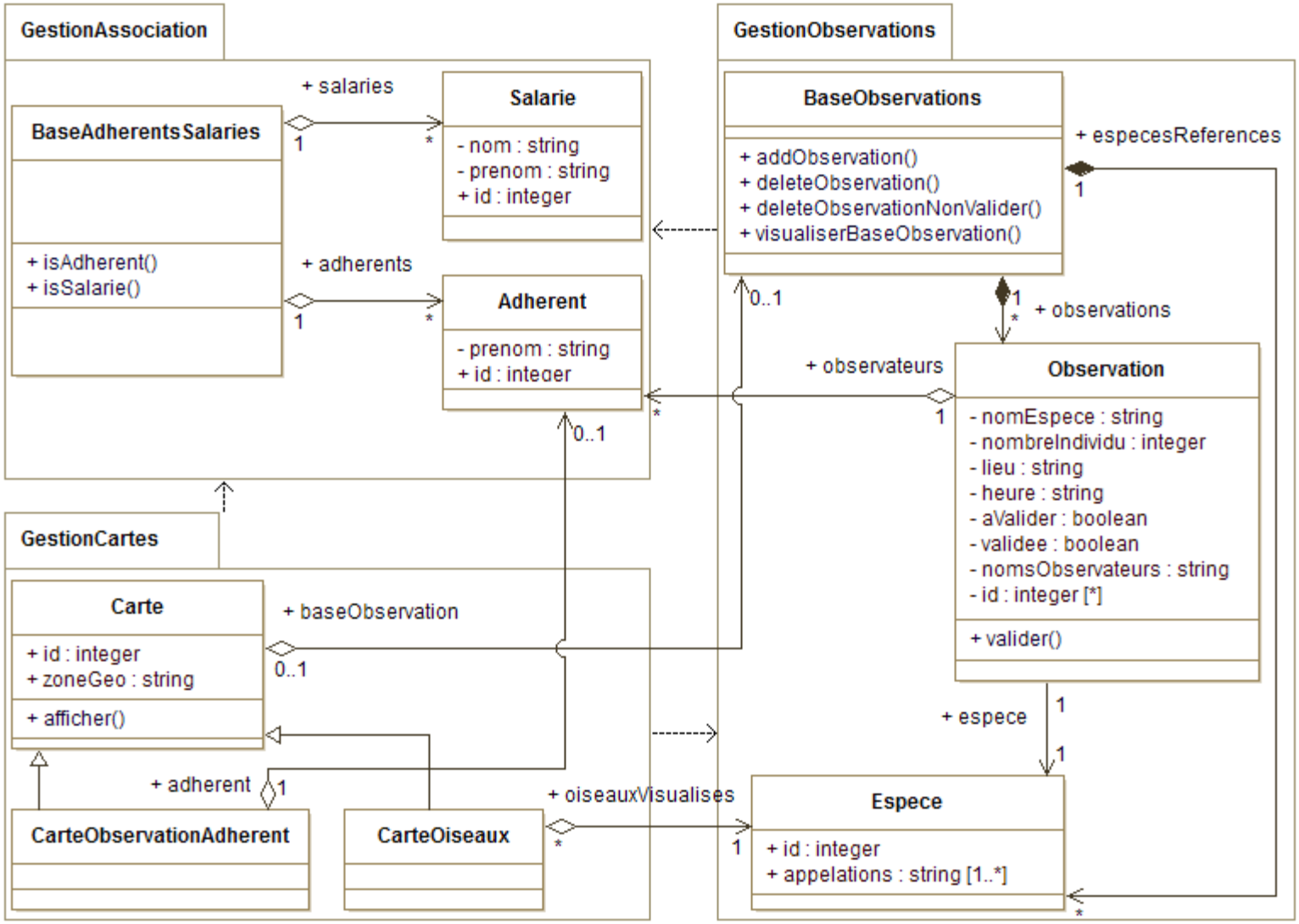
Afin de faciliter le traitement des fonctionnalités, nous décidons d'ajouter un troisième composant au système : **GestionAssociation** qui contient, entre autres, la classe BaseAdhérentsSalariés (voir étape suivante).

2.3. Conception niveau abstrait – classes

- Objectif : modéliser les classes des composants du système.
- Comment : modélisez les données manipulées par chaque composant et leur relation de dépendance avec un **diagramme de classe UML pure**. Chaque composant est représenté par un paquetage. Ces packages doivent être le plus indépendant possible et sans relations d'interdépendances.

Exemple diagramme 2.3

3 composants du système => 3 packages avec leurs classes détaillées



3. Conception niveau physique

- Objectif : modéliser l'application avec plus de détail possible et en tenant compte des spécificités techniques de la plateforme de développement pour pouvoir générer une partie du code automatiquement.
- Comment : à l'aide de **diagrammes en UML pour JAVA** contenant toutes les informations nécessaires à la génération de code automatique (classe et teste)
- Sous-étapes :
  - 3.1. Classes
  - 3.2. Interaction
  - 3.3. Cas d'utilisation

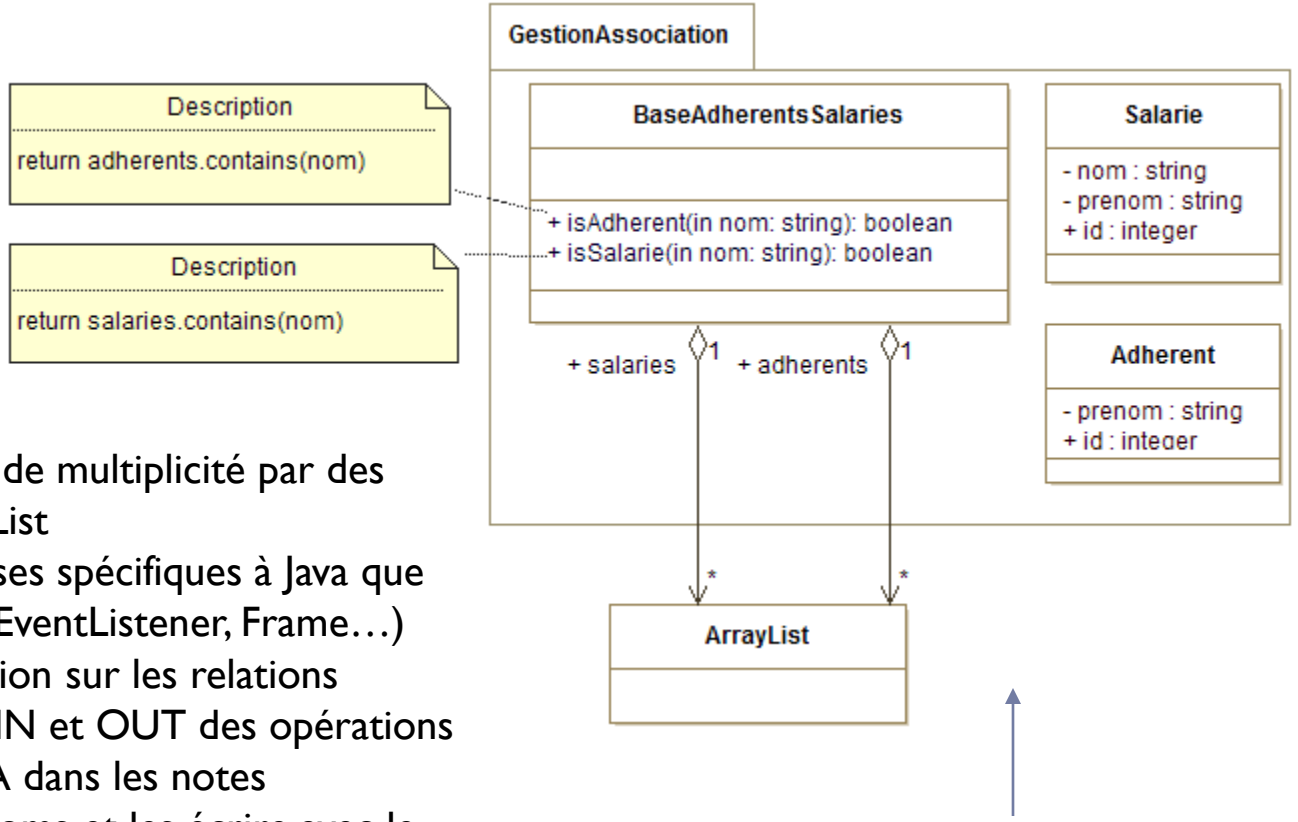
3.1. Conception niveau physique - classes

- Objectif : modéliser les classes des composants avec assez de détails afin de pouvoir faire de la génération de code automatique.
- Comment : Modéliser de façon très détaillée les classes de la plateforme d'exécution permettant la concrétisation du système en JAVA dans un **diagramme de classe en UML pour JAVA**. Ce diagramme peut être fait à partir du diagramme de classe réalisé à l'étape 2.3. Il suffit de le détailler et le compléter en respectant les spécificités de la plateforme de développement (JAVA, Jar). Il est aussi important de respecter toutes les contraintes qui permettent de faire de la génération de code automatique.

Exemple diagramme 3.1

Reprendre le diagramme de classe de l'étape précédente et la transformer pour qu'elle respecte toutes les règles de la génération de code :

- Modifier les associations de multiplicité par des liens avec la classe ArrayList
- Modéliser les autres classes spécifiques à Java que vous aller utiliser (Date, EventListener, Frame…)
- Mettre le sens de navigation sur les relations
- Spécifier les paramètres IN et OUT des opérations et insérer leur code JAVA dans les notes
- Enlever les accents des noms et les écrire avec le bon format (camel case avec 1ère lettre en majuscule pour les classes et les packages et en minuscule pour le reste)



Une petite partie du diagramme de classe ainsi transformé.

3.2. Conception niveau physique - interaction

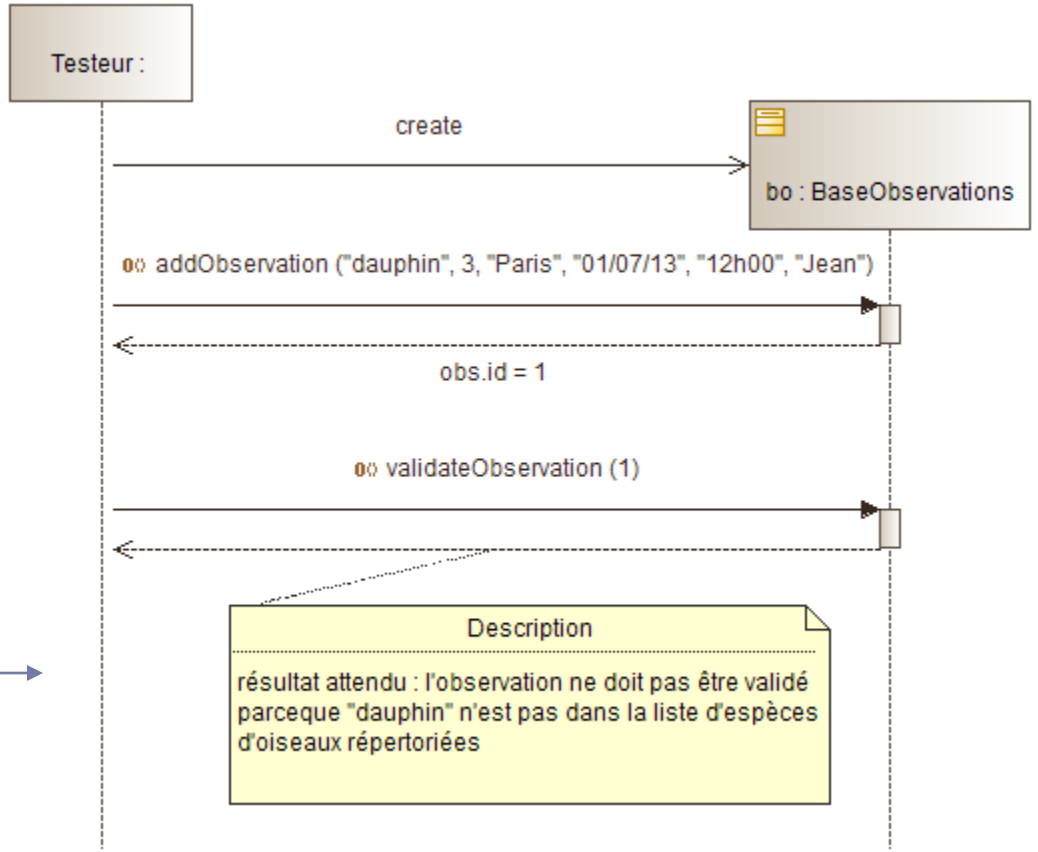
- Objectif : produire des cas de tests abstraits.
- Comment : pour chaque opération des classes, il faut modéliser plusieurs tests abstraits à l'aide de **diagramme de séquence de teste en UML pour Java.**

Exemple diagramme 3.2

8 opérations dans les classes du système à l'étape 3.1. => au moins 8 cas de teste pour définir le comportement que ces opérations doivent avoir.

Ces cas de teste pourront ensuite être utiliser pour valider le fonctionnement correct du système après le développement.

Un exemple de diagramme de séquence de teste pour l'opération addObservation() de la classe BaseObservation.



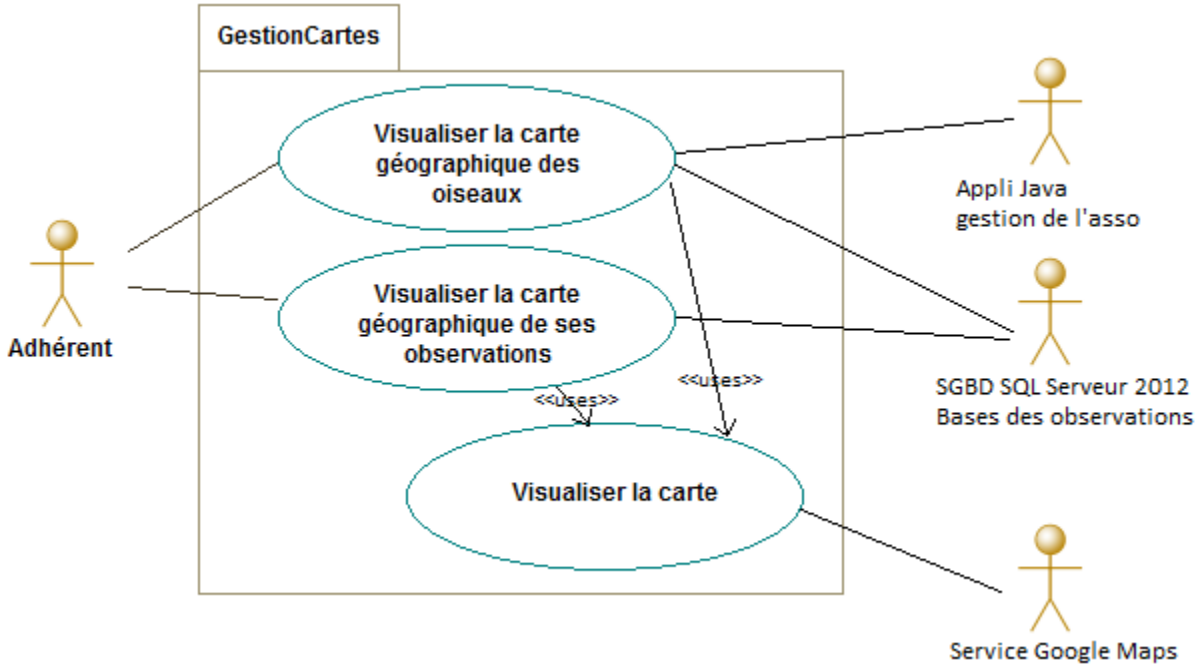
3.3. Conception niveau physique – cas d'utilisation

- Objectif : modéliser les fonctionnalités offertes par les composantes au niveau physique.
- Comment : Pour chaque package identifier lors de l'étape 3.1, modéliser les fonctionnalités offertes par la plateforme à l'aide d'un **diagramme de cas d'utilisation en UML pour JAVA**.
- Remarque : en le comparant avec le diagramme de classe de l'étape 2.3, ce diagramme permet de différencier les fonctionnalités directement réalisées par l'application de celles offertes par la plateforme d'exécution.

Exemple diagramme 3.3

Reprendre le diagramme de cas d'utilisation 2.1 mais en spécifiant la nature des acteurs externes choisis pour le développement et les services qu'ils fournissent.

Pour l'application Ornito, la visualisation des cartes est assurée par le service de Google Maps, l'accès à la base de donnée est faite par le SGBD SQLServeur 2012 et la gestion de l'association est assurée par une autre application développée en JAVA.



4. Développement et maintenance

- Objectif : modéliser l'application avec plus de détail possible et en tenant compte des spécificités techniques de la plateforme de développement pour pouvoir générer une partie du code automatiquement.
- Comment : à l'aide de **diagrammes en UML pour JAVA** contenant toutes les informations nécessaires à la génération de code automatique (classe et teste)
- Sous-étapes :
  - 4.1. Génération de code et tests
  - 4.2. Débugage et modification de l'application

4.1. Génération du code et des tests

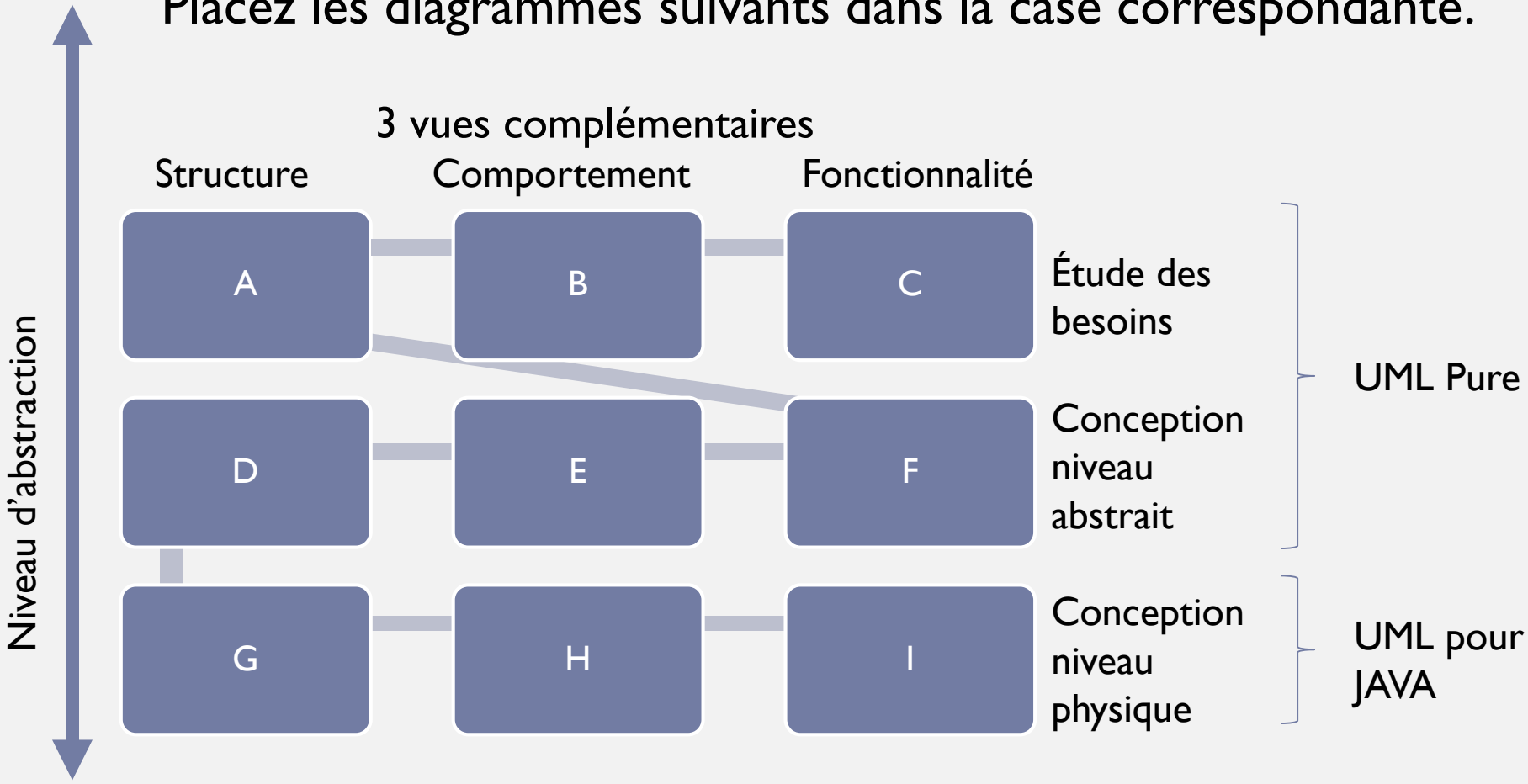
- Objectif : générer le code de l'application et celui des tests
- Comment : en utilisant les plugin de Java tels que Omondo, il est possible de générer automatiquement une partie du code et des tests de qualité à partir des diagrammes en UML pour JAVA.

4.2. Modification de l'application

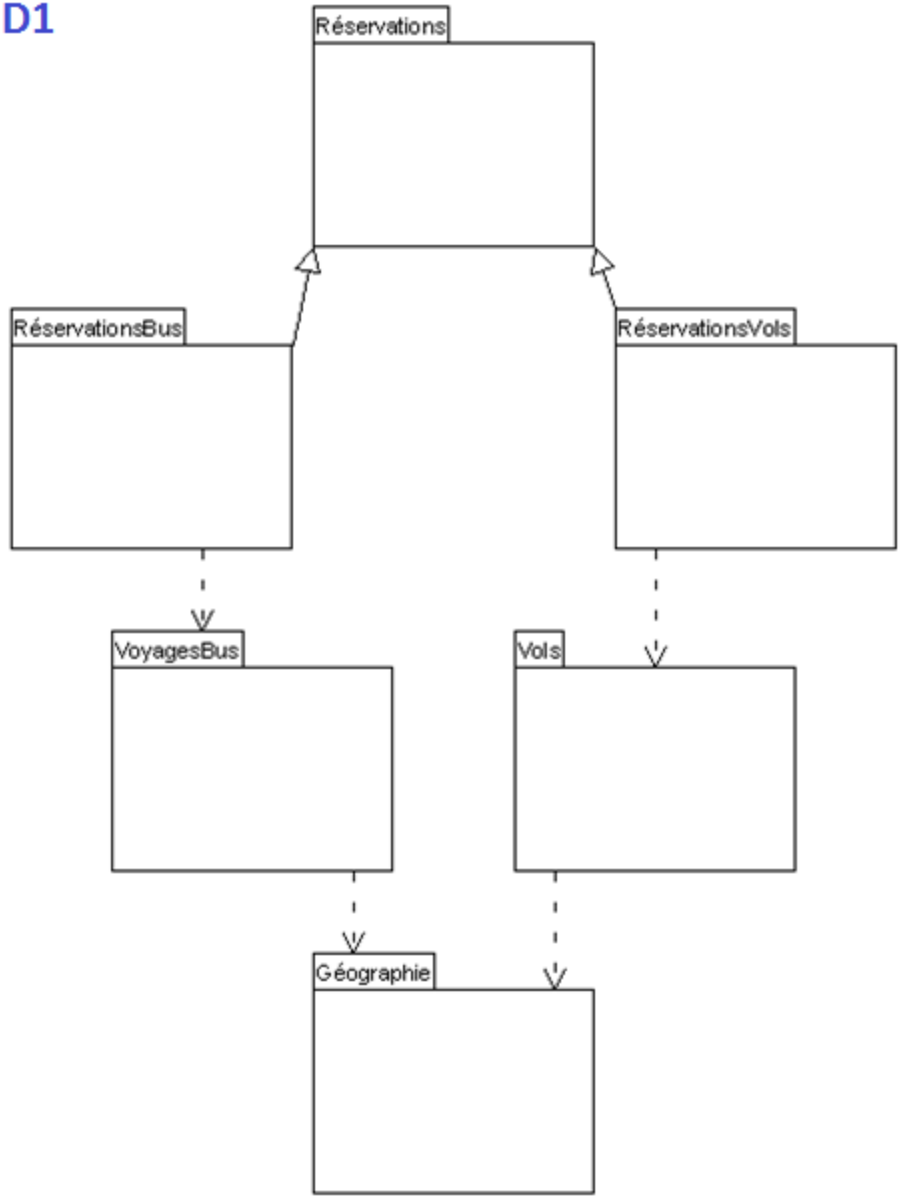
- Objectif : mettre à jour l'application
- Comment : après modification du modèle ou du code, il est nécessaire d'effectuer les opérations de génération de code et de reverse engineering afin d'assurer la synchronisation entre le code et le modèle.

Exercice

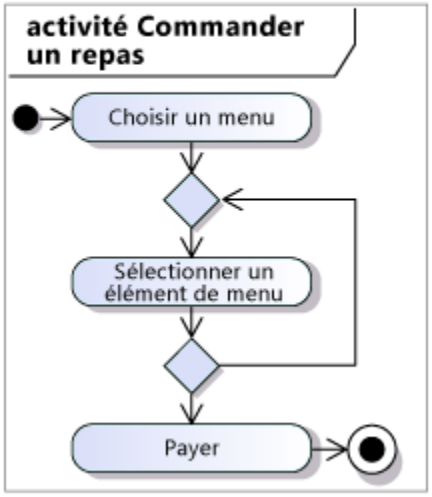
Placez les diagrammes suivants dans la case correspondante.



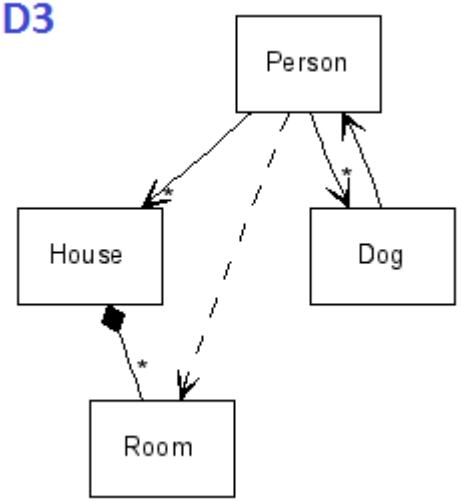
D1


Réservations
RéservationsBus
RéservationsVols
VoyagesBus
Vols
Géographie

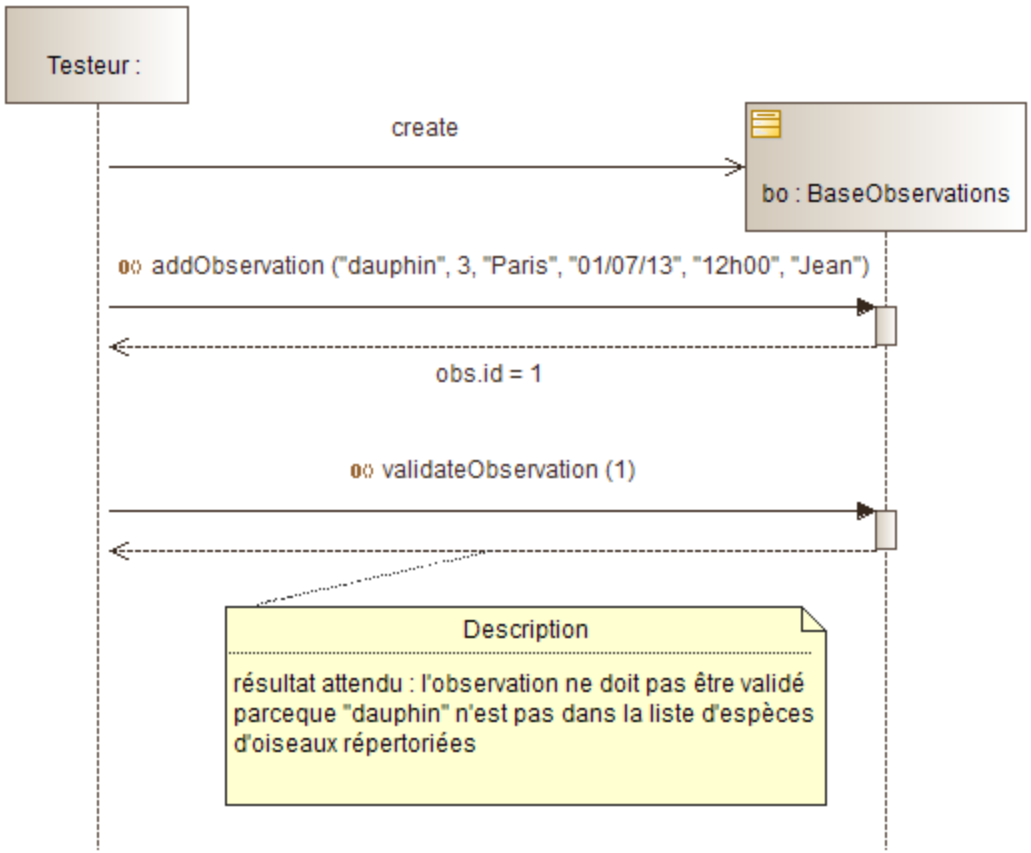
D2


activité Commander un repas
Choisir un menu
Sélectionner un élément de menu
Payer

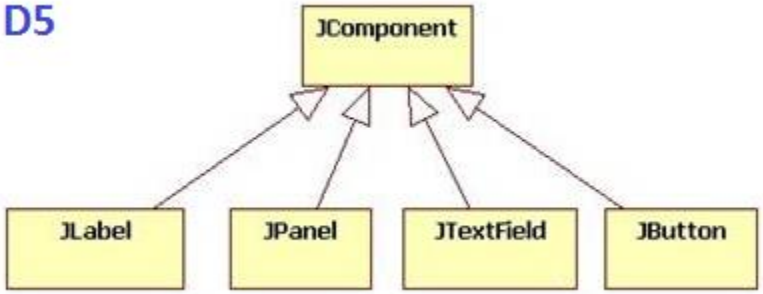
D3


Person
House
Dog
Room
*
*
*

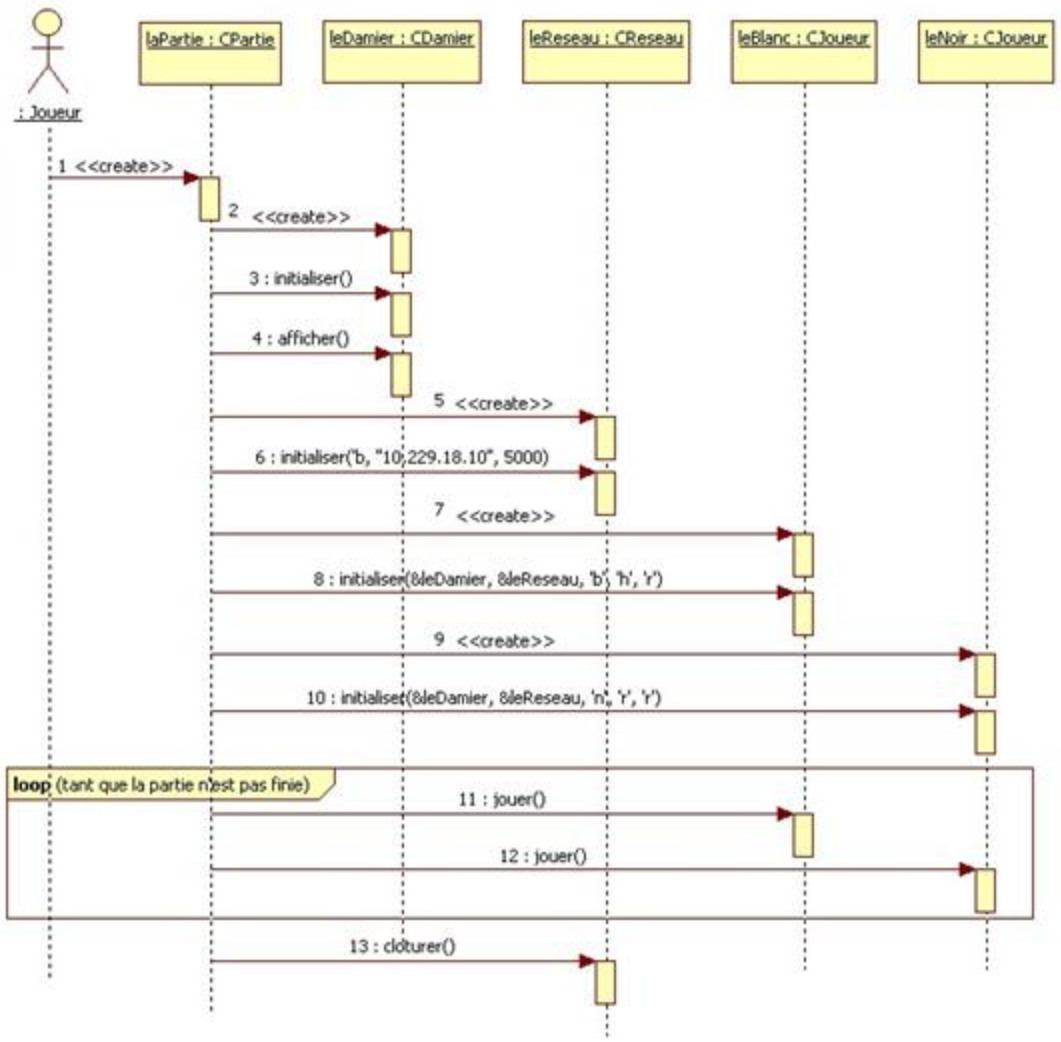
D4


Testeur :
create
bo : BaseObservations
addObservation ("dauphin", 3, "Paris", "01/07/13", "12h00", "Jean")
obs.id = 1
validateObservation (1)
Description
résultat attendu : l'observation ne doit pas être validé parceque "dauphin" n'est pas dans la liste d'espèces d'oiseaux répertoriées

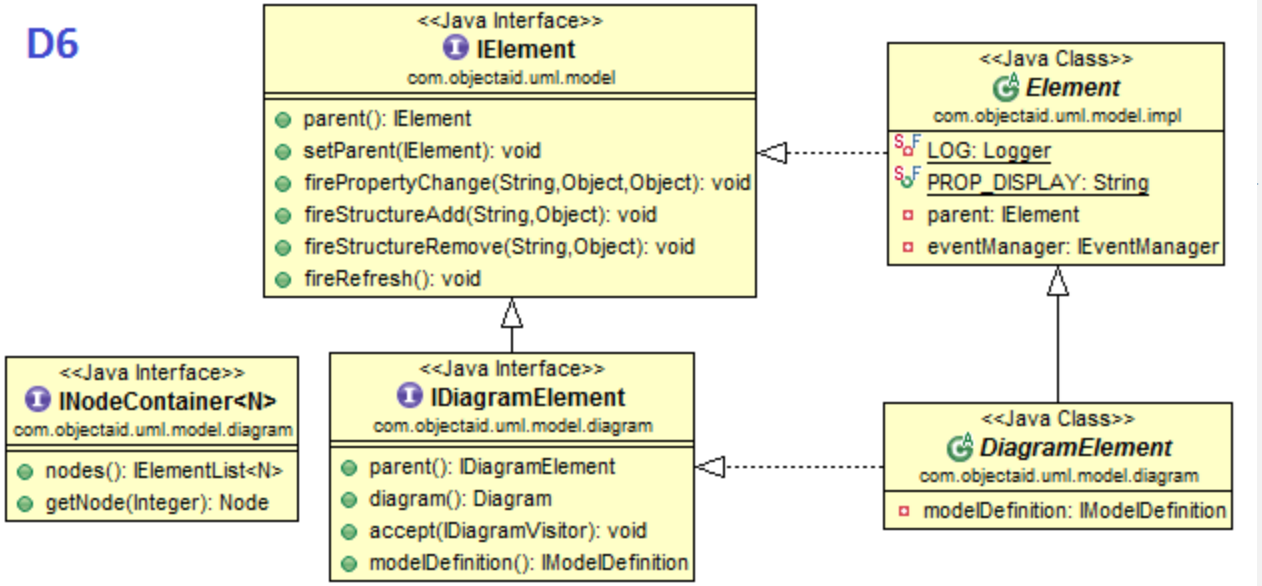
D5


JComponent
JLabel
JPanel
JTextField
JButton

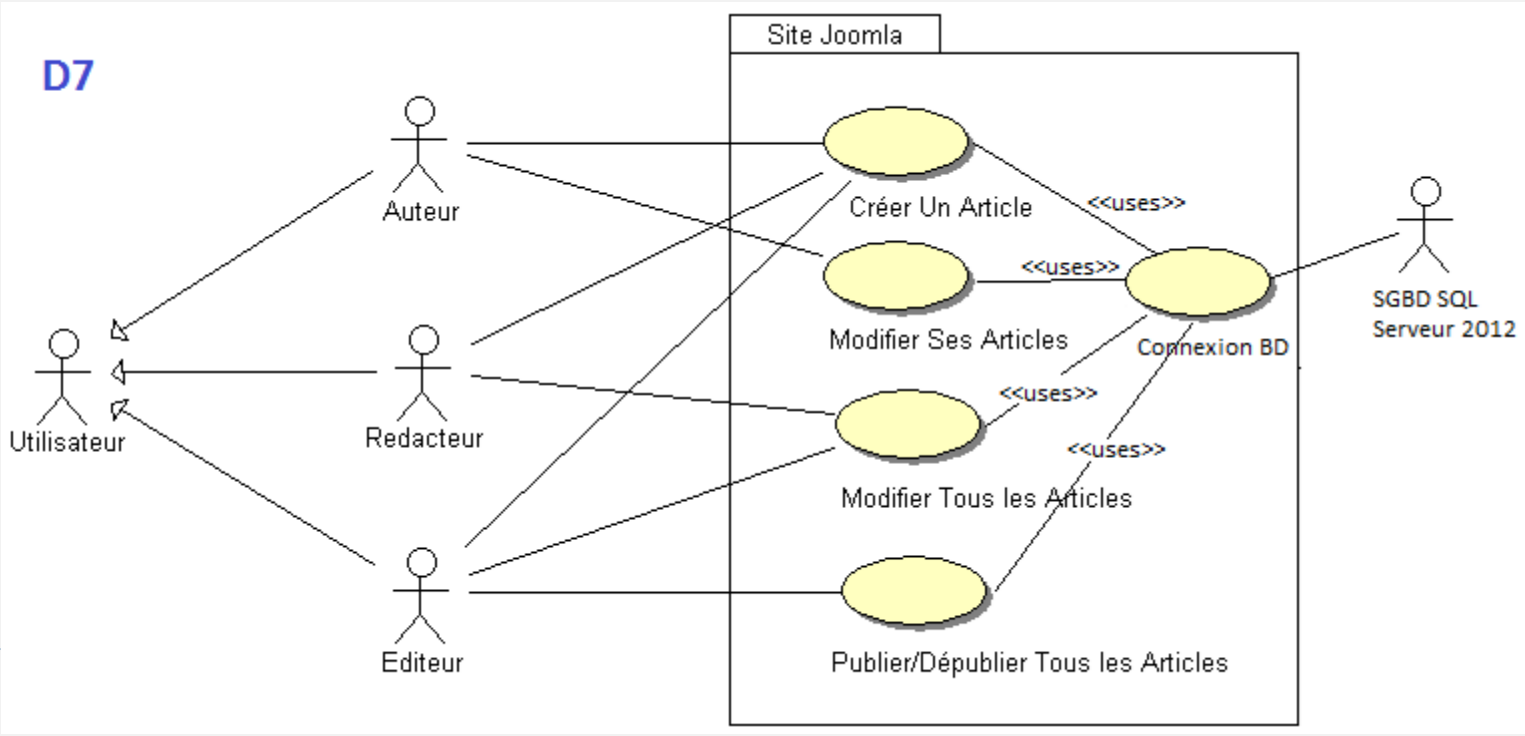
D6


: Joueur
laPartie : CPartie
leDamier : CDamier
leReseau : CReseau
leBlanc : CJoueur
leNoir : CJoueur
1 <<create>>
2 <<create>>
3 : initialiser()
4 : afficher()
5 <<create>>
6 : initialiser('b, "10.229.18.10", 5000)
7 <<create>>
8 : initialiser(&leDamier, &leReseau, 'b', 'h', 'r')
9 <<create>>
10 : initialiser(&leDamier, &leReseau, 'n', 'r', 'r')
loop (tant que la partie n'est pas finie)
11 : jouer()
12 : jouer()
13 : cloturer()

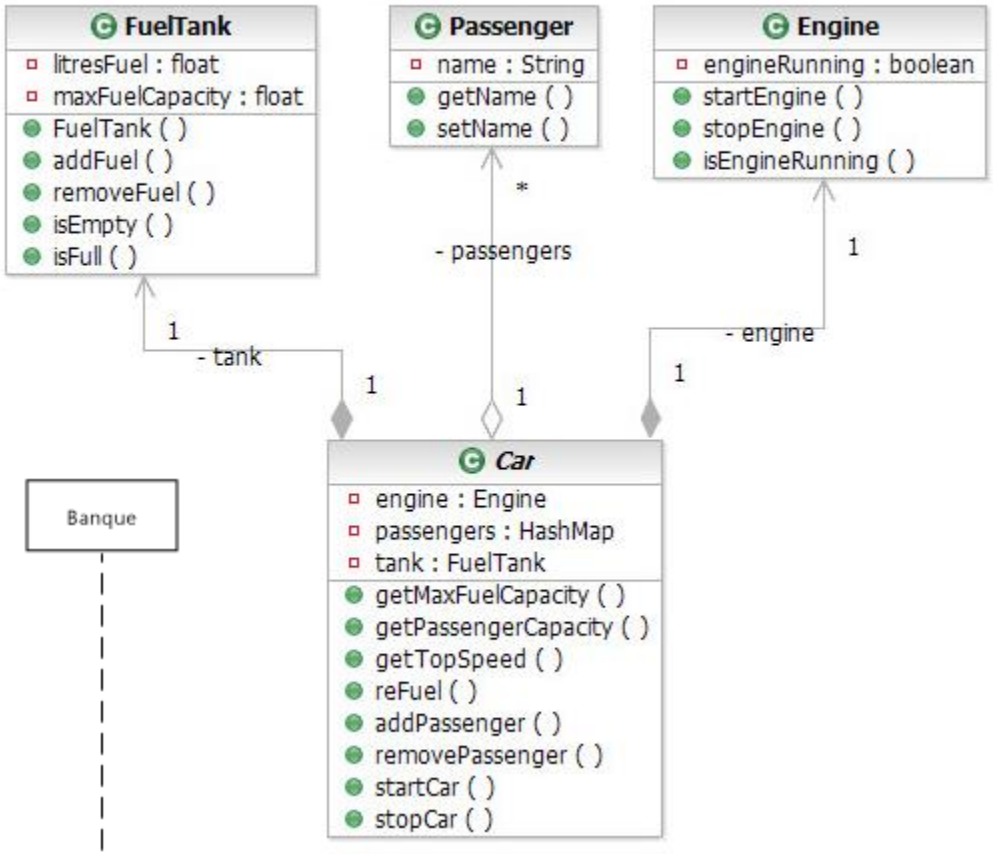
D6


<<Java Interface>>
IElement
com.objectaid.uml.model
parent(): IElement
setParent(IElement): void
firePropertyChange(String,Object,Object): void
fireStructureAdd(String,Object): void
fireStructureRemove(String,Object): void
fireRefresh(): void
<<Java Class>>
Element
com.objectaid.uml.model.impl
LOG: Logger
PROP_DISPLAY: String
parent: IElement
eventManager: IEventManager
<<Java Interface>>
INodeContainer<N>
com.objectaid.uml.model.diagram
nodes(): IElementList<N>
getNode(Integer): Node
<<Java Interface>>
IDiagramElement
com.objectaid.uml.model.diagram
parent(): IDiagramElement
diagram(): Diagram
accept(IDiagramVisitor): void
modelDefinition(): IModelDefinition
<<Java Class>>
DiagramElement
com.objectaid.uml.model.diagram
modelDefinition: IModelDefinition

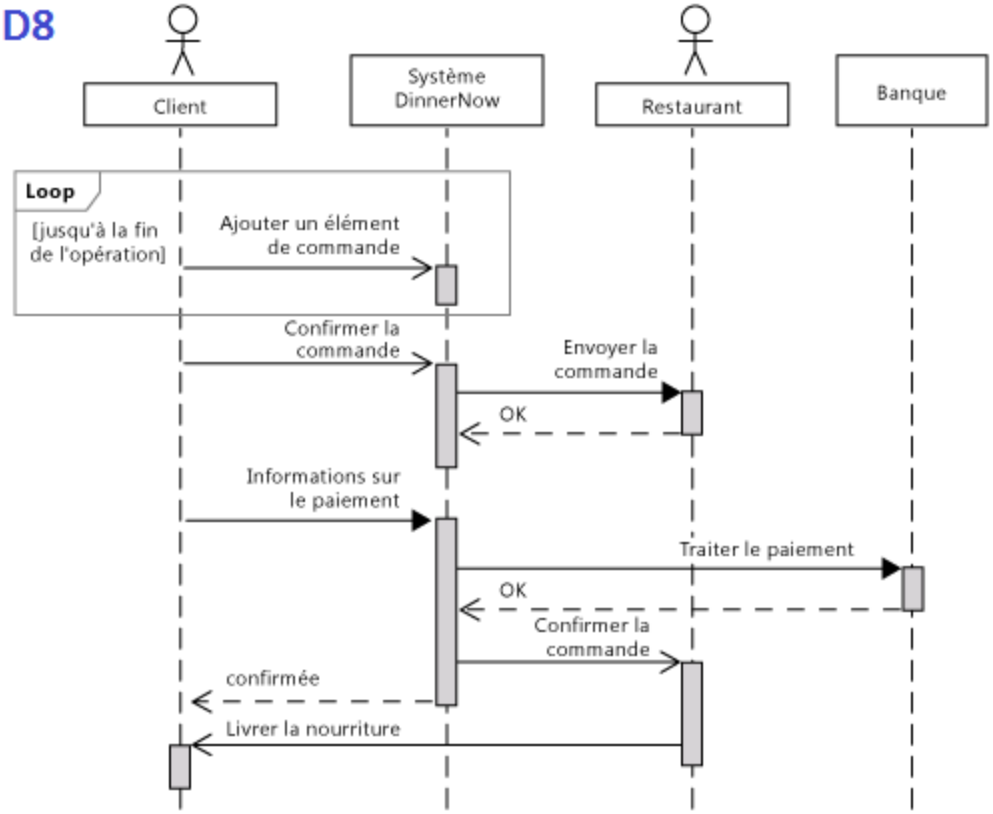
D7

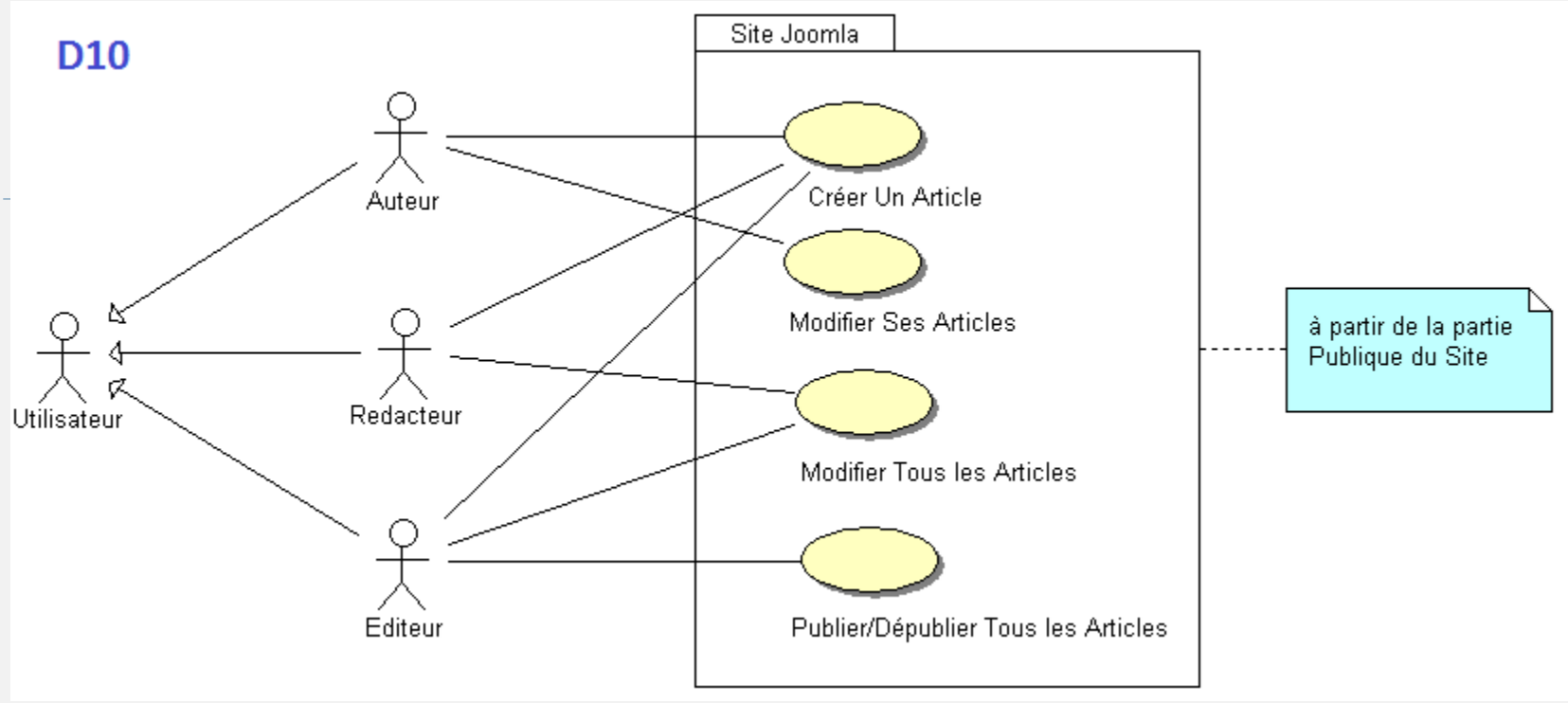

Site Joomla
Auteur
Utilisateur
Redacteur
Editeur
Créer Un Article
Modifier Ses Articles
Modifier Tous les Articles
Publier/Dépublier Tous les Articles
<<uses>>
<<uses>>
<<uses>>
<<uses>>
Connexion BD
SGBD SQL
Serveur 2012

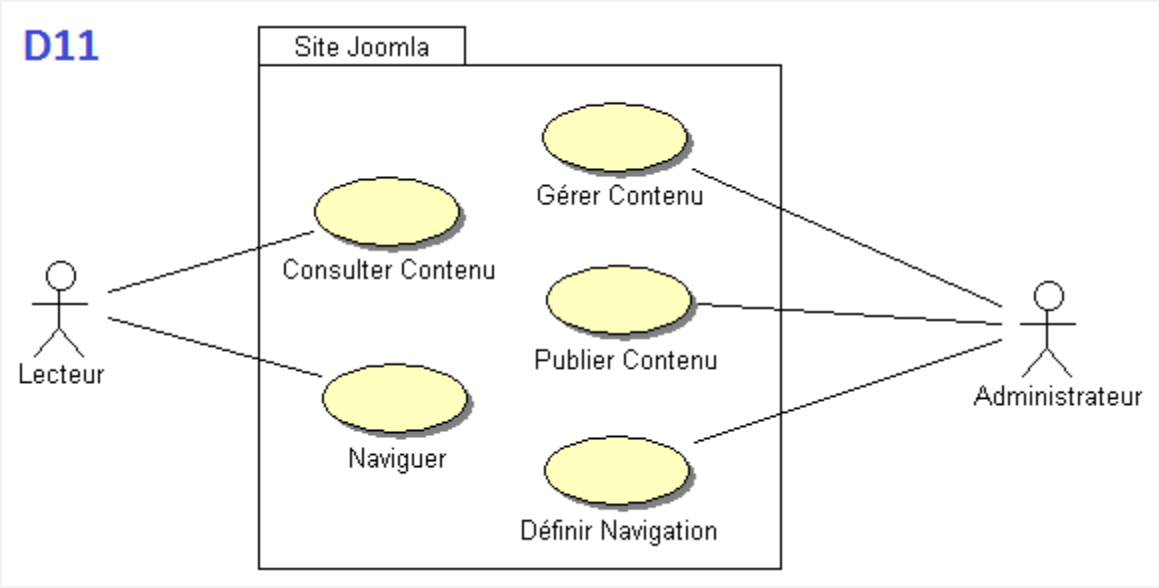
D9


FuelTank
litresFuel : float
maxFuelCapacity : float
FuelTank ( )
addFuel ( )
removeFuel ( )
isEmpty ( )
isFull ( )
Passenger
name : String
getName ( )
setName ( )
Engine
engineRunning : boolean
startEngine ( )
stopEngine ( )
isEngineRunning ( )
*
- passengers
1
1
- tank
1
1
- engine
1
Car
engine : Engine
passengers : HashMap
tank : FuelTank
getMaxFuelCapacity ( )
getPassengerCapacity ( )
getTopSpeed ( )
reFuel ( )
addPassenger ( )
removePassenger ( )
startCar ( )
stopCar ( )

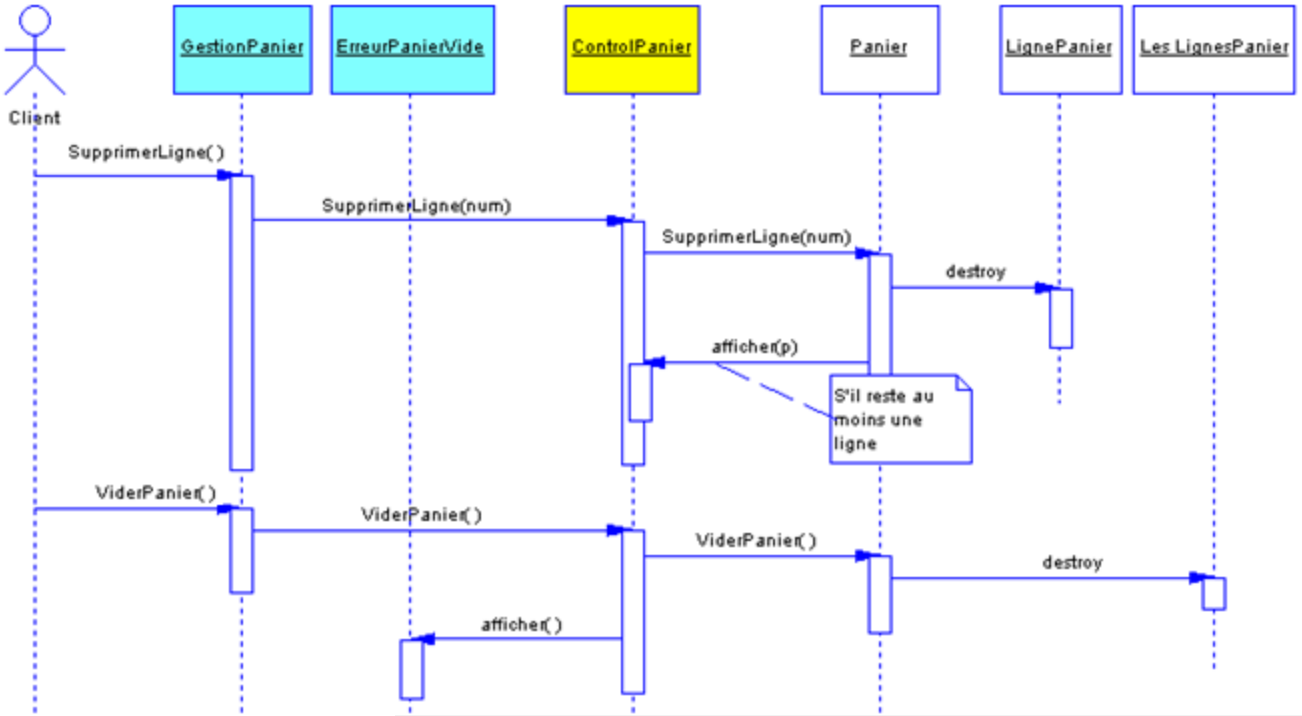
D8


Client
Système DinnerNow
Restaurant
Banque
Loop
[jusqu'à la fin de l'opération]
Ajouter un élément de commande
Confirmer la commande
Envoyer la commande
OK
Informations sur le paiement
Traiter le paiement
OK
Confirmer la commande
confirmée
Livrer la nourriture

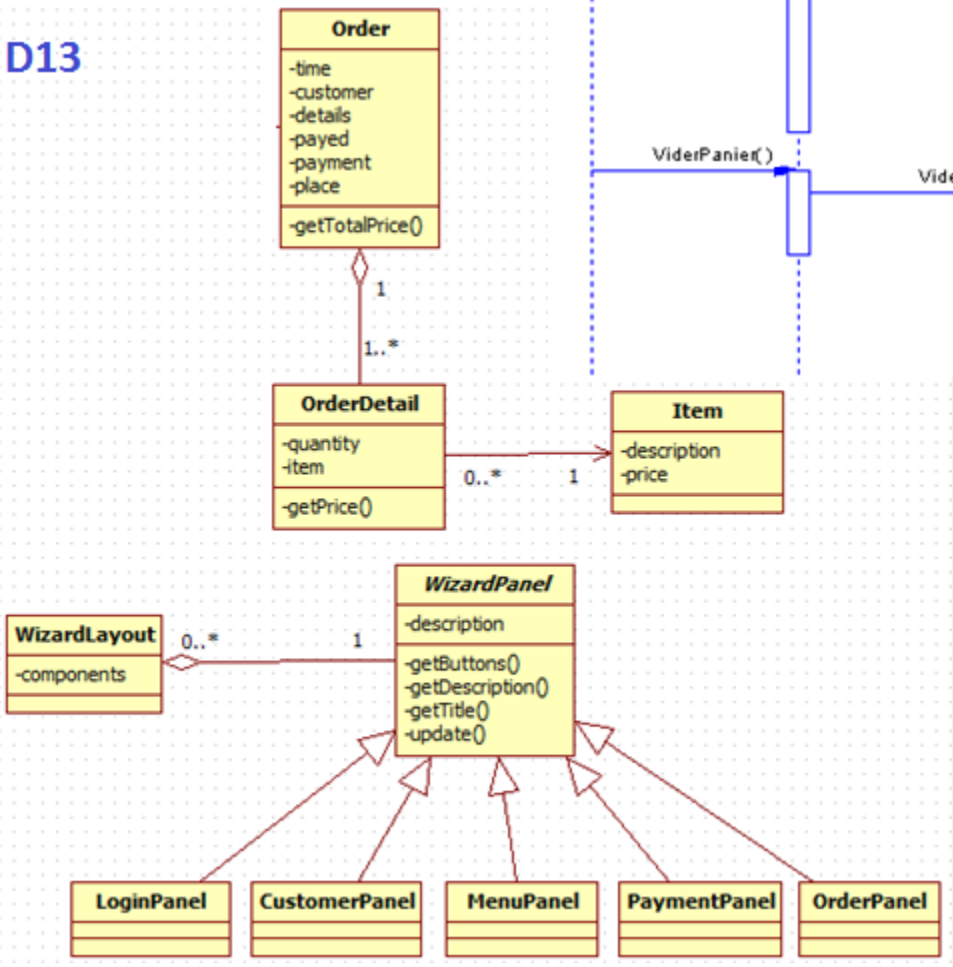

D10
Site Joomla
Auteur
Utilisateur
Redacteur
Editeur
Créer Un Article
Modifier Ses Articles
Modifier Tous les Articles
Publier/Dépublier Tous les Articles
à partir de la partie
Publique du Site


D11
Site Joomla
Lecteur
Consulter Contenu
Naviguer
Gérer Contenu
Publier Contenu
Définir Navigation
Administrateur

D12


Client
GestionPanier
ErreurPanierVide
ControlPanier
Panier
LignePanier
Les LignesPanier
SupprimerLigne()
SupprimerLigne(num)
SupprimerLigne(num)
destroy
afficher(p)
S'il reste au moins une ligne
ViderPanier()
ViderPanier()
ViderPanier()
destroy
afficher()

D13


Order
-time
-customer
-details
-payed
-payment
-place
-getTotalPrice()
1
1..*
OrderDetail
-quantity
-item
-getPrice()
0..*
1
Item
-description
-price
WizardPanel
-description
-getButtons()
-getDescription()
-getTitle()
-update()
WizardLayout
-components
0..*
1
LoginPanel
CustomerPanel
MenuPanel
PaymentPanel
OrderPanel