UML : Les diagrammes structurels

BAKAYOKO Mekossou

Janvier 2018

Table des matières

Objectifs

A la fin de cette leçon vous serez capable de :

- *Maîtriser* le diagramme de *classes*
- *Maîtriser* le diagramme d'*objets*
- *Maîtriser* le diagramme de *composants*

Introduction

Les diagrammes structurels permettent de présenter une vue des éléments statiques du systèmes : les objets en présence, les classes d'objets, les composants... Il n'est pas ici question de modéliser de quelle manière le système se comporte, mais de quels éléments il est constitué.

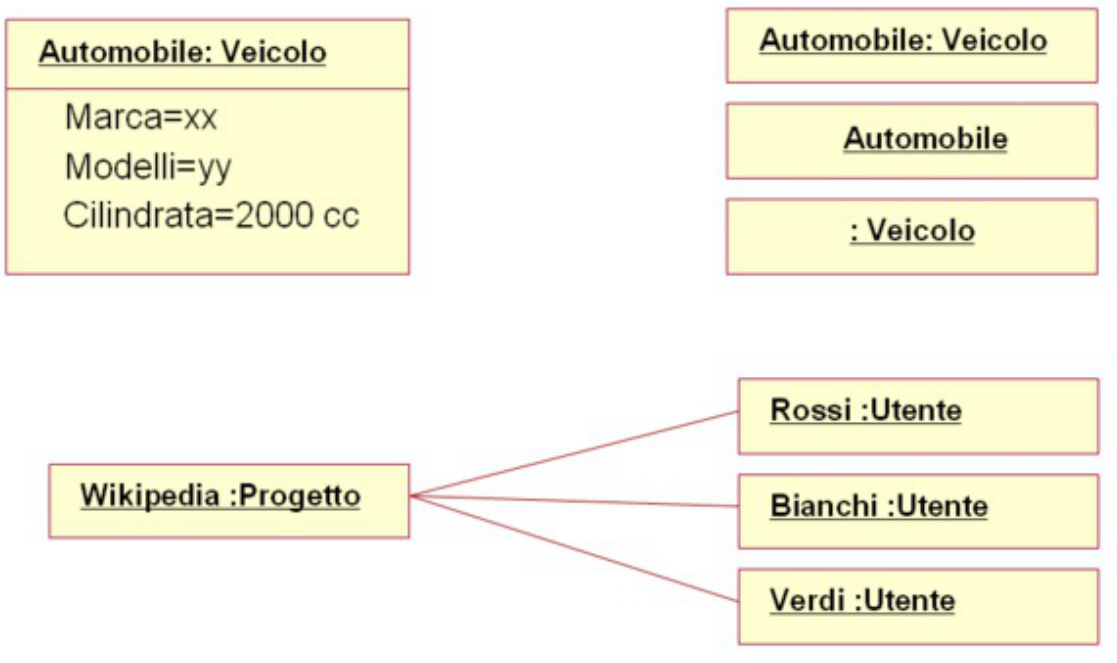


Figure 1 : Exemple de diagramme des objets

Le diagramme des classes

Objectifs

A la fin de cette section vous serez capable de définir le diagramme de classes.

Le diagramme de classes est un élément central d'UML. Il permet de représenter, avec une grande puissance d'expression, les types d'éléments en présence dans un système ainsi que les relations entre ces types d'éléments.le d

1. Le diagramme des classes

1.1. UML : La classe

1.1.1. Rappels sur la Définition et les objectifs d'un diagramme de classes

Définition : La Classe

Une classe définit un modèle d'objet (une sorte de moule, qui permet de créer des objets ayant des caractéristiques communes).

C'est un élément nommé, qui regroupe au sein d'une même structure des attributs (les données) et des méthodes (les traitements).

Fondamental : A quoi sert le diagramme de classe ?

On utilise les diagrammes de classe pour décrire la structure d'un système ou d'un sous-système.

le modélisateur objet crée un diagramme de classes, pour mettre à plat les différentes entités manipulées par le système.

1.1.2. Représentation graphique d'une Classe en UML

Dans sa plus simple représentation, une classe est un simple rectangle dans lequel on inscrit un nom.

Voici donc comment nous représentons une classe `Livre` :

Figure 1 : Classe simple

Tout livre a un titre et un certain nombre de pages. Ajoutons donc quelques attributs à cette classe :

Figure 2 : La classe livre avec attributs

Une classe regroupe des attributs et des méthodes permettant d'accéder et de manipuler les instances de la classe et leurs attributs. Nous allons donc représenter des méthodes :

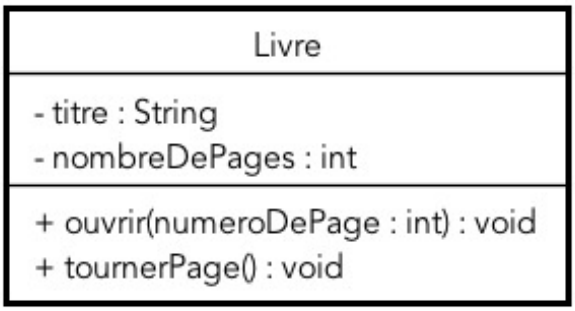


Figure 3 : La classe livre avec attributs et methodes

Conseil

- Rappelons que le nom d'une classe est au singulier
- Le nombre d'attributs et de méthodes est variable selon chaque classe. Toutefois, un nombre élevé d'attributs et/ou de méthodes est déconseillé. Il ne reflète pas, en général, une bonne conception de la classe.

Fondamental : Notion de type

Nous appelons ici *variable* tout *attribut*, *paramètre* et *valeur* de retour d'une *méthode*. D'une façon générale, nous appelons variable tout élément pouvant prendre une valeur.

Le type est une contrainte appliquée à une variable. Il consiste à fixer l'ensemble des valeurs possibles que peut prendre cette variable. Cet ensemble peut être une classe, auquel cas la variable doit contenir une référence vers une instance de cette classe. Il peut être standard comme l'ensemble des entiers, des chaînes de caractères, des booléens ou des réels. Dans ces derniers cas, la valeur de la variable doit être respectivement un entier, une chaîne de caractères, une valeur booléenne et un réel.

Les types standard sont désignés ainsi :

- *Integer* pour le type des entiers ;
- *String* pour le type des chaînes de caractères ;
- *Boolean* pour le type des booléens ;
- *Real* pour le type des réels.

Exemple

- *1* ou *3* ou *10* sont des exemples de valeurs d'*entier*.
- *"Cheval"* est un exemple de *chaîne de caractères* où les guillemets ont été choisis comme séparateurs.
- *False* et *True* sont les deux seules valeurs possibles du type *Boolean*.
- *3.1415*, où le point a été choisi comme séparateur des décimales, est un exemple bien connu de nombre *réel*.

Rappel : Encapsulation

L'encapsulation a été introduite dans le chapitre Les concepts de l'approche par objets. Certains attributs et méthodes ne sont pas exposés à l'extérieur de l'objet. Ils sont encapsulés et appelés attributs et méthodes privés de l'objet.

UML, comme la plupart des langages à objets modernes, introduit trois possibilités d'encapsulation :

- *L'attribut ou la méthode privée* : la propriété n'est pas exposée en dehors de la classe, y compris au sein de ses sous-classes.
- *L'attribut ou la méthode protégée* : la propriété n'est exposée qu'aux instances de la classe et de ses sous-classes.
- *L'encapsulation de paquetage* : la propriété n'est exposée qu'aux instances des classes de même paquetage. La notion de paquetage sera abordée au chapitre La structuration des éléments de modélisation.

L'encapsulation est représentée par un signe *plus*, un signe un *dièse*, ou un *tilde* précédant le nom de l'attribut.

Le tableau suivant détaille la signification de ces signes.

public	+	élément non encapsulé visible par tous
protégé	#	élément encapsulé visible dans les sous-classes de la classe
privé	-	élément encapsulé visible seulement dans la classe
paquetage	~	élément encapsulé visible seulement dans les classes du même paquetage

Exemple

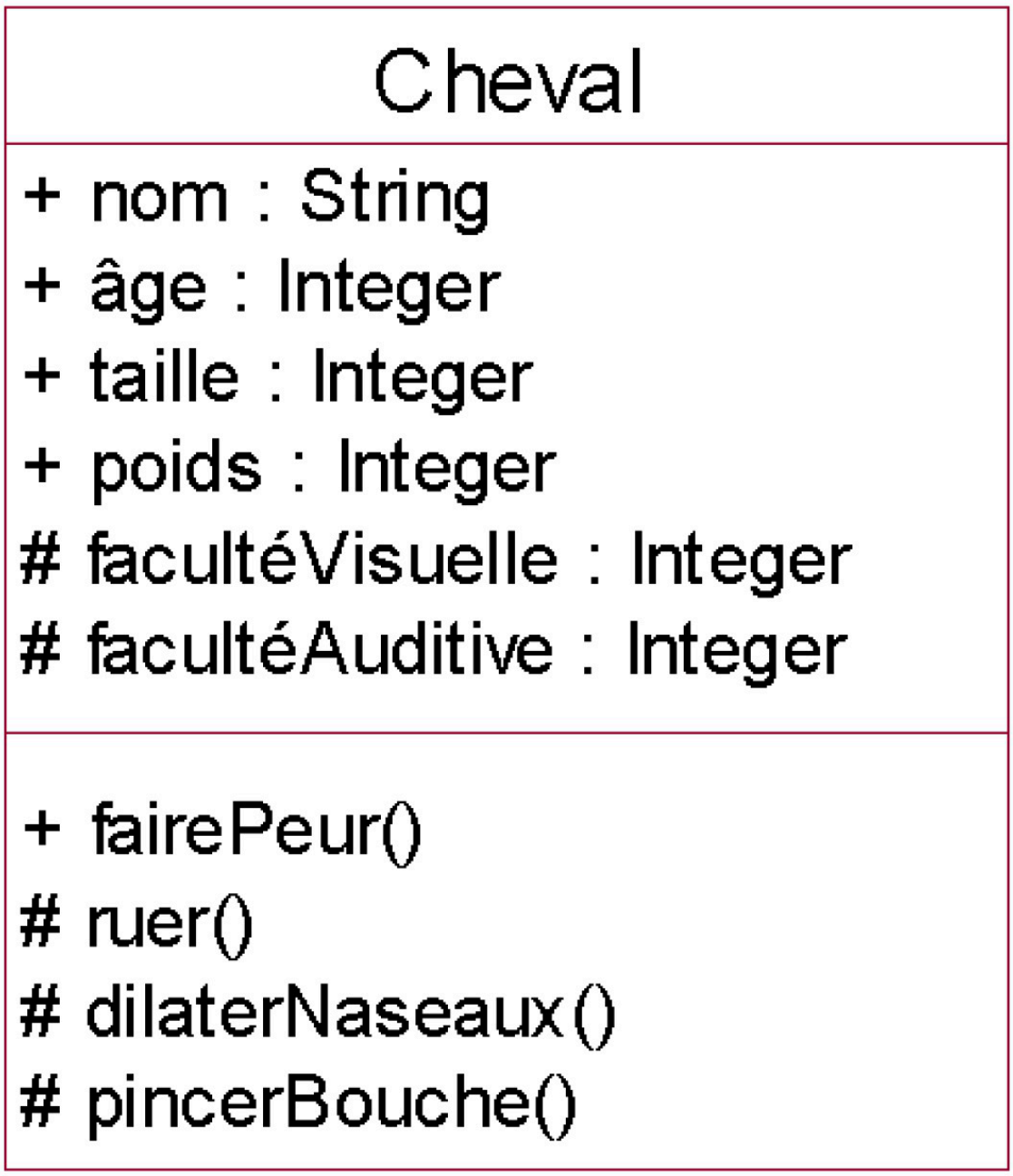


Figure 4 : La classe Cheval avec attributs typés et les caractéristiques d'encapsulation

1.2. UML : Les relations entre classes

Un diagramme de classe a vocation à représenter toutes les classes d'un système ou d'un sous système. Entre toutes ces classes, il existe généralement des relations. Un des points forts d'UML est qu'il permet de représenter très finement ces relations, selon leur nature et les contraintes qui les régissent.

1.2.1. L'héritage

Définition

L'héritage (que l'on appelle parfois subsomption, relation taxonomique ou encore relation verticale) est un concept très important de l'approche objet.

On dit qu'une classe hérite d'une autre si elle en reprend toutes les caractéristiques (attributs et méthodes), en les complétant éventuellement avec de nouvelles caractéristiques

C'est un concept relativement simple à comprendre, notamment en utilisant le qualificatif *« est un »* si je peux dire « *B* est un *A* », alors *B* hérite de *A* (*A* est la classe mère de *B*).

Exemple

Étudiant est un(e) *Personne*, *Voiture* est un *Véhicule*, *Roman* est un *Livre*.

En UML, l'héritage est représenté par une *ligne terminée par une flèche pleine* du côté de la classe mère :

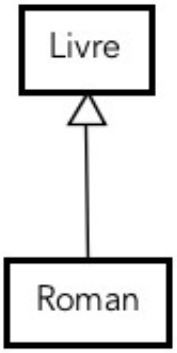


Figure 2 : Exemple d'héritage

Définition : héritage multiple

Il est également possible qu'une classe hérite de plusieurs classes à la fois : on appelle ceci l'héritage multiple, la classe fille hérite alors de l'ensemble des caractéristiques de ses classes mères.

La représentation est bien sûr possible en UML :

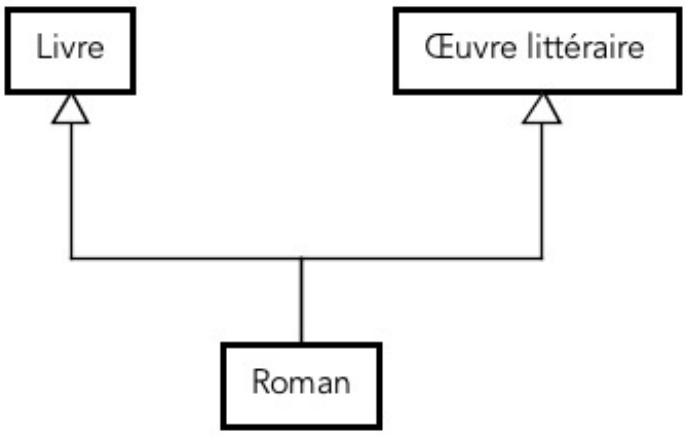


Figure 3 : Héritage multiple

a) Les relations horizontales

Définition

Certaines classes sont liées par des relations qui n'impliquent pas d'héritage de caractéristiques, mais simplement une sorte de collaboration dans le système

Attention

Les relations dites « *horizontales* » peuvent très bien être représentées par des *traits verticaux* ! Pensez à une relation hiérarchique du genre « Mon boss est au dessus de moi » : oui, mais votre boss peut très bien être assis à côté de vous.

i La relation simple

Définition

La relation simple est la plus générale de toutes, c'est aussi la moins « forte » : elle permet de lier des classes sans impliquer de modification dans les classes elles-mêmes.

On peut ainsi spécifier des choses telles que : *« Un livre a pour auteur une personne »*.

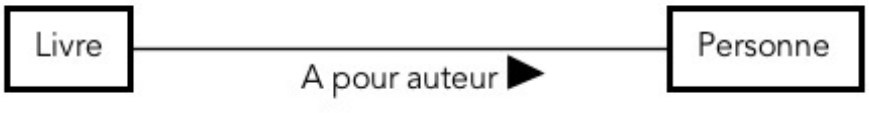


Figure 4 : La relation simple

Ce que nous venons de modéliser est intéressant, mais encore trop imprécis. En effet, il serait intéressant de pouvoir spécifier qu'un livre peut être écrit par plusieurs auteurs, qu'un livre a au moins un auteur, que toutes les personnes n'ont pas forcément écrit de livres...

Le diagramme suivant pourrait alors s'énoncer comme suit : *« Un livre a pour auteur au moins une personne, et une personne peut être l'auteur d'aucun, un ou plusieurs livres »*

Figure 5 : Relation simple avec cardinalité

Quelques exemples de cardinalité

Notation	Signification
1	Exactement un
*1. .**	Un ou plusieurs
*0. .**	Zéro ou un
*	Zéro ou n'importe quel autre nombre (équivalent à *0..**)

L'agrégation

La relation d'agrégation est une relation de type *« ensemble/élément »* qui représente un couplage fort entre deux classes.

Ceci étant dit, les cycles de vies de l'agrégat et de ses éléments agrégés sont indépendants : si je supprime l'agrégat, l'agréger peut continuer à vivre dans le système

En UML, l'agrégation est représentée par un *losange vide*.

La notion d'agrégation entre un livre (*agrégat*) et une personne (*élément agrégé*), comme le montre la figure ci dessous.

Figure 6 : Agrégation

Remarque

Cette agrégation est donc *« plus forte »* qu'une simple relation,

avec une nuance importante : si un livre est supprimé du système, ses auteurs peuvent continuer à y vivre (en d'autres termes, ils ne seront pas supprimés automatiquement, ils ne font pas « partie » du livre, mais y sont seulement agrégés).

La composition

Définition

La composition est la relation la plus forte entre deux classes : la classe composante « fait partie » de la classe composée. Ceci a un impact tout aussi fort : si l'on supprime une instance de la classe composante, alors les instances liées de la classe composée sont automatiquement supprimés.

En UML, la composition est représentée par un *losange plein.*

Figure 7 : la composition

Exemple

Un livre est composé de pages : si je supprime un livre, ses pages sont nécessairement supprimées avec lui (je ne peux pas récupérer ces pages pour les mettre dans un autre livre).

2. Exercice : Exercices

[Solution n°1 p 20]

Exercice : Exercice 01

Parmi les propositions suivantes, lesquelles sont valides ?

- ☐ Le signe + permet de caractériser un attribut publique.
- ☐ Le signe + permet de caractériser une méthode statique.
- ☐ Le signe + permet de caractériser une méthode publique.
- ☐ Le signe − permet de repérer une classe abstraite.

Exercice : Exercice 02

Comment décrire la visibilité en UML ?

- ☐ + public, − protected, # private
- ☐ + public, # private, − protected
- ☐ + public,
- ☐ # public, + protected, − private

Exercice : Exercice 03

*0..**

- ☐ Est la description d'une cardinalité
- ☐ Est équivalent à *
- ☐ N'est pas une cardinalité valide en UML
- ☐ Est équivalent à *..0
- ☐ Est équivalent à 1..*

Exercice : Exercice 4

Une classe est .....

- ☐ Un composant physique
- ☐ Un modèle
- ☐ Un diagramme
- ☐ Un concept

Le diagramme d'objets

II

Objectifs

A la fin de cette section vous serez capable de définir et comprendre les notions ci dessous:

Le diagramme des classes est une représentation statique du système. Le diagramme des objets montre, à un moment

donné, les instances créées et leurs liens lorsque le système est actif.

1. Rappels

Rappel : Définition d'une classe

Un objet est une instance d'une classe, c'est à dire une élément créé selon le « moule » qu'est sa classe.

1.1. Instanciation

Définition

L'instanciation est le mécanisme par lequel on crée des objets sur la base d'un modèle, la classe.

Chaque instance est représentée dans un rectangle qui contient son nom en style souligné et éventuellement, la valeur d'un ou de plusieurs attributs.

- Le nom d'une instance est de la forme : `nomInstance : nomClasse`

Le nom de l'instance est optionnel.

- La valeur d'un attribut est de la forme : `nomAttribut = valeurAttribut`

Enfin, les liens entre instances sont représentés par de simples traits continus.

1.2. Représentation des objets en UML

La représentation des objets en UML ressemble à celle des classes : l'objet est représenté par un rectangle, contenant le nom de l'objet suivi du signe « : » et du nom de sa classe, le tout étant souligné d'un trait.

Figure 7 : Objet en UML

Il est également possible de représenter un objet sans lui donner de nom, en omettant simplement ce nom

(pour éviter les *« monObjet »*, *« objet_1 »*...) :

Figure 8 : Objet anonyme

On peut également, au besoin, faire apparaître des valeurs d'attributs pour un objet donné.

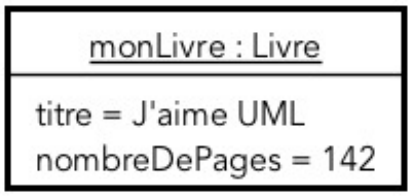


Figure 9 : Objet avec attributs

1.3. Liens entre objets

Dans le diagramme de classe, des relations d'association sont définies entre les classes. C'est tout logiquement que ces relations s'instancient elles aussi, pour relier les instances de classes entre-elles.

Reprenons l'un de nos premiers diagrammes de classe :

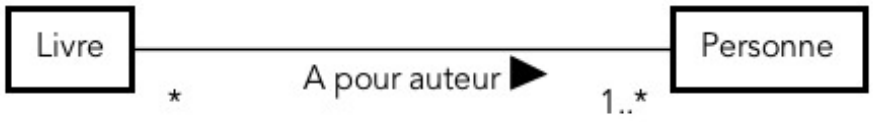


Figure 10 : Relation simple cardinalité

Vous vous souvenez que ce diagramme stipule les faits suivants : dans le système, tout livre a pour auteur au moins une personne, et une personne peut être l'auteur de livre(s) ou pas.

Nous allons pouvoir instancier ces classes et faire apparaître les liens entre les objets ainsi créés :

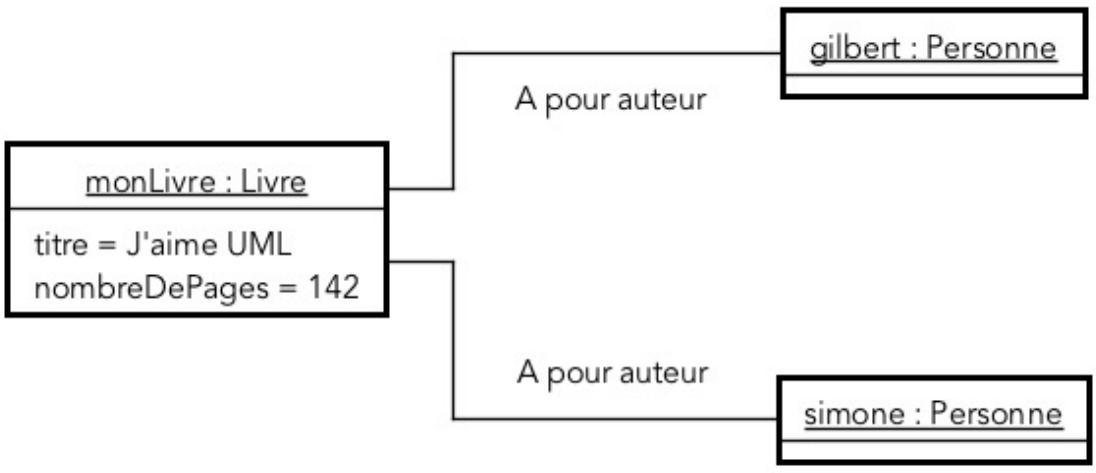


Figure 11 : Les liens entre les objets

1.4. Objets composites

Nous avons vu dans la leçon sur les diagrammes de classes un type de relation intéressant : *la*

composition

Dans ce cas de figure, on spécifie que les objets d'une classe sont « composés » d'autres objets.

Leur cycle de vie est alors le même (si je supprime l'objet composite, je supprime les composants !).

Dans ce cas de figure, il est possible d'utiliser deux représentations.

D'une manière classique, les liens apparaissent comme des « instance » des relations d'association :

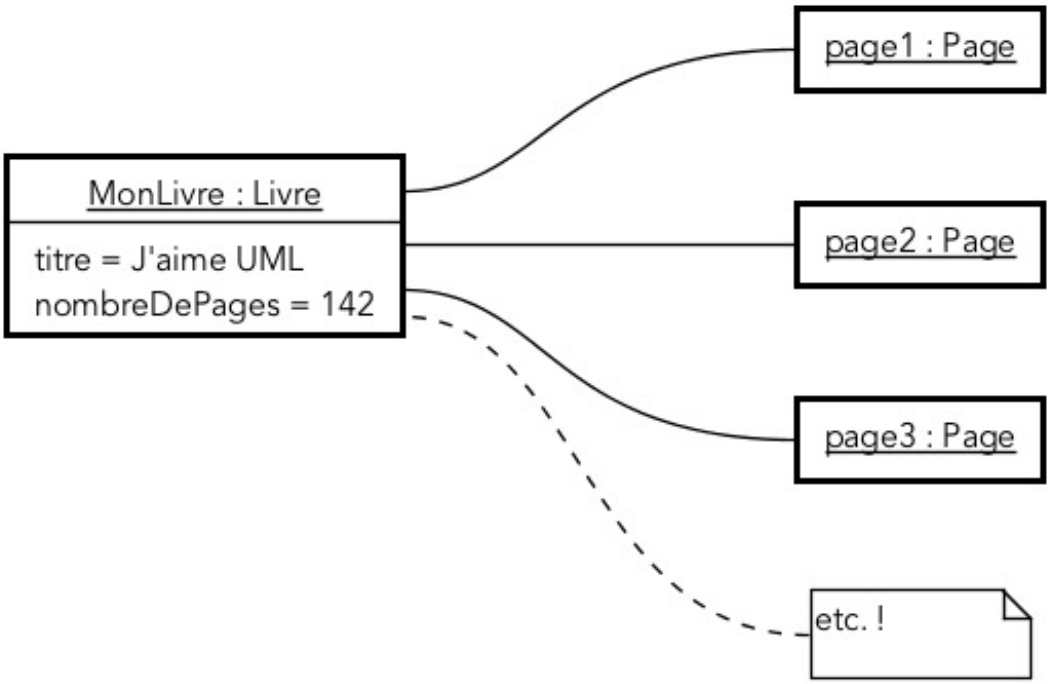


Figure 12 : Objet compose

Ou de manière simplifiée, les objets composants sont intégrés à l'objet composite :

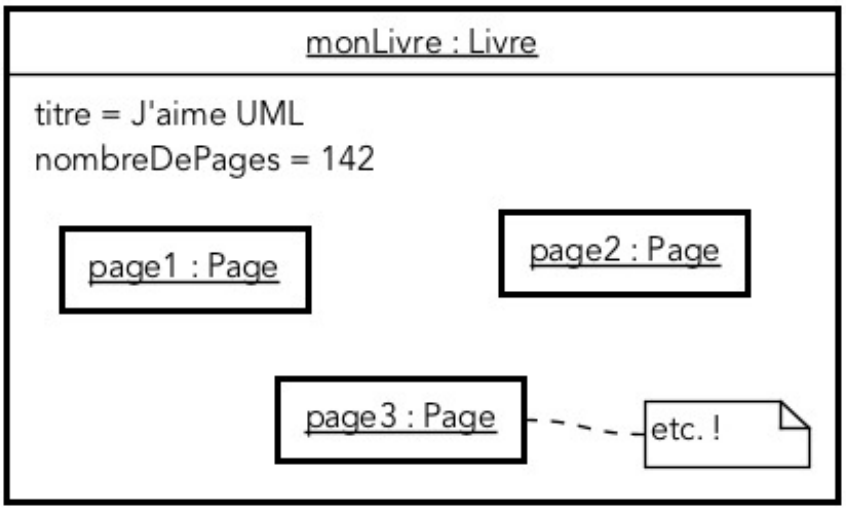


Figure 13 : Diagramme de structure composite

2. Exercice : Exercices

[Solution n°2 p 20]

Exercice : Exercice 01

Un diagramme d'objet permet de représenter

- [ ] Toutes les instances envisageables pour une classe donnée
- [ ] Des exemples d'instances de classes définies préalablement dans un diagramme de classes
- [ ] Des objets
- [ ] Des instances
- [ ] Les cas d'utilisation du système

Exercice : Exercice 02

Les liens qui relient les objets d'un diagramme d'objets sont...

- [ ] Des instances des associations qui relient les classes du modèle
- [ ] Des liens invalides
- [ ] Des relations de composition entre objets
- [ ] Des relations d'héritage entre objets

Exercice : Exercice 03

Les diagrammes d'objets sont pratiques pour...

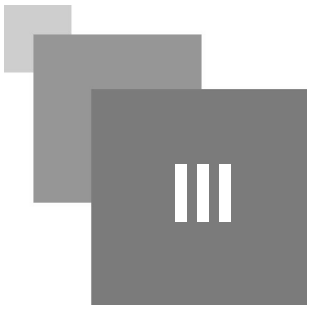
- [ ] Représenter des objets particuliers, comme « Jacky » ou « mon_objet_1 »
- [ ] Représenter le comportement d'un acteur
- [ ] Disposer d'une vue concrète présentant des exemples d'instanciations des classes du diagramme de classes
- [ ] Identifier les concepts du domaine

Exercice : Exercice 03

Le principe d'abstraction est permis par quel mécanisme ?

- ◯ Héritage
- ◯ Encapsulation
- ◯ Prototypage dynamique

Le diagramme des composants



Objectifs

A la fin de cette section vous serez capable de définir et comprendre les notions ci dessous:

Le diagramme de composants permet de décrire les composants du systèmes et les relations qui existent entre-eux.

1. Des composants physiques logiciels de différentes natures...

Définition : Les composants

Un composant est une unité logicielle offrant des services au travers d'une ou de plusieurs interfaces. C'est une boîte noire dont le contenu n'intéresse pas ses clients.

Le mot *composant* est une sorte de mot valise dans le domaine de l'informatique : on l'utilise ici et là, pour décrire des choses très différentes. Les composants peuvent être de natures très variées...

En effet, un composant peut être :

- Un programme
- Un logiciel regroupant un ensemble de programmes
- Un fichier binaire
- Un code source

Les *diagrammes de composants* UML permettent de représenter toutes ces choses.

2. Représentation d'un composant

Un composant est représenté par un rectangle contenant son nom et un petit pictogramme,

Exemple

- représentation d'un back office de gestion des clients

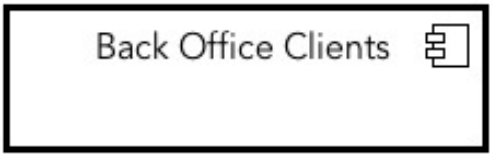


Figure 14 : Exemple de composant

3. Dépendances entre composants

il est possible de spécifier des relations de dépendance entre composants. comme le montre la figure

qui décrit une dépendance , sans détailler sa nature, du composant Back Office Client vis-à-vis du composant SSO

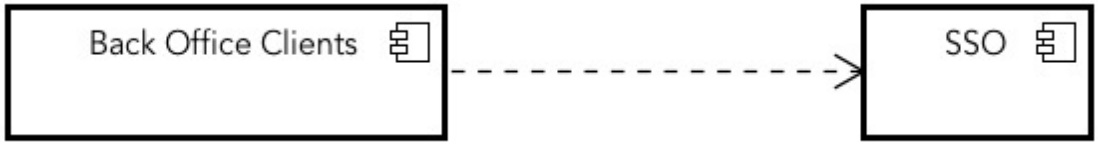


Figure 15 : Exemple de dépendance entre composants

4. Les interfaces

Définition

- Les *interfaces* permettent aux composants communiquent entre eux.

UML permet de représenter simplement ces interfaces, qui peuvent être :

- Des *interfaces offertes* (fournies par le composant)
- Des i*nterfaces requises* (dont le composant a besoin pour fonctionner)

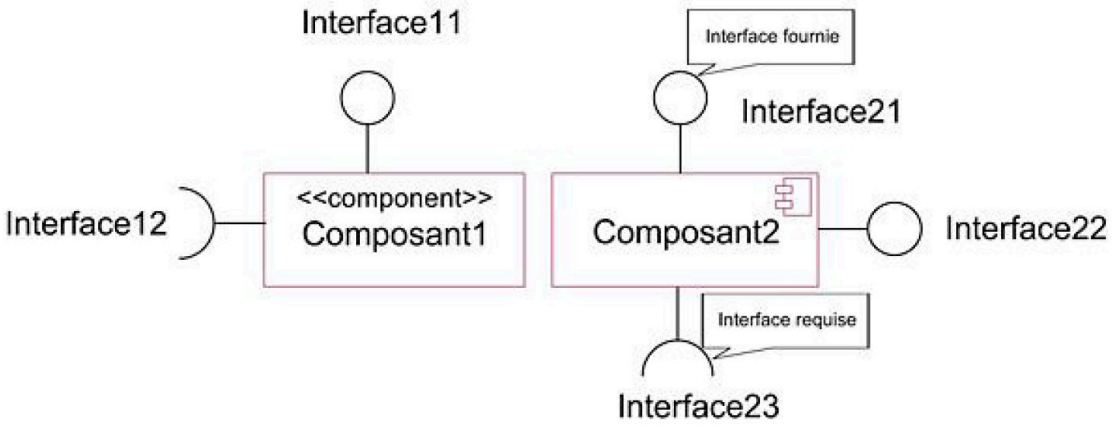


Figure 16 : Représentation graphique d'un composant et de ses interfaces

- La définition de l'encapsulation se fait au niveau de la classe. Les objets extérieurs à un objet sont donc les instances des autres classes.

L'architecture logicielle par composants

Dans l'approche par objets, l'architecture logicielle d'un système est construite par un assemblage de composants liés par des interfaces fournies et des interfaces requises. Le diagramme des composants décrit cette architecture.

Exemple

Le logiciel de gestion d'un élevage de chevaux est réalisé par assemblage de composants. La figure ci dessous illustre le diagramme des composants de ce système. Les composants *FenêtreGestionChevaux* et *FenêtreGestionVentes* gèrent respectivement à l'écran une fenêtre consacrée à la gestion des chevaux et à la gestion des ventes de chevaux. Le composant *SystèmeBaseDonnées* est un système de base de données

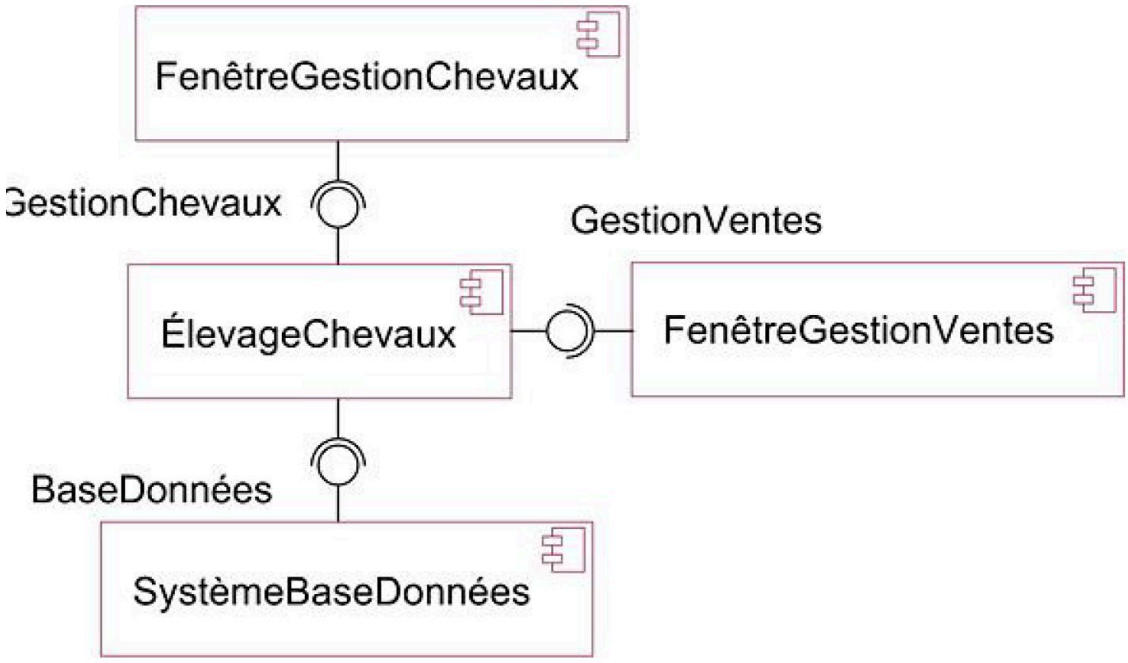
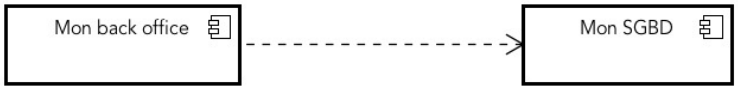


Figure 17 : Exemple de diagramme de composant

5. Exercice : Exercices

[Solution n°3 p 21]

Exercice : Exercice 01



- ☐ Le composant « Mon back office » a une dépendance vis-à-vis de « Mon SGBD ».
- ☐ Le composant « Mon SGBD » a une dépendance vis-à-vis de « Mon back office ».
- ☐ void
- ☐ Mon back office hérite de Mon SGBD.
- ☐ Ce diagramme est invalide.

Exercice : Exercice 02

Une interface requise...

- ○ Nécessite une interface offerte
- ○ Est représenté par un lien d'association
- ○ Décrit une dépendance du composant

Solutions des exercices

> Solution n°1

Exercice p. 12

Exercice 01

- [x] Le signe + permet de caractériser un attribut publique.
- [ ] Le signe + permet de caractériser une méthode statique.
- [x] Le signe + permet de caractériser une méthode publique.
- [ ] Le signe − permet de repérer une classe abstraite.

Exercice 02

- [ ] + public, − protected, # private
- [ ] + public, # private, − protected
- [x] + public,
- [ ] # public, + protected, − private

Exercice 03

- [x] Est la description d'une cardinalité
- [x] Est équivalent à *
- [ ] N'est pas une cardinalité valide en UML
- [ ] Est équivalent à *..0
- [ ] Est équivalent à 1..*

Exercice 4

- [ ] Un composant physique
- [ ] Un modèle
- [ ] Un diagramme
- [x] Un concept

Exercice p. 16

> Solution n°2

Exercice 01

- [x] Toutes les instances envisageables pour une classe donnée
- [x] Des exemples d'instances de classes définies préalablement dans un diagramme de classes
- [ ] Des objets
- [ ] Des instances
- [ ] Les cas d'utilisation du système

Exercice 02

- [x] Des instances des associations qui relient les classes du modèle
- [ ] Des liens invalides
- [ ] Des relations de composition entre objets
- [ ] Des relations d'héritage entre objets

Exercice 03

- [x] Représenter des objets particuliers, comme « Jacky » ou « mon_objet_1 »
- [ ] Représenter le comportement d'un acteur
- [ ] Disposer d'une vue concrète présentant des exemples d'instanciations des classes du diagramme de classes
- [ ] Identifier les concepts du domaine

Exercice 03

- (•) Héritage
- ( ) Encapsulation
- ( ) Prototypage dynamique

> Solution n°3

Exercice p. 19

Exercice 01

- [x] Le composant « Mon back office » a une dépendance vis-à-vis de « Mon SGBD ».
- [ ] Le composant « Mon SGBD » a une dépendance vis-à-vis de « Mon back office ».
- [ ] void
- [ ] Mon back office hérite de Mon SGBD.
- [ ] Ce diagramme est invalide.

Exercice 02

- (●) Nécessite une interface offerte
- ( ) Est représenté par un lien d'association
- ( ) Décrit une dépendance du composant