

Mon lutin d'UML

Table des matières

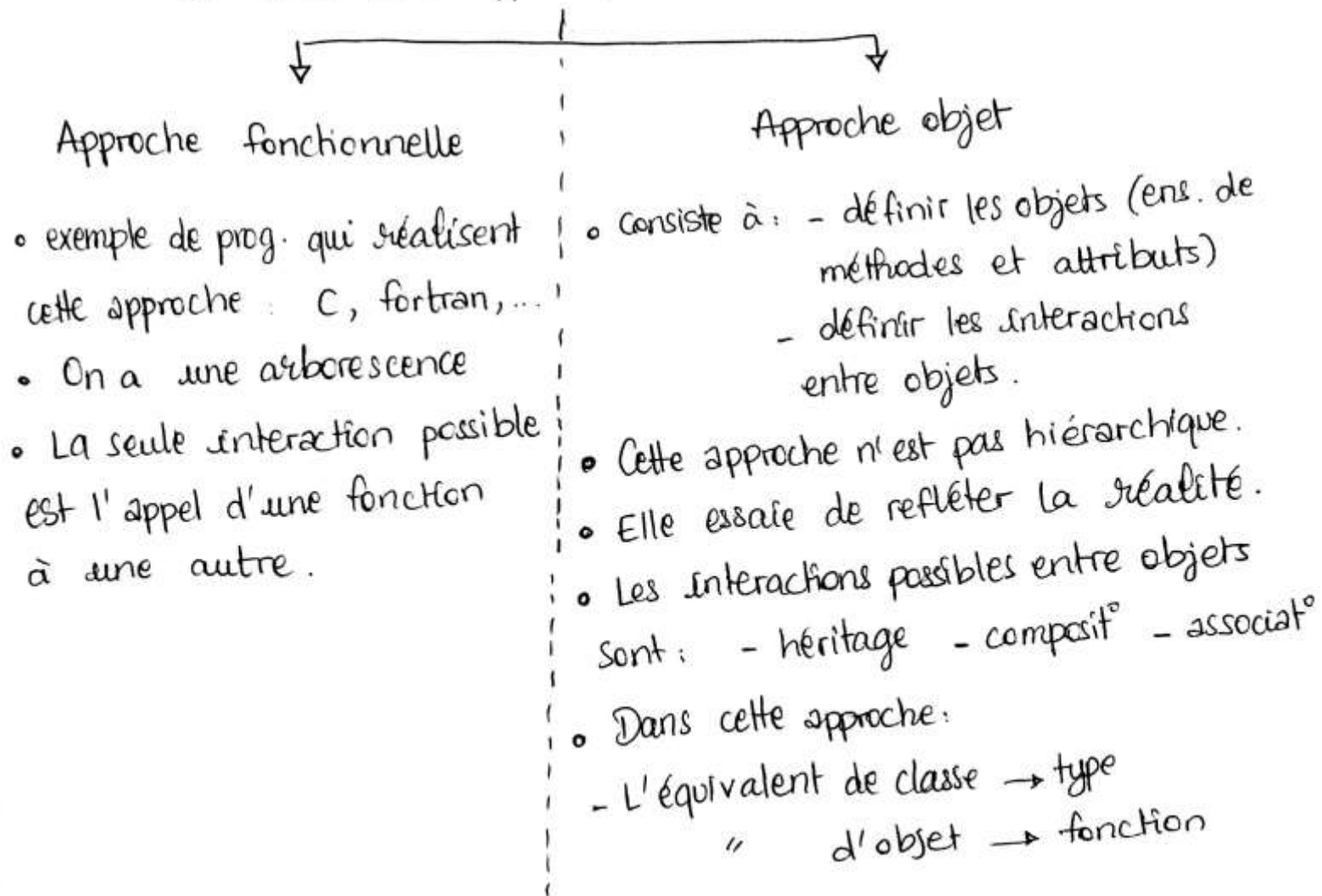
Notes du cours	4
Enoncé TD	29
Correction du TD	30

15/04/15

①

• On dit qu'une science s'appuie sur la philosophie de génie lorsque : Toute chose devrait être "solvée" en se basant sur la modélisation.

• Il existe deux approches :



• UML est un langage de modélisation

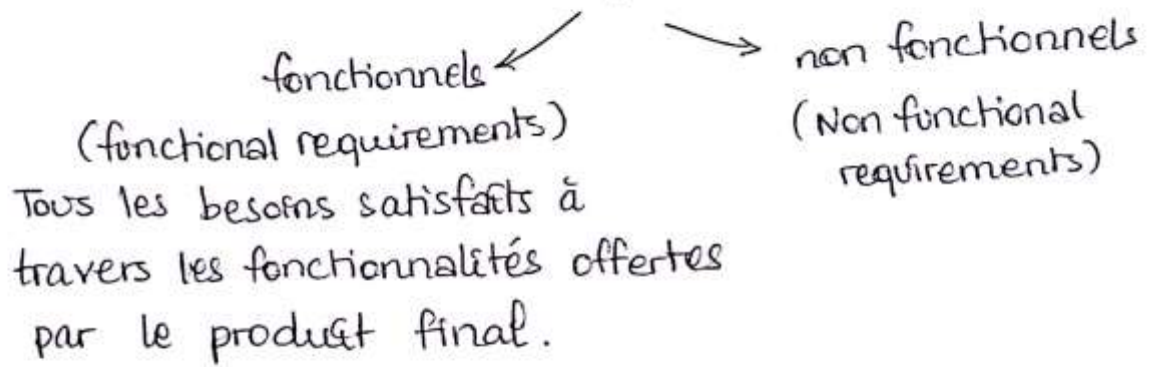
↓
ce n'est pas une méthode (car il n'y a pas de processus de développement standardisé).

* Diagrammes des cas d'utilisation

- Ce diagramme permet d'identifier et de décrire les utilisateurs du système (acteurs) et leur interaction avec le système.
- C'est l'ensemble de toutes les utilisations possibles.
- On exploite le fait que les utilisateurs n'ont pas le même point de vue d'utilisation.
- Quel en est l'intérêt? → déterminer les besoins fonctionnels du sys

→ Comment?

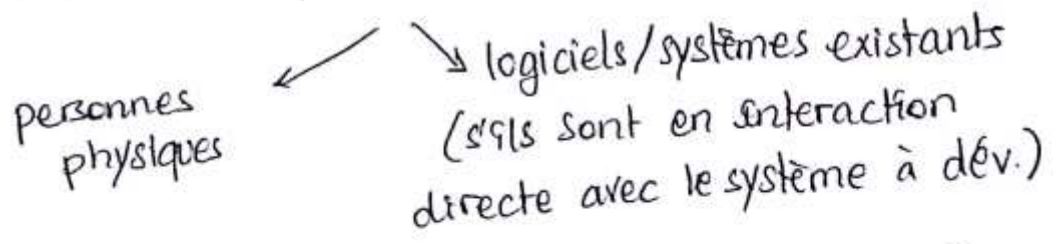
Cahier de charges → Ensemble de besoins spécifiés (requirements)



Donc Analyse des besoins ⇒ Diag. use case.

- La 1^{ère} chose à faire → Énumérer les acteurs.

- Acteurs ≡ utilisateurs → peuvent être



- Attention : Un acteur est un rôle (ce n'est pas une personne).

En d'autres termes: Un seul acteur peut jouer +¹⁵ rôles vis-à-vis de l'appli. Il sera donc représenté par un acteur selon le rôle qu'il joue.

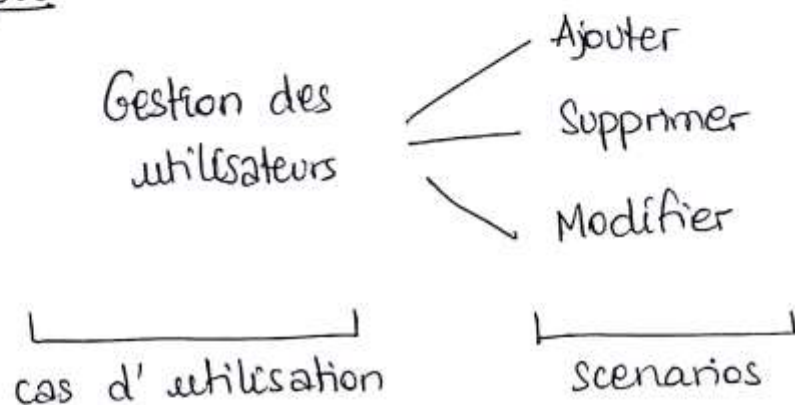
- Cas d'utilisation = ensemble de scénarii.

(2)

↳ décrit le comportement du système du point de vue des différents utilisateurs.

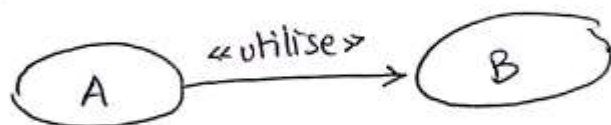
Il existe une différence entre un cas d'utilisat° et un scénario.

exemple



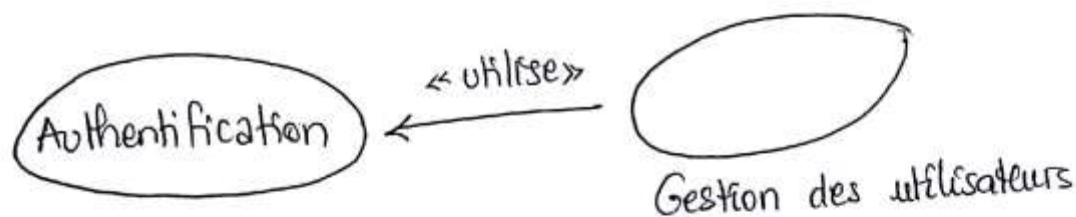
• Relations

④ • utilise (uses)



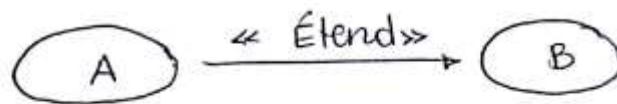
The meaning is simple, il suffit de lire following the arrow:
A utilise B. Mais ce n'est pas une simple utilisation dont on peut se passer, B est obligatoire pour réaliser A.

exemple



encore une fois, en suivant le sens de la flèche, on comprend que la Gestion des Utilisateurs utilise l'Authentification. Autrement dit, l'authentification est obligatoire pour la gestion des utilisateurs.

② • Étend (extend)

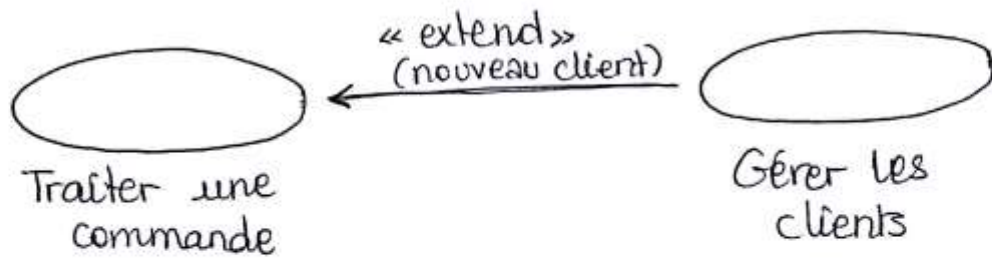


Encore une fois, il suffit de suivre le sens de la flèche.
A étend B, donc A est une extension de B.

Indeed: A est optionnel et ne se déclenche que par une condition dans le comportement de B.

on parle également du concept de "Point d'extension". C'est en général une condition qui déclenche le cas d'utilisation optionnel.

exemple



Ici, le point d'extension = Nouveau client.

Initialement, on est en train de traiter une commande. Je suis en train de saisir les données shik tik, soudain le système détecte que c'est un nouveau client (Point d'extension). L'appli va partir directement ouvrir le scénario "Gérer les cts", puis une fois l'ajout réglé, on retourne là où on s'était arrêté.

③ • Généralisation

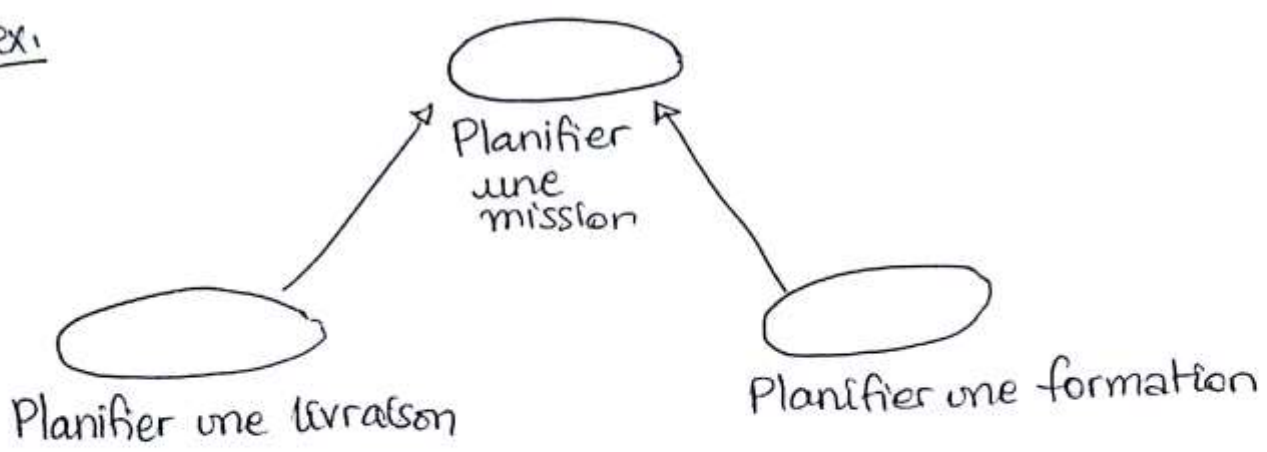


B est une abstraction de A.

A hérite de B. (la relation d'héritage permet d'affiner la classe "mère" tout en bénéficiant de sa struct. de données et de son comportement)

ex:

③



Etude de cas

24

Système d'Inscription Automatique

Au début de chaque session, les étudiants demandent un catalogue contenant une liste des cours disponibles durant la session. Pour aider les étudiants dans leurs choix, des informations sur chaque cours sont fournies (enseignant, département, pré-requis, etc.). Le système permettra aux étudiants de choisir 4 cours dans une session. En plus, chaque étudiant choisit deux cours optionnels en cas d'annulation ou de saturation d'un cours parmi les 4 précédemment choisis. Le nombre d'étudiants dans un cours est entre 3 et 10. Un cours contenant moins de 3 étudiants est annulé. Le processus d'inscription d'un étudiant est validé, le système envoie les informations au système de paiement. L'étudiant peut donc payer ses frais de scolarité.

Un enseignant peut indiquer les cours qu'il veut assurer, consulter la liste des étudiants inscrits dans un cours donné



© Ali Idri/UML/2002-2003

Exercices: cas d'utilisation

25

SIA

- ⊙ Identifier tous les acteurs de l'application SIA
- ⊙ Identifier tous les cas d'utilisation de l'application SIA
- ⊙ Donner le diagramme des cas d'utilisation de l'application SIA

© Ali Idri/UML/2002-2003

Acteurs

- Etudiant →
 - demander_catalogue_information
 - choisir_cours
 - modifier_cours
 - annuler_cours
 - consulter_cours

Inscription

Informat° cours

Pourquoi les a-t-on séparés ?

- si on ne les avait pas séparés, on aurait rendu "Infos-cours" exclusif à l'étudiant. Or, le professeur aura également besoin de consulter les inf
- Tout le monde peut consulter les infos, ils n'ont pas besoin de s'authentifier pour ce faire.

- Prof →
 - choisir_cours_enseigné
 - Modifier
 - Annuler
 - Consulter



Cours à enseigner

- système de paiement

- scolarité (Registrariat) ⇒

Gestion des étudiants

Gestion des profs

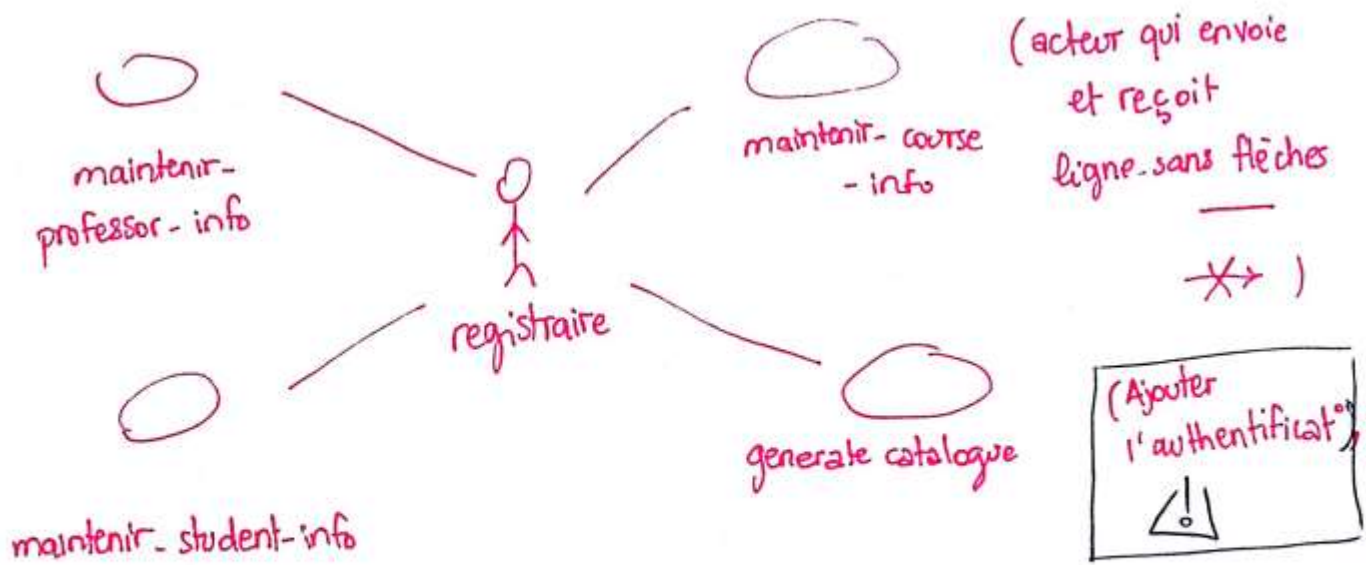
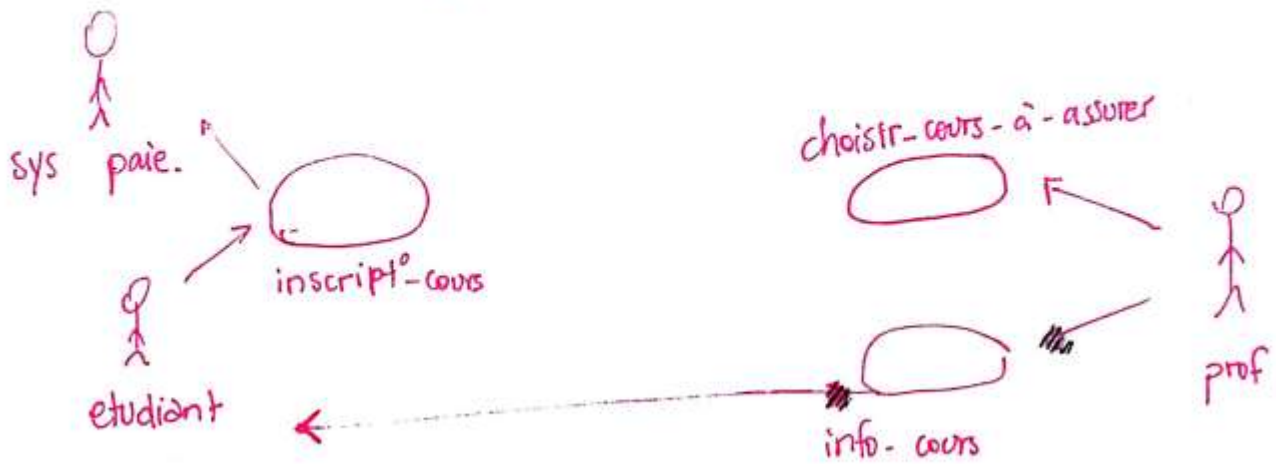
Gestion des cours

Gestion du catalogue

22/04/15

4

Diag. cas d'utilisat° (SIA)

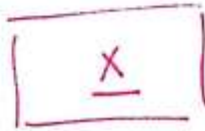


Pour un habitué au fonctionnel, on a l'impression que ce diagramme contient plusieurs systèmes. (Puisqu'il n'y a pas de relation^{apparente} entre les différents acteurs)
 Il n'y a pas de relation de type include, extend et généraliser, c'est pour cette raison qu'on ne montre pas les relations entre acteurs.
 Abstraction de niveau élevé.

on commence par les scénarii essentiels . (scénario principal)
et puis pour chaque scénario, on va représenter les objets qui interviennent
dans ce scénario.

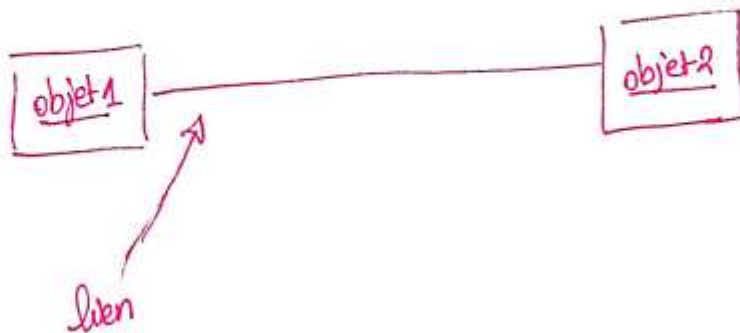
concrets abstraits

Notation :



objet de type X.

Interaction entre objets :



Diagrama

- de séquence → mettre l'accent sur l'aspect chronologique de
- de collaborat → flux de données

Si vs voulez les 2, et bien il faudra faire deux diagrammes.

1. Décrire textuellement (bullet points) le scénario "Créer calendrier".

1. L'étudiant s'authentifie dans l'application.
2. L'application affiche un "menu étudiant".
3. L'étudiant choisit "Créer-calendrier".
4. L'application affiche le catalogue.
5. L'étudiant choisit (4+2) cours.
6. L'étudiant soumet son calendrier à la validation.
7. L'application crée le calendrier de

1. ~~objet élève~~ initie la créat°.
2. ~~élève~~ consulte ~~cours~~ disponibles.
3. ~~élève~~ sélectionne

même nom que l'analyse pour une certaine homogénéité

2. Faire apparaître tous les objets

Etudiant



Forme_Auth

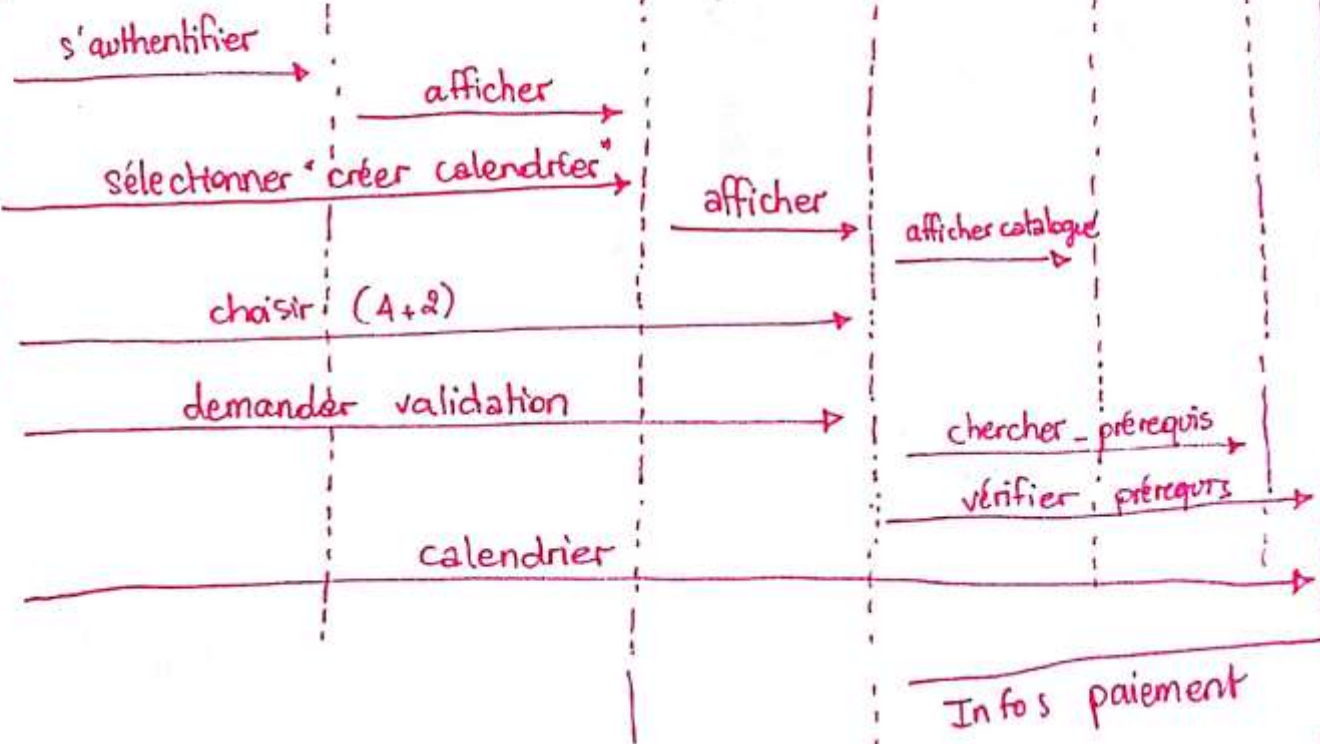
Forme_menu_etudiant

Forme_inscription

catalogue

cours

relevé de notes



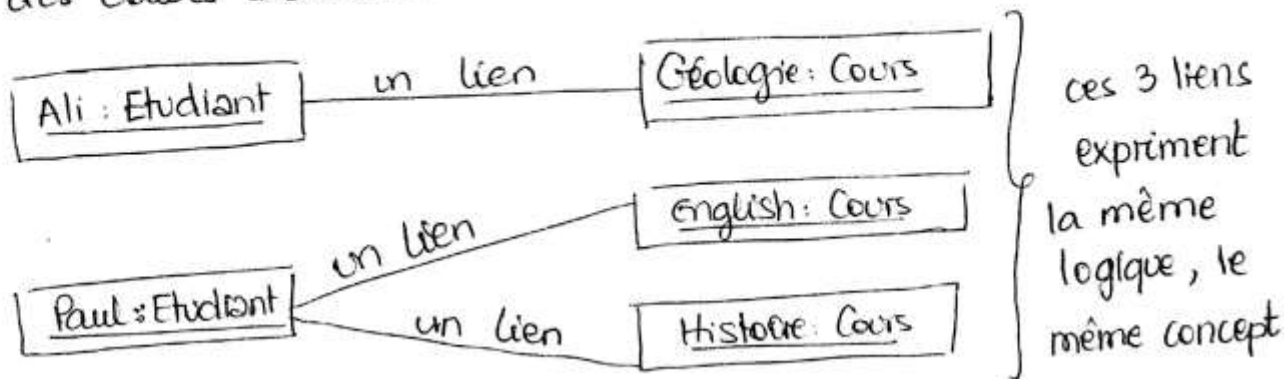
* Diagrammes de classes

- On garde la même sémantique que celle utilisée dans le monde extérieur.
- Une classe est une description d'un ensemble d'objets ayant :
 - les mêmes propriétés (attributs)
 - le même comportement (opérations)
 - les mêmes relations avec d'autres objets.
- Relations entre classes :
 - Association
 - Agrégation
 - Généralisation

① Association

C'est une abstract° des liens qui existent entre les objets instances des classes associées.

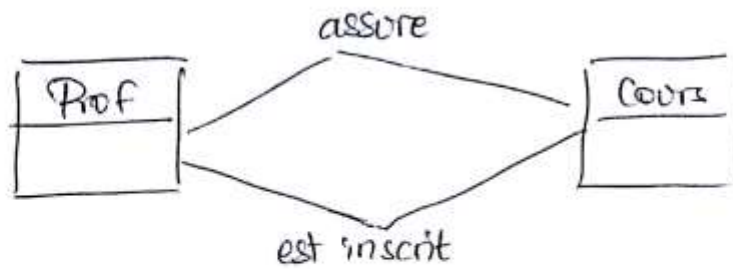
ex:



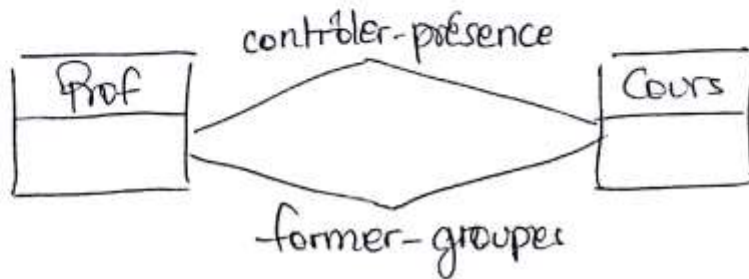
On peut avoir des associations multiples entre 2 classes.

⑥

Good :



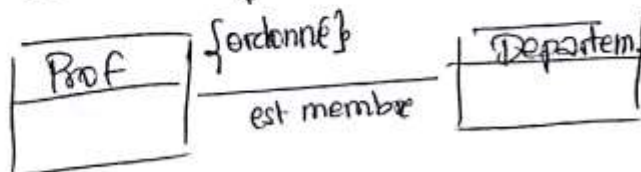
Bad:



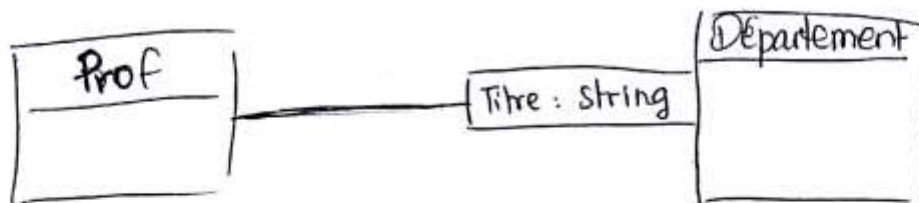
why is it a bad example ? Parce que les relations doivent être indépendantes (Alors qu'ici "Contrôler-présence" et "former-groupes" sont reliées).

• Notion de contrainte = C'est une condition qu'on doit préserver sur une association.


(tjs entre {})




• Notion de clé = sélection d'un sous-ens. d'obj. parmi l'ensemble des objets qui participent à une association.

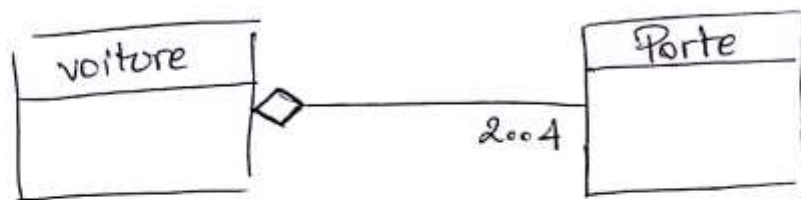


② Agrégation → c'est une forme particulière d'une association qui décrit une relation d'inclusion entre une partie et un tout (l'agrégat).

Si elle exprime une relation de composition (C'est-à-dire que les durées de vie des objets sont liées → Quand on supprime l'elt composite, il y a obligatoirement suppression des composants), elle est représentée par un losange plein .

Sinon (durées de vie indépendantes), elle est représentée par un losange vide  du côté de l'agrégat (le tout).

exemple d'une agrégation :



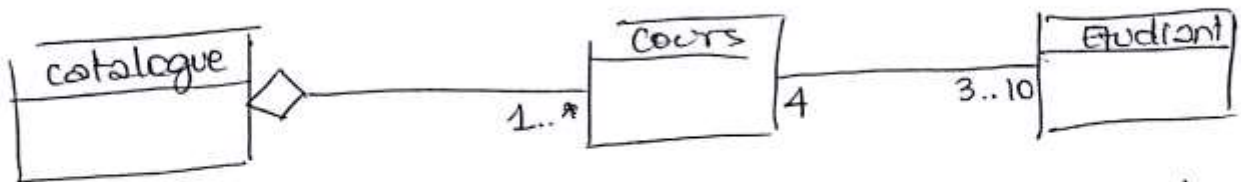
Étant donné une voiture, celle-ci est composée de 2 à 4 portes.

Association vs Agrégation

2 entités autonomes
qui interagissent

inséparable (+ ∃ un lien de couplage fort. C.à.d dès qu'on touche à un composant, on peut affecter les autres).

ex:



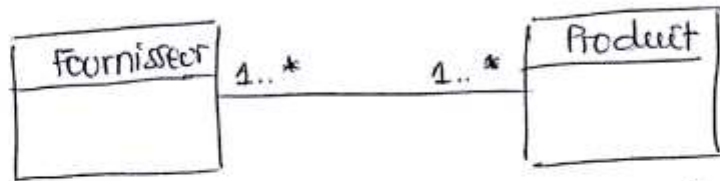
Un cours est associé à 10 étudiants mais les étudiants ne sont pas des composants du cours ⇒ C'est donc une associat°. Par contre, le catalogue est composé de plusieurs cours, donc on parle d'agrégation.

- Navigat° d'une association et d'une agrégation.

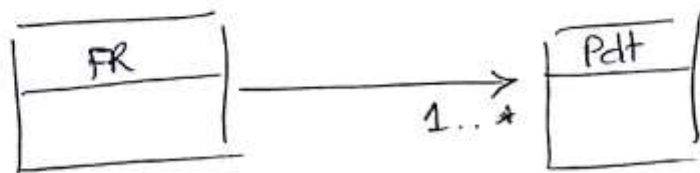
(7)

↳ on ne laisse que le sens dans lequel nous sommes intéressés.

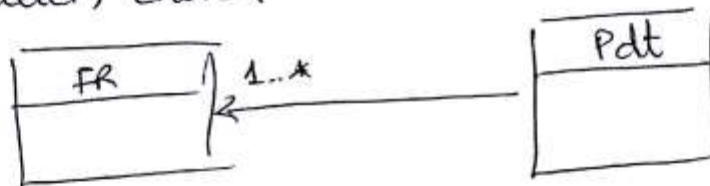
exemple



si je m'intéresse à ce que, étant donné un fournisseur, on puisse voir tous les produits fournis par ce fournisseur, alors on aura ce schéma:



si on s'intéresse plutôt à chercher tous les fournisseurs d'un produit particulier, alors:



Donc, des fois, d'association dans les deux sens doesn't make sense (si on n'est intéressé que par un sens).

Pour le cas de l'agrégation, elle ne peut être naviguée que dans le sens de la classe composante:

ex:



L'implantation d'une agrégation se fait de 2 manières :

Par valeur ◆

L'objet composant est un attribut de l'objet composant.

ex:

```
class voiture {  
    ...  
    porte new [4]  
    ...  
}
```



once we delete the car, on perd les portes également.

Par référence ◇

L'objet composant est référéncé dans l'objet composé par un ptr, index, etc...

ex:

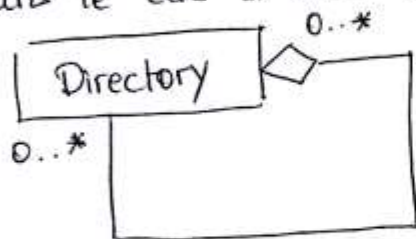
```
class voiture {  
    ...  
    int IDporte [4]  
    ...  
}
```



Même si on perd la voiture, les portes sont toujours là.

Dans le cas d'une association, on le fait par réf.

o



Ref de mes sous-répertoires

class repertoire

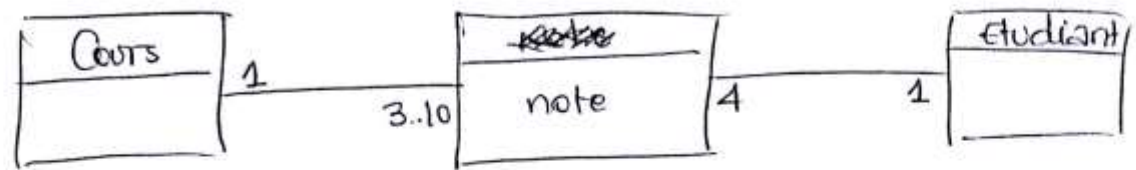
```
{  
    string sous_repertoire [Max];  
    string super_repertoire [Max-1];  
}
```

Ref des répertoires dont je suis un sous-rep.

- o var
 - locale → existe et est utilisée dans la fonction où elle est utilisée
 - globale → elle peut être utilisée dans une autre place.

• Implantation d'une association représentée par une classe

exemple



class cours

```

{
ent IDnotes [10]; // une liste de refs vers les notes
}
    
```

class note_cours_etudiants

```

{
ent ID_cours;
ent ID_etudiant;
}
    
```

class Etudiant

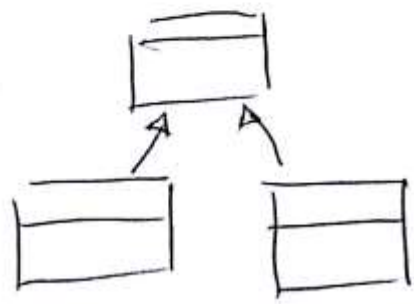
```

{
ent IDnotes [4];
}
    
```

• Généralisation / Spécialisation

Classes qui partagent des trucs en commun

⇒ on regroupe ces trucs (on factorise) dans une super-classe, et c'est dans les sous-classes où on va spécifier les détails.

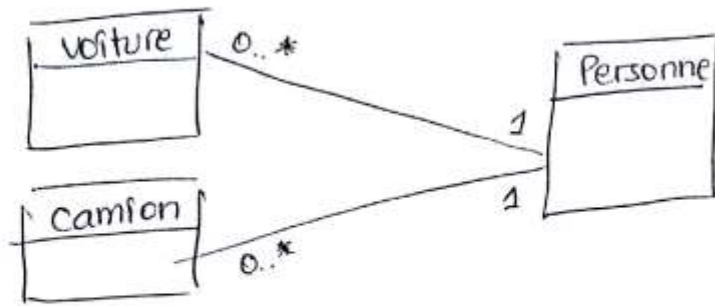


↑ This way : on factorise (en terme d'att et de méthodes)

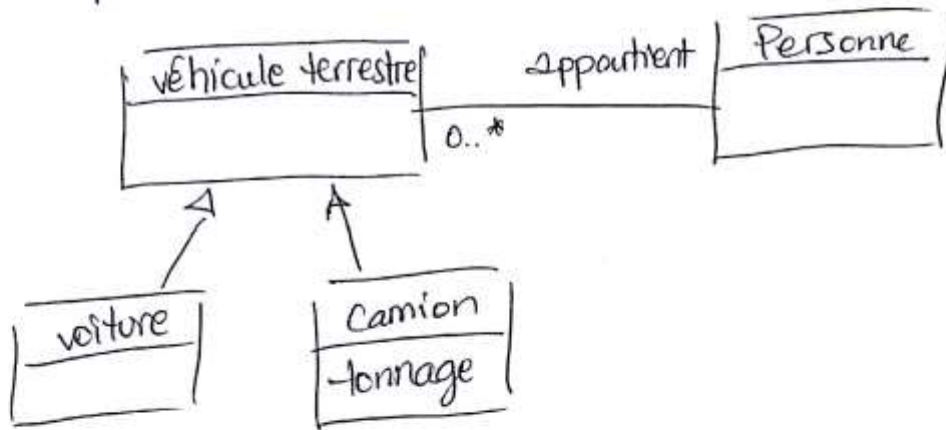
↓ This way : les spécificités des classes

ex:

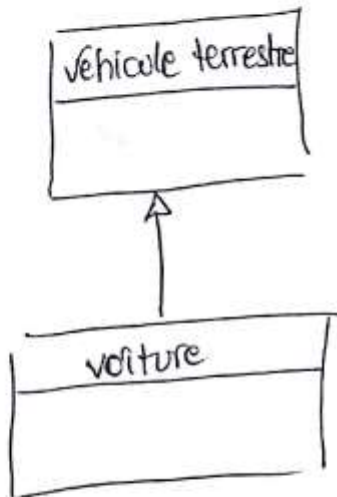
Avant la factorisation, on avait:



on crée une classe véhicule terrestre qui regroupe tout ce qui est commun.



o



Comment peut-on représenter cette relation d'héritage dans ~~un~~ le modèle relationnel?

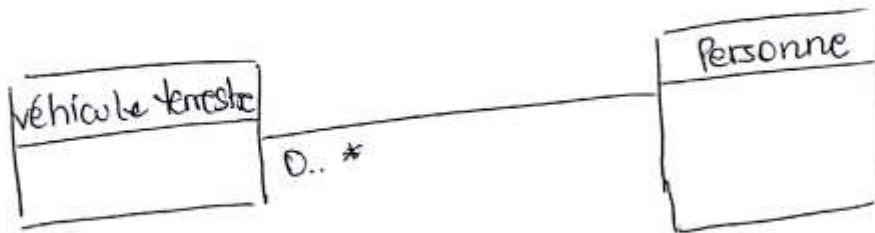
↑
Cette question fait partie de celles que le prof nous demandait de noter car elle pourrait figurer ds l'exam.


Have the answer? How about you post it in the comment section?


- Si une classe est généralisée par au plus une classe
 \Rightarrow Généralisation simple.
- Classe généralisée par $+^{\text{IS}}$ classes \Rightarrow Généralisat^o multiple.
- L'héritage multiple est dangereux
 \Rightarrow confusion au niveau des attributs / méthodes
 (Le langage doit avoir un algorithme de priorité pour résoudre ces situations de conflit.)

• L'héritage

on dit qu'une classe X généralise Y et non Y hérite de X (vocabulaire propre aux lang. O.O)



Pour , on déclare un attribut qui n'est qu'un pointeur
 (Actually, une liste de ptr car on a 0..*) vers la structure "véhicule terrestre".

Pour , il suffit de déclarer un seul pointeur vers Personne.

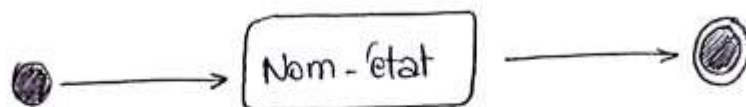
Agrégation \neq Généralisation /spécialisation.

How? Quand une classe est une spécialisation d'une autre, elle est de même nature, ce qui n'est pas le cas pour l'agrégation.

* Diagramme d'états-transitions

- on parle de diag. d'Etats-transit° d'une classe.
- L'état d'un obj est généralement défini par les vals de ses attributs.
- Si on a une classe dont l'état des objets ne varie pas, on n'a pas recours à ce diagramme.

Notation:



Tout diag. d'E. a un seul état initial.

- peut avoir $+^{\infty}$ états finaux.

- Notion d'événement := occurrence d'une situation particulière dans le domaine du problème.

exemple

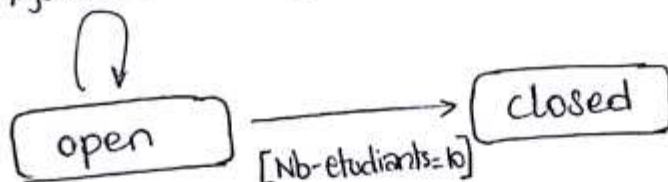
Evt: Ajouter un étudiant ds le cours



- Notion de transit° := chgt d'état d'un objet suite à un evt.

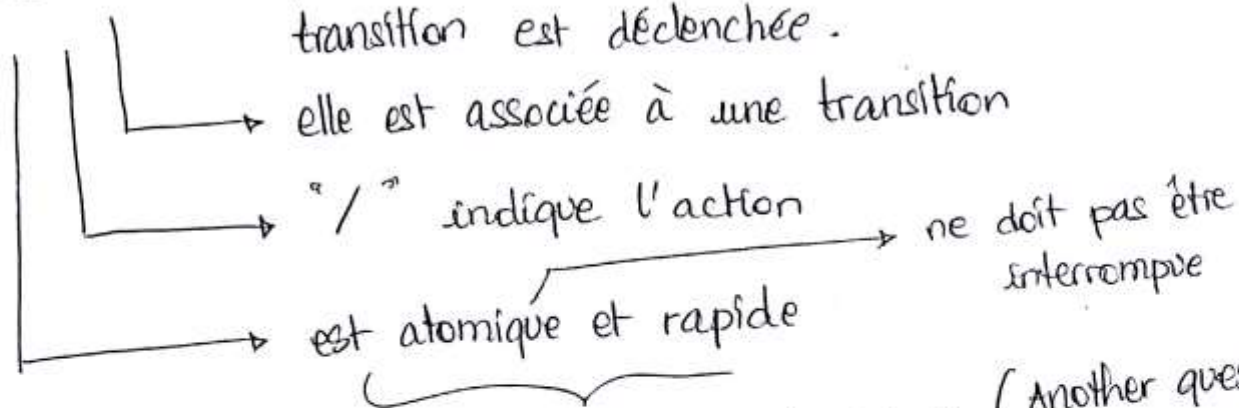
ex:

Ajouter_etudiants [Nb_etudiants < 40]



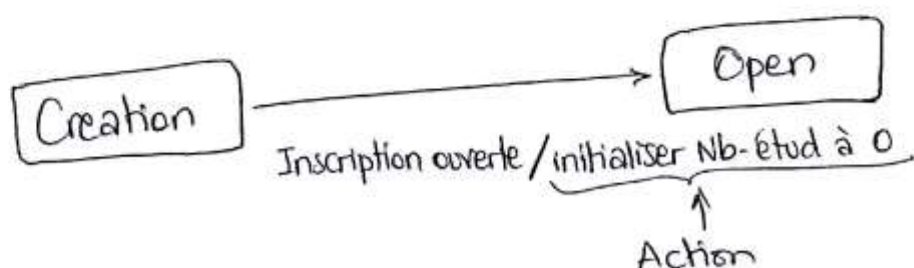
↑
condition de déclenchement
de la transition

- Action → opération exécutée lorsque la transition est déclenchée.



Quel en est l'intérêt ? (Another question the teacher asked us !)

ex:



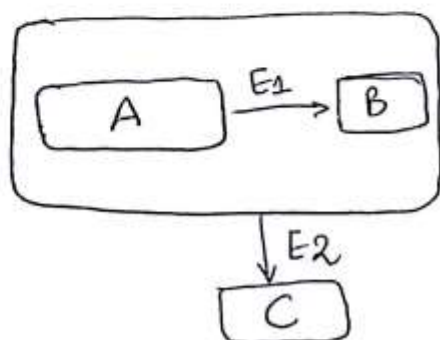
- Activité → Associée à un état plutôt qu'à une transition.



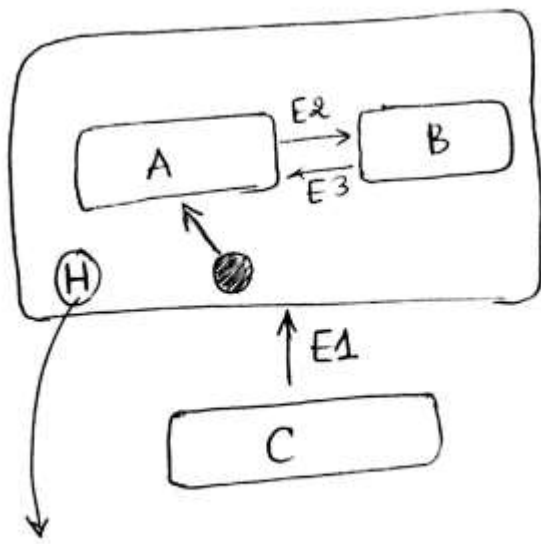
Pourquoi cela ne pose pas de pbs ici ?
(Another question !)

- Transition automatique → déclenchée suite à un deadline ou à une activité qui va se terminer.

- Imbrication des états.



Être dans un super-état ici
c'est être dans A ou B



Lorsqu'on est dans C et on a E1, on passe au super-état et on rentre automatiquement à l'état initial du super état qui nous renvoie automatiquement à A.

Autrement dit, $C + E1 \rightarrow A$

(sans l'état initial, on n'aurait pas su par où commencer)

historique :

Quand je suis dans le superétat et que je sors pour une raison ou une autre, je veux que lorsque je revienne, je retrouve le dernier état visité et non l'état initial.

Comment peut-on réaliser le mécanisme d'historique avec du code ?

(Another question!)

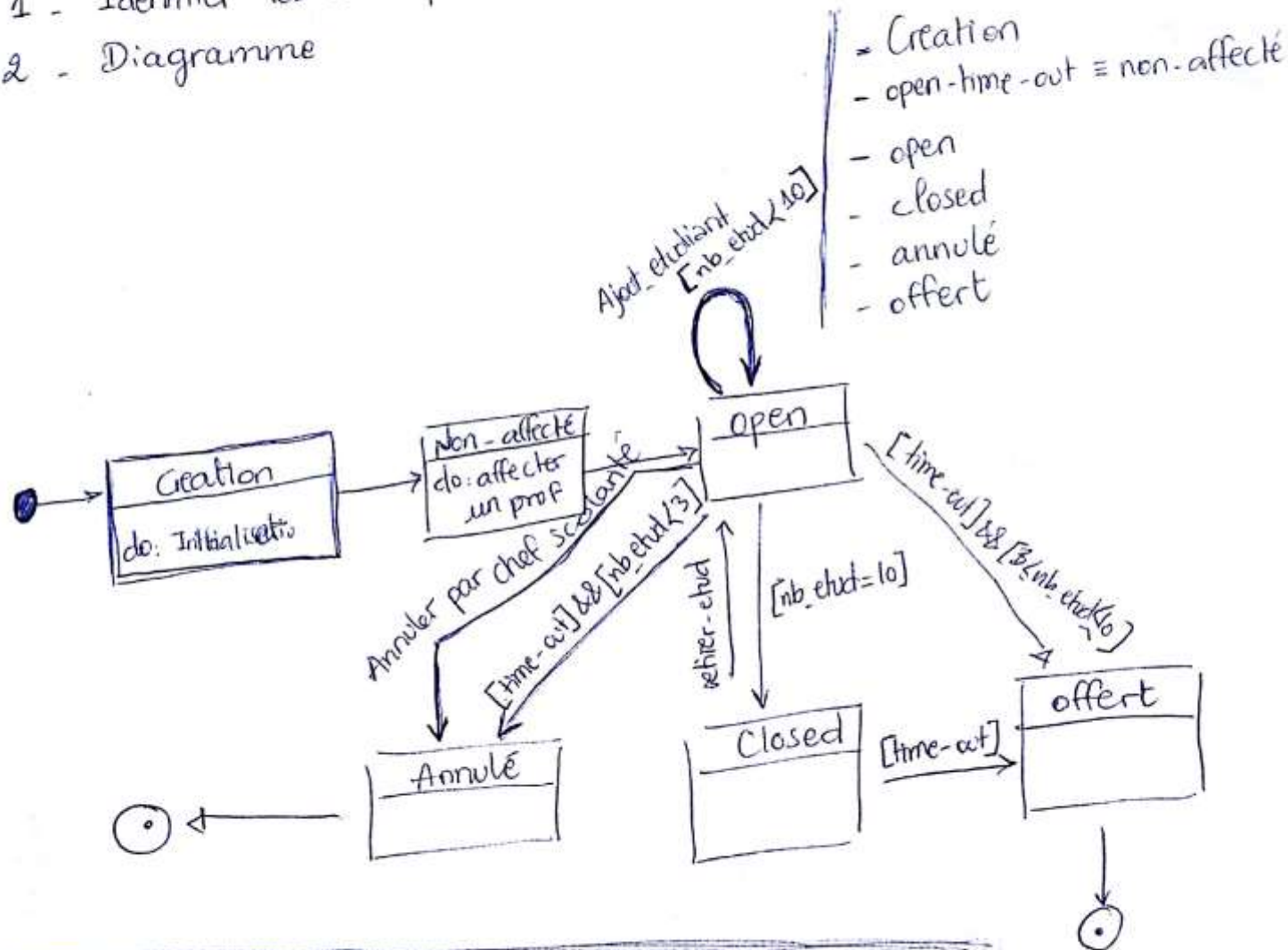
Exercices: Diagrammes d'états-transitions

- ⊙ Donner le diagramme d'états-transitions de la classe Cours de l'application SIA
- ⊙ Simplifier ce diagramme en utilisant les mécanisme d'imbrication des états avec un historique

20/05/15

Diagramme d'Etats-transitions de la classe "Cours"

- 1 - Identifier les états possibles de la classe "Cours"
- 2 - Diagramme



Mapping?

Analyse et Conception: UML

Code:



stockage:

BD relationnelle
Diag. de tables

généralisé/spécialisé
tables ont la même clé
(infos communes de la table gén.)

association

↓ code
par référence

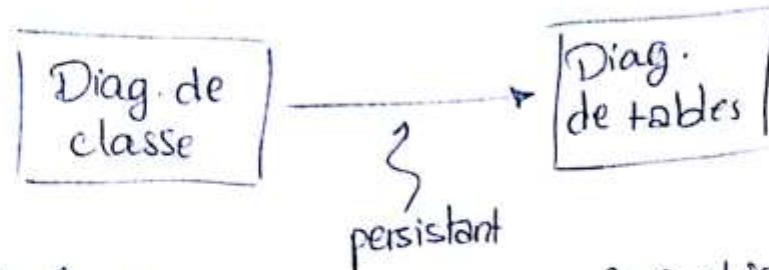
(vs déclarer une liste des ids des commandes)

contrôler l'évolution des états.



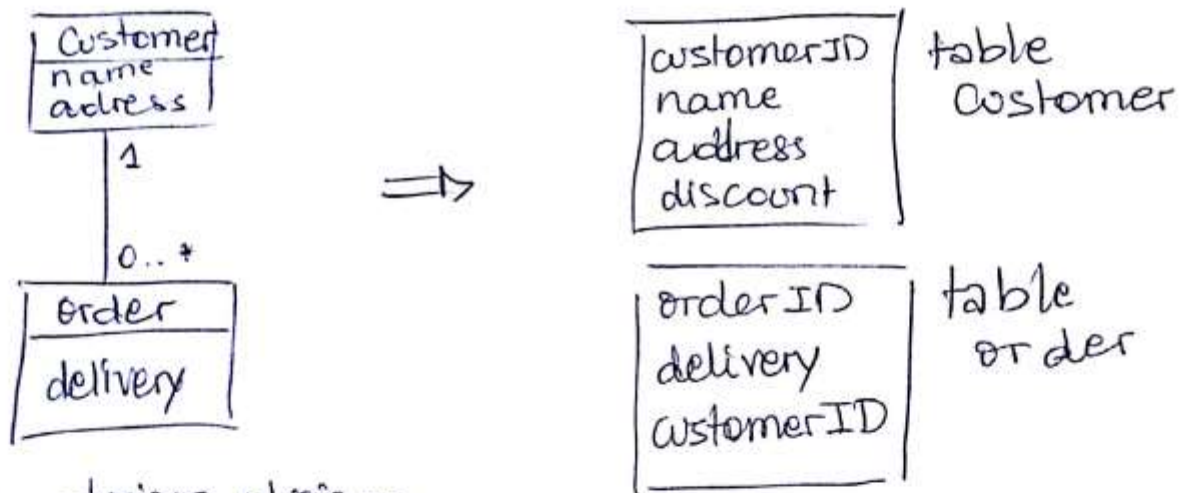
Static. durée de vie = exécut^o du code.
 (il est créé à l'entrée de l'exécution et détruit vers la fin)

persistant: dont la durée de vie ne dépend pas du pb.

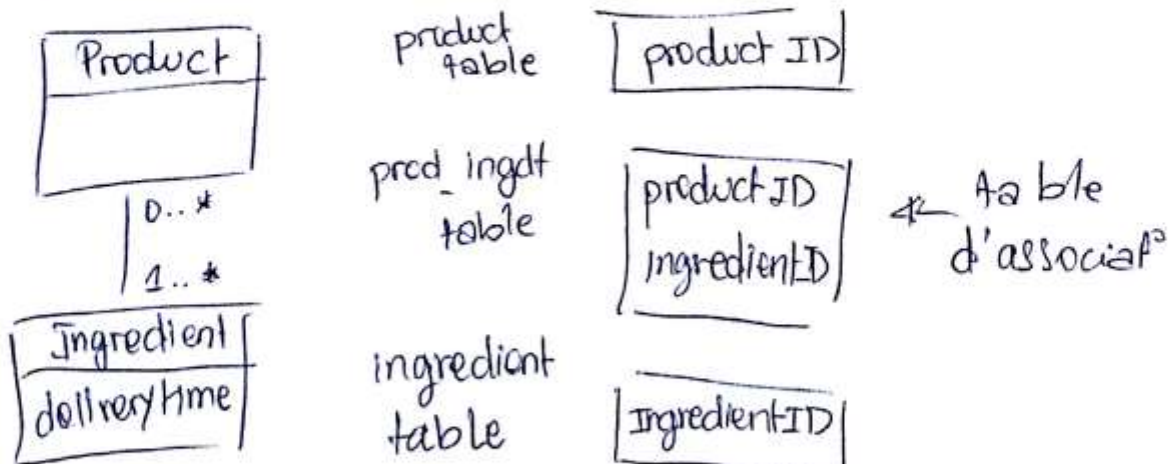


3 relat^o entre les classes: - généralisation.
 - association
 - agrégation

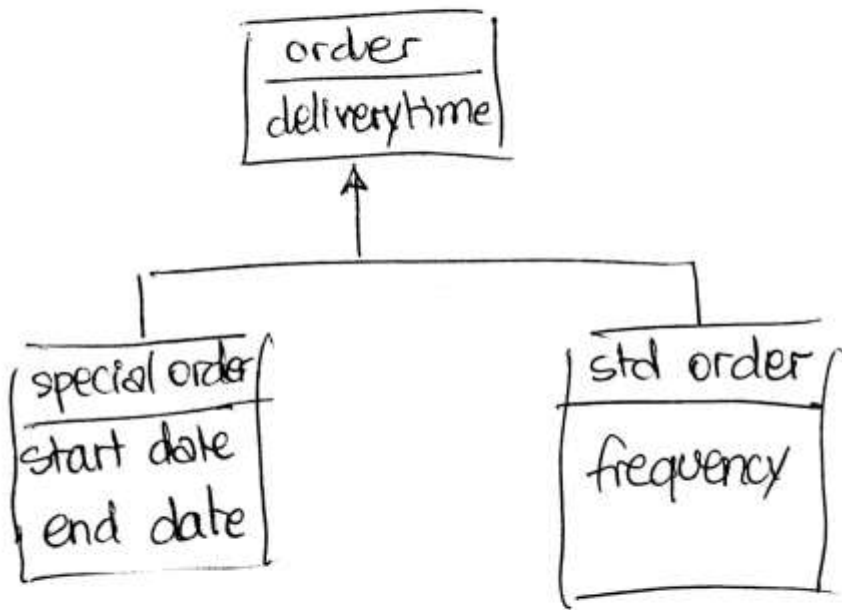
Dans la classe Customer, on déclare une liste des ID des orders.
 (comme étant une clé étrangère)



* relation plusieurs plusieurs



Généralisation



order table

order ID	delivery time
-------------	---------------

special order table

order ID	end date	start date
-------------	----------	------------

std order table

order ID	frequency
----------	-----------

L'héritage est réalisé avec la clé qui est la même dans toutes les tables.

Etude de cas : Analyse et Conception Orientées objet -UML-

Sujet

Une agence de location de voitures veut développer son propre système pour la gestion des demandes de location. Le système doit permettre à ses clients de :

- 1- consulter les informations sur les voitures à louer (marque, couleur, puissance, carburant, nb de places, ..)
- 2- faire une réservation,
- 3- annuler une réservation,
- 4- modifier une réservation (durée de réservation, type d'assurance,..)

Le système annule une réservation si le client ne se présente pas dans un délai de 2 jours. Il doit après avis, par email, le client suivant dans la liste d'attente (s'il y en a). Le système doit permettre aussi le suivi d'une location et aider à prendre des décisions si un événement s'est produit au cours de la location (accident, panne, vol, ...).

Le système gère aussi les clients et calcule la facture de chaque location. Les clients sont des abonnés 'simples' ou 'fidèles'. A chaque location réalisée par un client, le système lui accorde un bonus dépendamment des caractéristiques de la location (type d'abonnement, durée de location, ..). Le coût d'une location est calculé selon plusieurs critères : type d'abonnement, durée de location, Bonus cumulé par le client, Prix par jour, Nombre de kilomètres parcourus par la voiture, ...).

Travail demandé

Premiere seance :

- 1- Identifier et décrire tous les acteurs du système
- 2- Identifier et décrire brièvement chaque cas d'utilisation
- 3- Donner le diagramme des cas d'utilisation de ce système.

Deuxieme seance :

- 4- Donner et expliquer les diagrammes de séquence des deux scénarios 'faire une réservation' et 'calculer facture'

Troisieme seance :

- 5- Donner le diagramme de classes du système

Quatrieme seance

- 6- Terminer le diagramme de classes
- 7- Donner le diagramme d'états-transitions de la classe 'voiture'


05/05/15

Etude de cas : Analyse et Conception Orientées Objet - UML -

Première séance

- 1 -  - client
- Manager
- comptable
- système de messagerie
-) on peut les regrouper si on veut.

2 - Identifier et décrire brièvement chaque cas d'utilisation.

*  Client

Reservation

- créer une réservation
- modifier une "
- annuler une "
- consulter mes réservations
- consulter Bonus

Consulter Informations

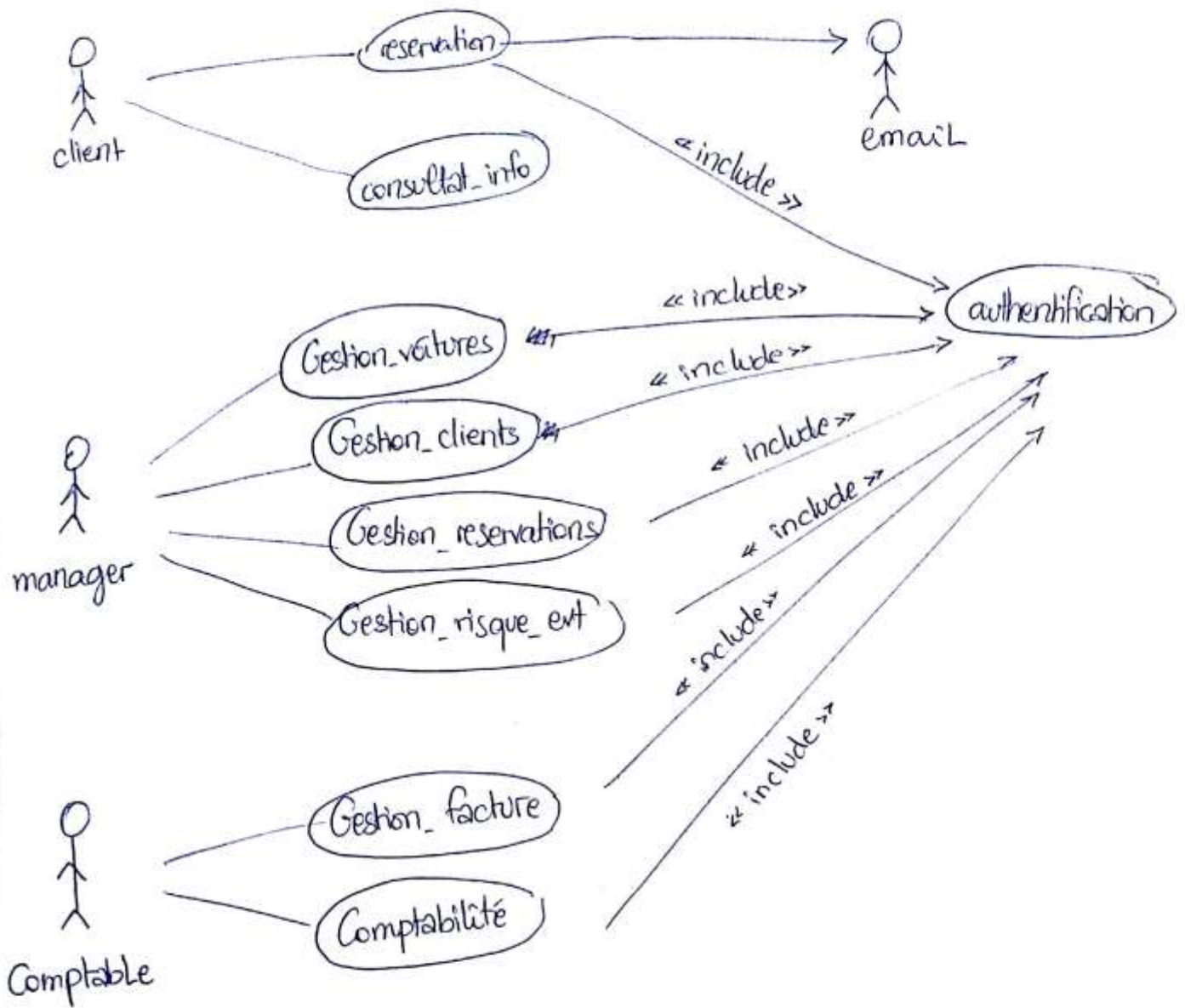
* Manager 

- Gestion des clients
- Gestion des réservations
- Gestion des voitures
- Gestion des risques - événement

* Comptable 

- Gestion facture
- Comptabilité

3. Donner le diagramme des cas d'utilisation de ce système



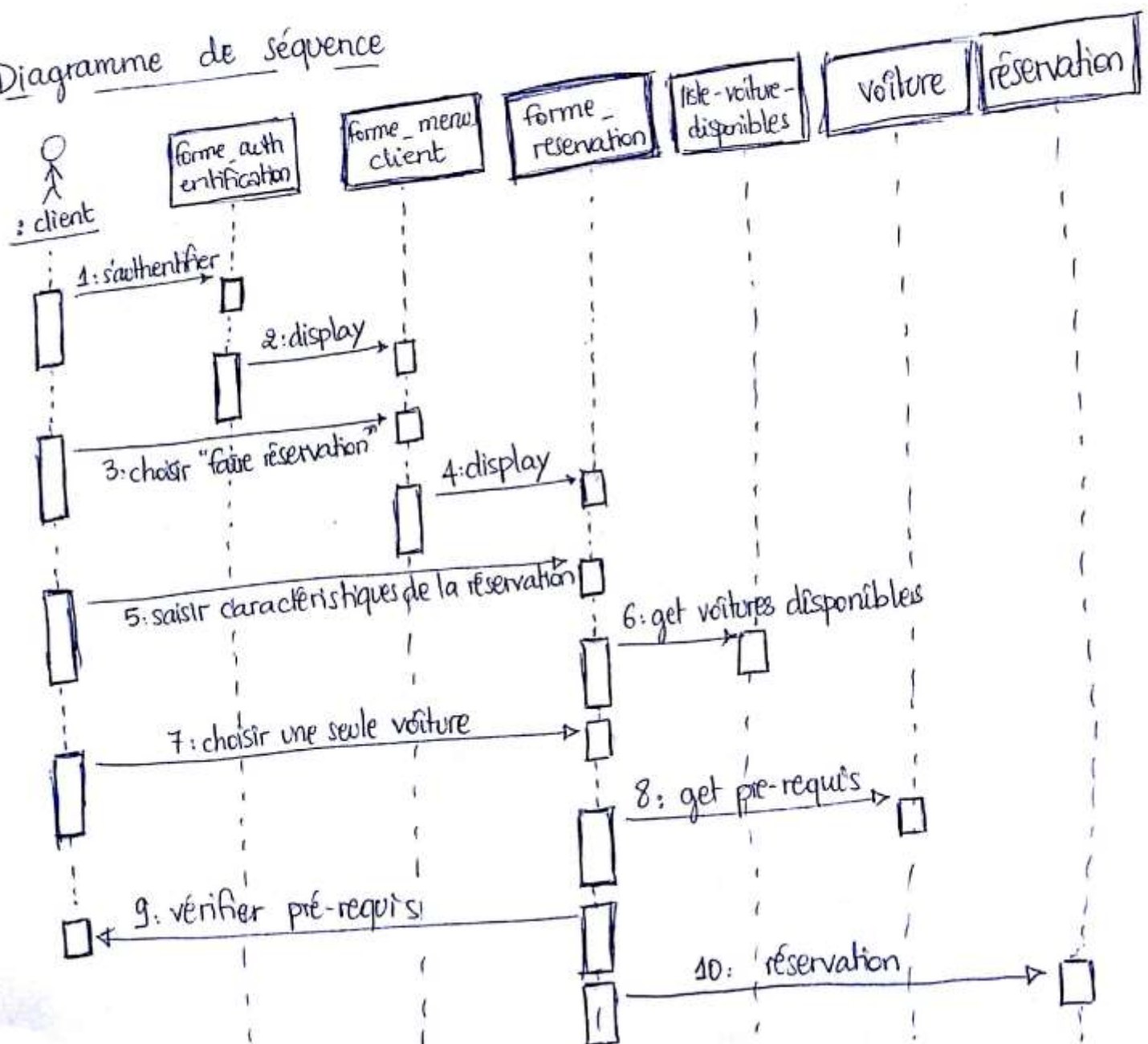
files
→ descript° détaillée

19/05/15

Description du scenario : faire une réservation

- 1 - Le client s'authentifie dans le système.
 - 2 - Le système affiche le menu du client.
 - 3 - Le client choisit l'option "Faire une réservation"
 - 4 - Le client saisit les caractéristiques de la réservation.
 - 5 - Le système affiche la liste des voitures disponibles (qui répondent aux caractéristiques)
 - 6 - Le client choisit une seule voiture.
 - 7 - Le système sauvegarde la réservation.
- 6' : Le système cherche les pré-requis de la voiture.
6'' : le système vérifie les pré-requis

Diagramme de séquence

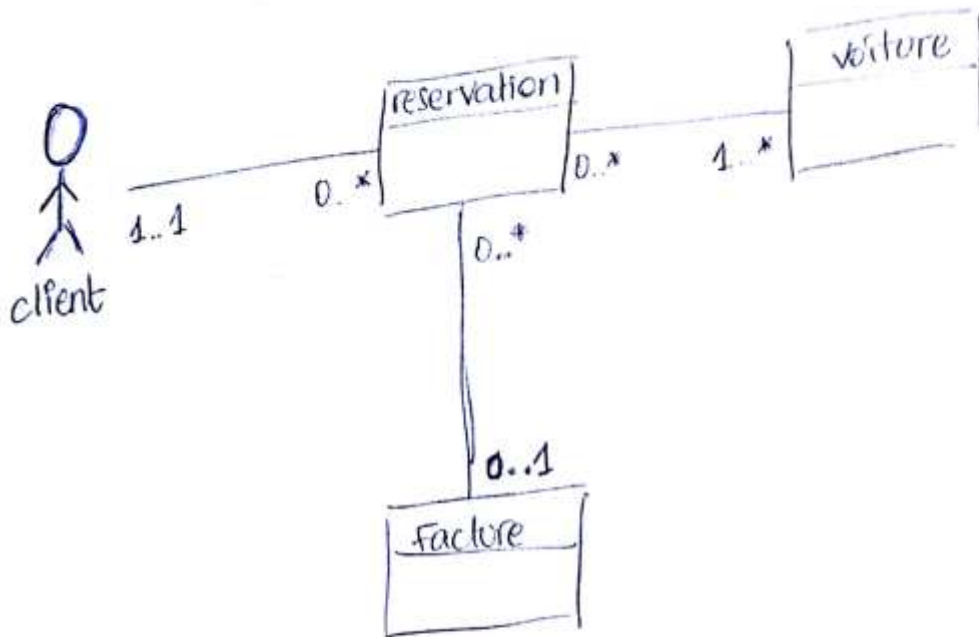


Description du scénario : Calculer facture

- 1 - Le comptable s'authentifie.
- 2 - Le système affiche "menu-comptable"
- 3 - Le comptable choisit l'option "calculer-facture".
- 4 - Le système affiche la forme "forme-facturation"
- 5 - Le comptable saisit ID de la réservation.
- 6 - Le système affiche les caractéristiques de la réservation.
- 7 - Le système cherche le type d'abonnement et le bonus du client.
- 8 - Le système cherche le prix par jour et le nombre de kilomètres parcourus.
- 9 - Le système calcule le montant de la facture
- 10 - Le comptable indique le mode paiement.
- 11 - Le système sauvegarde la facture.

26/05/15

Diagramme de classes



nbre limité de classes

↓
Alrighty!

Très gd nbre de classes

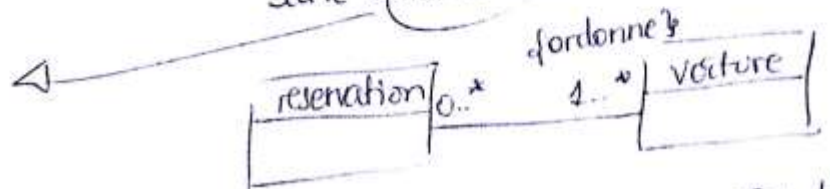
↓
on a recours aux packages

=
ens. de classes qui partagent un certain nbre de caract.

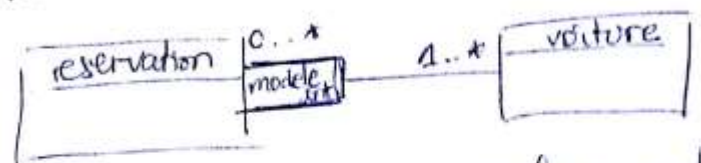
Par ex, on pourrait avoir des packages JHM où l'on regrouperait toutes les classes

Pour avoir la liste des voitures dans un ordre, il faut ajouter une contrainte:

n'exclut pas les objets

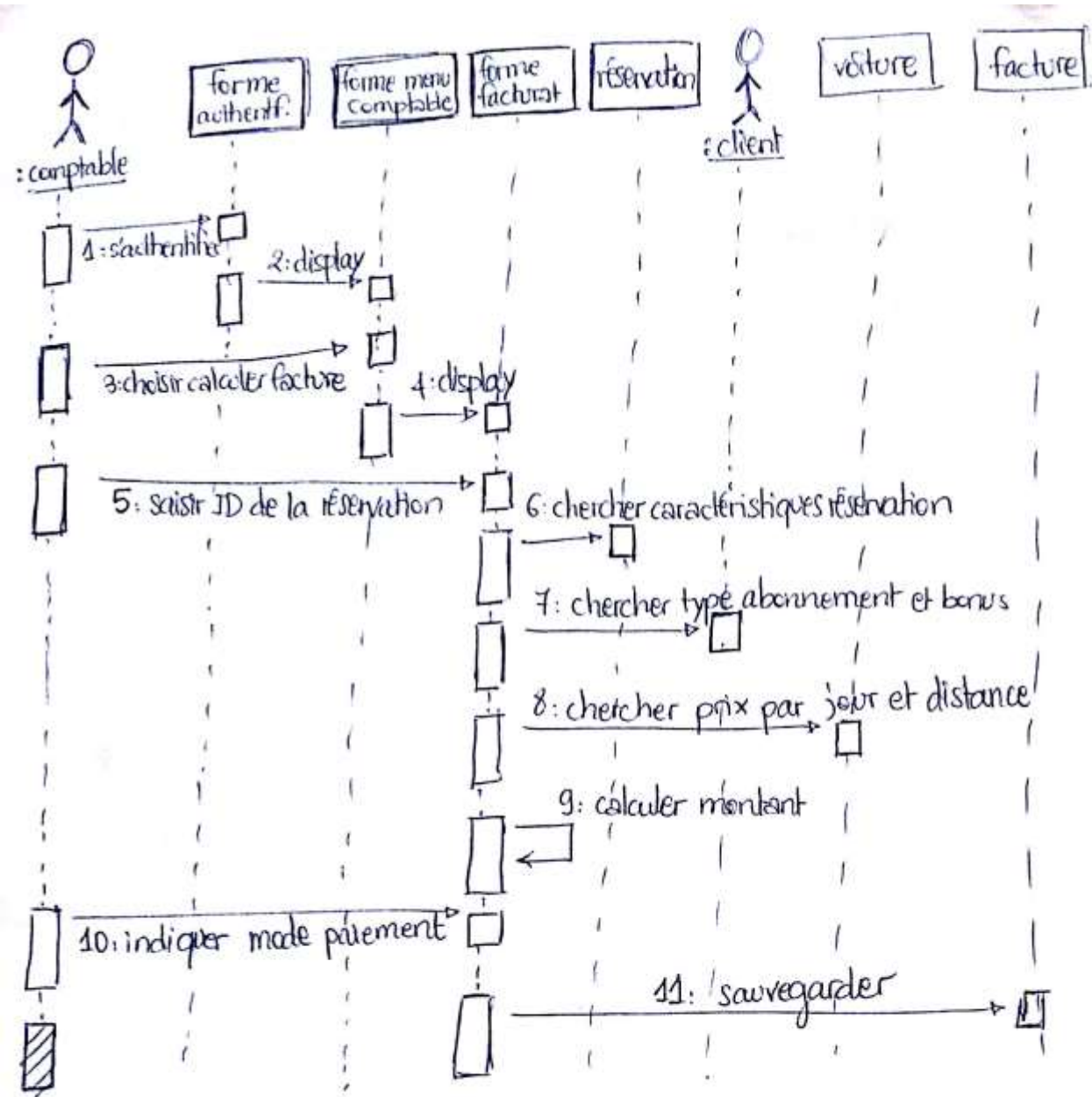


Etant donné une réservation, je veux avoir les voitures dont l'âge est > x, on met une clé.



select° → on peut exclure certains objets.

propriété sur laquelle vous faites une select°.



objets de forme st
généralement de type
transit.
(existe juste un moment)

il n'y a pas un
seul objet de
type x qui est
impliqué.

persistents (indép.
de l'exécution)
(Mm si le sys n'est pas
en exécution, il faut être
capables de les retrouver
en BD).



associat° intéressante ds un seul sens.

losange noir

↓
par valeur

⇓
partie intégrante.



on met le losange
du côté composant

* Diag. état-transit° de la classe voiture

Etats de la classe "voiture"

- Loué
- Disponible
- En panne
- volé
- accidenté
- réservé
- confirmé-réservation

