UML

Baptiste Bauer

Version v0.0.2.sip-221121191720, 2022-11-21 19:05:50

Table des matières

1. A lire avant de commencer

Ce support contient de nombreux extraits de code PHP.

Vous allez voir régulièrement des renvois numérotés dans le code. Ces renvois ne font pas partie du langage. Il ne faut donc pas les copier.

Voici l'illustration d'un renvoi :

```
1 class UneClassePhp {
2
3     public function uneMethode():void{ ①
4         //du code qui fait de la magie ②
5     }
```

① Je suis du texte qui est lié au numéro de renvoi. Je ne fais pas partie de la syntaxe du langage PHP.

② Je suis un autre renvoi.

Ces renvois permettent de cibler des lignes afin d'apporter des explications spécifiques. Il est important de les lire.

Le langage utilisé dans ce support est le langage PHP dans sa version 8.1.

Bonne lecture !

2. Représentation d'une classe avec un diagramme de classes UML

2.1. Représentation générique

La **représentation d'une classe** est formalisée par un rectangle découpé en trois parties du haut vers la bas :

1. le nom de la classe
2. les attributs de la classe (également appelés membres ou propriétés)
3. les opérations de la classe (également appelées membres ou méthodes)

2.2. Précisions sur la visibilité des membres

- Le signe `-` désigne une visibilité privée. Le membre n'est accessible que depuis l'intérieur de la classe.
- Le signe `#` désigne une visibilité protégée. Le membre n'est accessible que depuis la classe et ses héritières.
- Le signe `+` désigne une visilité publique. Le membre est accessible depuis la classe et en dehors.

2.3. Précisions sur la notion de type

- Le type correspond aux types des attributs et des valeurs retournées par les opérations
- Exemples de type : int, bool, string, float, array, List, etc (tout dépend du langage de programmation utilisé)
- Le type peut être le nom d'une classe puisqu'une classe revient à définir un type.

2.4. Précisions sur la notion d'opération

Une opération est tout simplement une méthode. Son nom doit être réfléchi afin d'exprimer ce qu'elle fait. Par exemple, une méthode qui calcule l'âge d'une personne pourrait s'appeler « calculerAge » ou encore mieux « obtenirAge ». Une méthode qui vérifie qu'une personne est majeur (donc soit c'est vrai, soit c'est faux) pourrait s'appeler « **est**Majeur ».

Il est fortement recommandé d'utiliser l'anglais pour nommer les membres.

2.5. Précisions sur la notion de direction

Cette notion de direction est pertinente dans des langages compilés tels que `C#` ou encore `Java`. En `PHP` et `javascript`, il n'y a pas la possibilité de spécifier une direction à un paramètre.

- `in` : direction par défaut, la variable passée comme argument restera inchangée dans le programme appelant (même si dans l'opération, sa valeur a été modifiée). Elle est utilisée à l'intérieur de l'opération sans être modifiée. Cela prend le nom de passage par valeur.
- `inout` : une modification de la variable passée comme argument dans l'opération se verra également modifiée dans le programme appelant. (c'est ce qui s'apparente à un passage par référence).

Les objets passés en argument le sont automatiquement par référence. Pour les types qui ne le sont pas par défaut (exemples : le type « int » en C#, une chaîne en PHP) et qui doivent l'être dans la méthode, il faut indiquer « `inout` » dans le diagramme.

En PHP, le signe `&` représente ce type de passage alors qu'en C#, c'est le mot clé `ref`.

- `ref` : idem à inout (les logiciels de modélisation proposent généralement `inout`).
- `out` : précise la variable que l'opération doit retourner afin que le programme appelant puisse l'utiliser. Celui-ci doit avoir prévu sa déclaration. Ce terme sera utilisé dans le chapitre sur les procédures stockées.

Si un objet est utilisé comme argument d'une opération, il est TOUJOURS PASSE PAR REFERENCE.
Cela revient à utiliser par défaut « inout » ou « ref »

2.6. Précision sur la notion de stéréotype

Le stéréotype permet d'étendre le vocabulaire de l'UML.

Le **stéréotype d'une classe** permet une meilleure compréhension des éléments qui composent une architecture logicielle. Vous pouvez donc utiliser n'importe quel mot qui permet de faciliter la lecture du diagramme.

Voici quelques exemples :

Le **stéréotype d'une opération** (méthode) peut permettre de la classer dans une catégorie de comportement.

2.7. Avec UML, on affiche que ce qui est essentiel

Un diagramme UML peut rapidement devenir complexe à lire. Certaines classes peuvent avoir de nombreux membres. En fonction du destinataire de l'information, il est possible de ne montrer que l'essentiel.

Imaginons une application qui laisse la possibilité à celui qui l'utilise de créer un ou plusieurs menus de navigation qui pourront être placés à des endroits spécifiques de la fenêtre (en haut, à droite, en bas ou à gauche)

Le diagramme ci-dessous peut être utile au développeur car il sait exactement ce qu'il doit

développer :

Cependant, il n'y a pas besoin de connaître les méthodes pour créer la table correspondante. Le diagramme ci-après est suffisant :

Lors de la réflexion sur les différentes classes à créer, seule les nom des classes peuvent être affichés

L'utilisateur d'une classe n'a besoin de connaître que ce qu'il peut utiliser (donc les membres publiques) :

Comme vous pouvez le constater, il n'y a pas qu'une seule façon de représenter un même diagramme. Le degré de précision de la conceptualisation dépend de la volonté de transmettre plus ou moins d'informations. Il convient de s'interroger sur le destinataire du diagramme de classes.

3. Quels outils pour réaliser des diagrammes de classes ?

3.1. Quelques outils UML

Il existe de nombreux outils pour réaliser des diagrammes de classe.

- **Dia** (https://dia.fr.softonic.com/) : logiciel gratuit, léger et simple à prendre en main.
- **Argouml** (https://argouml.fr.uptodown.com/windows). Logiciel gratuit assez simple à prendre en main.
- **Staruml** (https://staruml.io/). Logiciel agréable à utiliser et bénéficiant d'une interface assez pratique (mais payant après une période d'essai).
- et de nombreux logiciels payants (windesign,...)
- **PlantUml** (https://plantuml.com/fr/). C'est plus un moteur de rendu qui permet de générer des diagrammes uml à partir de lignes de texte. C'est gratuit.

3.2. PlantUml

PlantUml est l'outil que je préconise car il permet de faire évoluer rapidement les diagrammes. Il permet de générer de nombreux diagrammes (https://plantuml.com/fr/).

Sa particularité tient dans le fait qu'il n'y a aucune interface graphique. Un diagramme est réalisé à partir de lignes de "code" qui décrivent ce que le moteur de PlantUml doit "dessiner".

Comme il s'agit de lignes de "code", il est très facile de versionner ses modélisations.

Q1) Travail à faire

- Chargez la documentation de Plant Uml concernant les diagrammes de classe (https://plantuml.com/fr/class-diagram)
- Chargez la page qui permet de réaliser des diagrammes (http://www.plantuml.com/plantuml/uml/)
- Réalisez directement dans l'éditeur ouvert précédemment le diagramme de classes correspondant à ces besoins :
  - Il faut modéliser deux classes dont les instances vont être persistées en base de données. Il y a deux objets à conceptualiser.
    - Le premier est un employé qui est caractérisé par un numéro unique, un nom, une date de naissance. Il doit être possible de retourner l'âge d'un employé en années entières.
    - Le second objet est une entreprise caractérisée par sa dénomination sociale et les employés qu'elle fait travailler. Une méthode doit retourner le nombre d'employés qu'elle fait travailler.

- La définition des différents attributs doit respecter le principe d'encapsulation.

3.3. Utiliser PlantUml dans son éditeur de code

Les IDE tels que PhpStorm et Visual Studio Code sont capables de rendre les diagrammes une fois que le bon plugin est installé.

Q2) Travail à faire

- En fonction de votre éditeur, cherchez et installez le plugin permettant de rendre des diagrammes écrits pour PlantUML.
- Copiez et collez le code du travail précédent de façon à le prévisualiser dans votre éditeur.

Q3) Travail à faire

- A l'aide du langage php, implémentez la classe suivante :
- Une fois la classe implémentée, testez-la de façon à créer deux animaux.
- Afficher le nom du premier objet animal avec un echo directement appliqué sur l'objet afin d'appeler automatiquement la méthode `__toString`
- Afficher le nom du second objet animal en utilisant la méthode `getName()`.

4. Faire des liens entre les classes

Le diagramme de classes **permet de mettre en évidence les liens entre les différentes classes** composant une application.

La **logique sémantique d'architecture de l'application** peut ainsi être lue. C'est-à-dire que l'on peut comprendre le rôle de chaque classe dans l'application et le ou les liens qu'elles ont entre elles. C'est également très utile pour le développeur car il sait exactement ce qu'il doit "coder".

Voici un extrait d'un diagramme de classes :

[extrait diagramme de classes] | *../images/extrait-diagramme-de-classes.png*

Les classes sont reliées entre elles avec parfois des traits continus, des traits pointillés, des flèches, etc. Chaque lien a une signification particulière. La suite de ce cours va vous permettre de déterminer quel lien utiliser pour relier des classes entre elles et comment implémenter ces liaisons.

5. Le lien associatif : l'association

L'**association** désigne un lien entre deux objets A et B sachant que A **contient** une ou des instances de B et/ou que B **contient** une ou des instances de A.

Une association exprime une **relation de contenance**. A prévoit un **attribut qui stocke** une ou plusieurs instances de B et / ou vice-versa.

Ce lien est représenté par un trait continu entre les classes dont les objets sont liés.

Modélisons le fait qu'un véhicule est entretenu par un technicien :

Grâce au lien, nous savons que `Vehicle` est entretenu par `Technician`. S'il n'y avait aucun lien, cela signifierait que `Vehicle` n'utilise pas `Technician` et vice-versa. Ces deux classes n'auraient alors aucune interaction l'une avec l'autre, comme deux personnes qui vivraient à 1000km l'une de l'autre sans même connaître l'existence de l'autre.

Les représentations suivantes sont possibles :

Réprésentation avec un lien et du "texte"

Représentation qui précise le sens de lecture de la relation :

Représentation avec des **terminaisons d'association** :

Une **terminaison d'association** est un attribut de la classe liée. Pour mieux comprendre, la représentation précédente peut être modélisée ainsi :

Il n'est pas obligatoire de nommer la relation :

Nous pouvons tout à fait ne laisser que les classes :

L'association peut être représentée sans le lien mais en précisant les attributs pertinents dans chacune des classes liées :

Ce qu'il faut bien comprendre, c'est qu'une association conduit l'entité liée à **contenir** une ou plusieurs instances de l'entité liée. Nous reviendrons sur ce point à plusieurs reprises dans la suite du cours.

6. Les cardinalités d'une association

Des **cardinalités** peuvent être ajoutées afin d'exprimer une **multiplicité du lien associatif entre deux classes**. **C'est utile lorsque l'on souhaite indiquer qu'une instance de classe peut être liée (sémantiquement) à plusieurs instances d'une autre classe.**

Dans le cas ci-après, il est impossible de savoir si plusieurs techniciens entretiennent un même véhicule ou si un véhicule est entretenu par plusieurs techniciens.

Les règles de gestion à exprimer sur le diagramme sont les suivantes :

- Un technicien entretien zéro, un ou plusieurs véhicules.
- Un véhicule est entretenu par au moins un technicien

Voici notre diagramme à jour de ces dernières informations :

Afin de bien comprendre le sens de lecture, voici de nouvelles règles de gestion :

- Un technicien entretien au moins 1 véhicule
- Un véhicule est entretenu par 1 seul technicien

Voici quelques exemples de cardinalités :

Exemple de cardinalité	Interprétation
1	Un et un seul. On n'utilise pas la notation 1..1.
1..*	Un à plusieurs
1..5	1 à 5 (maximum)
1-5	1 à 5 (maximum)
3..7	3 à 7 (maximum)
3-7	3 à 7 (maximum)
0..1	0 ou 1 seul
1,5	1 **ou** 5
1,5,7	1 **ou** 5 **ou** 7
0..*	0, 1 ou plusieurs
*	0, 1 ou plusieurs

Si vous avez l'habitude de faire de l'analyse selon la méthode Merise, vous aurez remarqué que les cardinalités sont inversées par rapport à celles d'UML.

La cardinalité est très utile au développeur pour savoir s'il doit contrôler le nombre d'objets B qu'il est possible d'associer à un objet A.

Q4) Pour chaque diagramme, exprimez la relation en prenant en compte les cardinalités.

a.)

b.)

c.)

Lorsqu'une association exprime un lien vers un maximum de 0 ou une instance de l'objet lié, on parle d'**association simple**.

Lorsqu'une association exprime un lien vers un maximum de plusieurs instances de l'objet lié, on parle d'**association multiple**.

Q5) Pour chaque diagramme, indiquer s'il s'agit d'une association simple ou d'une association multiple en fonction du sens de lecture.

a.)

b.)

c.)

Q6)

Réalisez le diagramme de classes correspondant au domaine de gestion décrit ci-après :

Une entreprise gère des hôtels. Des clients peuvent réserver des chambres dans ces hôtels. Une réservation ne peut porter que sur une seule chambre. Des prestations supplémentaires (petit déjeuner, réveil par l'accueil, encas nocturne) peuvent compléter la mise à disposition d'une chambre. Ces prestations peuvent être prévues lors de la réservation ou ultérieurement. Une chambre est équipé ou non de différentes options (lit simple / double, micro-onde, lit enfant, baignoire de type balnéo, etc)

Les associations doivent être nommées et les cardinalités précisées.

L'implémentation des cardinalités nécessite de savoir implémenter la navigabilité. Nous reviendrons alors sur ce sujet dans la partie sur l'implémentation des cardinalités.

7. La navigabilité d'une association

La **navigabilité** désigne le fait de connaître à partir d'une instance de classe la ou les instances d'une autre classe. Autrement dit, la navigabilité permet de savoir qu'une instance d'une classe A contient une ou des instances de la classe B.

Illustrons ce concept avec ce diagramme :

Cette modélisation nous permet d'affirmer qu'une instance de `Vehicle` contient zéro ou une instance de `Technician`. La classe `Vehicle` doit prévoir un attribut capable de contenir une instance de `Technician`. Nous retrouvons la relation de contenance abordée lors de la découverte de la notion d'association.

Lorsqu'un objet de type `Vehicle` a un attribut qui peut contenir un objet de type `Technician`, on dit que l'on peut **naviguer** de `Vehicle` vers `Technician`.

Le **sens de la navigabilité** doit être explicitement précisé sur l'association. Effectivement, la navigabilité peut être exprimée :

- **dans un seul sens** : de `A` vers `B` OU de `B` vers `A`. Dans ce cas, on parle de **navigabilité unidirectionnelle**.

  La **représentation de la navigabilité unidirectionnelle** est modélisée par une flèche qui pointe l'objet vers lequel il est possible de naviguer.

- **dans les deux sens** : de `A` vers `B` ET de `B` vers `A`. Dans ce cas, on parle de **navigabilité bidirectionnelle**.

  La **représentation de la navigabilité bidirectionnelle** est modélisée par **l'absence de flèches sur le lien associatif**.

Sachez qu'il est possible de trouver une représentation avec une flèche de chaque côté de l'association. Mais ce formalisme est peu utilisé :

Comprendre la navigabilité est indispensable car elle se traduit par du code à écrire dans la classe depuis laquelle on navigue vers l'objet lié.

8. Implémentation des cardinalités

Nous avons appris lors de l'étude de la notion de cardinalité qu'elle permet d'exprimer une contrainte

Dans la partie du cours sur les cardinalités, nous avons vu que les cardinalités exprimait une contrainte sur le nombre d'objets B associés à un objet A.

Le développeur doit tenir compte de celles-ci dans l'implémentation de la classe.

Pour prendre en compte la cardinalité à l'extrémité d'une association navigable, le développeur doit compter le nombre d'instances liées et s'assurer que ce nombre respecte cette cardinalité. En PHP, la fonction `count` retourne le nombre d'éléments dans un tableau (utile pour dénombrer une collection).

Les cardinalités minimale et maximale doivent être vérifiées par le développeur.

Il n'y a aucune difficulté dans le contrôle des cardinalités. Ainsi, vous pouvez attaquer les exercices qui suivent.

Q7) Dans le diagramme ci-dessous, y a-t-il des cardinalités à contrôler ? Si oui, indiquez pour chacune d'elle si le contrôle doit être fait dès l'instanciation de l'objet depuis lequel commence la navigabilité ou après (dans ce cas préciser depuis quelle méthode).

Q8) Dans le diagramme ci-dessous, y a-t-il des cardinalités à contrôler ?

Q9) Dans le diagramme ci-dessous, y a-t-il des cardinalités à contrôler ?

Q10) Dans le diagramme ci-dessous, y a-t-il des cardinalités à contrôler ?

Q11) Implémentez le diagramme suivant :

Q12) Implémentez le diagramme suivant :

Q13) Implémentez le diagramme suivant :

9. L’association réflexive

Une association réflexive est un lien entre deux objets de même type.

Imaginons un technicien qui peut être le supérieur hiérarchique d’autres techniciens. Le diagramme suivant illustre cette relation :

Comme les deux classes mobilisées sont identiques, il ne faut en utiliser qu’une seule et donc faire un lien qui point sur elle-même :

C’est équivalent à cette représentation :

Voici la même modélisation mais avec une bidirectionnalité :

Ce qui est équivalent à :

Q14) Travail à faire

a. Implémentez le diagramme suivant (il n’y a rien de nouveau, cela reste une association bidirectionnelle comme nous savons les implémenter) :
b. Vous veillerez à ce qu’un technicien ne puisse pas être son propre subordonné ou supérieur.

10. L'agrégation

10.1. Qu'est-ce qu'une agrégation ?

Rappel : l'association traduit un lien entre deux objets.

Le terme d'agrégation signifie l'action d'agréger, d'unir en un tout.

L'**agrégation** exprime la construction d'un objet à partir d'autres objets. Cela se distingue de la notion d'association que nous avons abordée jusqu'à maintenant. Effectivement, l'association traduit un lien entre deux objets alors que l'agrégation traduit le "regroupement" ou "l'assemblage" de plusieurs objets. Le lien exprimé est donc plus fort que pour une association.

Imaginez des pièces de Légo que vous utilisez pour construire une maison. Chaque pièce est un objet qui une fois agrégée avec les autres pièces permettent d'obtenir un autre objet (la maison). Il est tout à fait possible d'utiliser chaque pièce pour faire une autre construction. Détruire la maison ne détruit pas les pièces.

Implicitement, l'agrégation signifie « contient », « est composé de ». C'est pour cela qu'on ne la nomme pas sur le diagramme UML.

Autrement dit, une agrégation est le regroupement d'un ou plusieurs objets afin de construire un objet « complet » nommé **agrégat**.

Représentons le lien entre une équipe et les joueurs qui composent celle-ci (ici une équipe de volley) :

Le losange vide est le symbole qui caractérise une agrégation. Il est placé du côté de l'agrégat (l'objet qui est composé / assemblé).

La classe `Team` est l'**agrégat** (le composé de) alors que la classe `Player` est le **composant**.

Ce diagramme objet représente les joueurs qui composent une équipe de volley

Une agrégation présente les caractéristiques suivantes :

- L'agrégation est composée d'"éléments".
- Ce type d'association est non symétrique. Il n'est pas possible de dire "Une équipe est composée de joueurs et un Joueur est composé d'une équipe"
- les composants de l'agrégation sont **partageables**
- l'agrégat et les composants ont leur **propre cycle de vie**)

Les composants sont partageables :

La particularité d'une agrégation est que **le composant peut être partagé**.

Par exemple, un joueur d'une équipe peut jouer (se partager) dans d'autres équipes :

Voici un diagramme objet pour illustrer ce partage :

Les instances de `Player` "aurel" et "clem" sont **partagées** dans deux équipes. Celle représentant "sofian" n'est pas partagée mais peut l'être à un moment donné.

Voici un autre exemple :

L'entreprise est la réunion en un tout de personnes et de locaux. Les personnes qui composent une entreprise peuvent travailler dans d'autres et un local peut servir à plusieurs entreprises. Nous retrouvons la notion de partage des composants.

Comme les composants sont partageables, la multiplicité du côté de l'agrégat peut être supérieur à 1.

[cardinalite multiple agregation] | *../images/cardinalite_multiple_agregation.png*

L'agrégat et les composants ont leur propre cycle de vie

Le cycle de vie d'un objet désigne sa création, ses changements d'état jusqu'à sa destruction.

- Un agrégat **peut** exister sans ses composants. (en programmation, il doit être possible d'instancier un agrégat sans ses composants)
  Une équipe peut exister même s'il elle n'a aucun joueur.
- Un composant **peut** exister sans être utilisé par l'agrégat. (en programmation, il doit être possible d'instancier un composant sans que l'agrégat l'utilise ou existe)
- la destruction de l'agrégat ne détruit pas ses composants (et vice versa) ce qui va dans le sens des deux points précédents

10.2. Navigabilité et agrégation

Tout ce que nous avons vu dans la partie sur la navigabilité s'applique dans le cadre d'une agrégation.

Il y a navigabilité bidirectionnelle entre `Person` et `Enterprise` et navigabilité unidirectionnelle de `Enterprise` vers `Local`.

10.3. Implémentation d'une agrégation

L'implémentation d'une agrégation est exactement la même qu'une association classique.

L'agrégation permet seulement d'exprimer conceptuellement le fait que les instances d'une classe sont des "assemblages" d'autres instances de classe. Une conceptualisation est une représentation de la réalité. Cela peut aider à mieux cerner la logique métier de l'application.

Index