CONCEPTION des SYSTÈMES d'INFORMATION

UML

Cours n°1

MODELISATION OBJET PREMIERE APPROCHE

Epitech 3

Bertrand LIAUDET

SOMMAIRE

v0 : printemps 2009

Première édition : automne 2009

Bibliographie

La bibliographie complète se trouve dans l'introduction générale du cours (au début de l'analyse fonctionnelle).

- **Penser objet avec UML et Java**, Michel Lai, Dunod, 2000.

  Apprentissage de la conception objet et de l'UML associé avec des exemples concrets en Java. Très bonne introduction concrète.

MODELISATION OBJET

PREMIERE APPROCHE

1. L'objet et sa classe

Dans le monde réel, on trouve des objets statiques (une table, une statue, etc.) et des objets coopératifs (une téléviseur, un robot-mixeur, etc.).

Exemple du téléviseur

Caractéristiques communes : le téléviseur abstrait

Tous les téléviseurs n'ont pas les mêmes caractéristiques, mais tous les téléviseurs ont des **caractéristiques communes**. Ces caractéristiques sont de trois sortes :

- les caractéristiques de **ce qu'il est**
- les caractéristiques de **ce qu'il fait** (= de ce qu'il sait faire, =de ce qu'on peut en faire, =des services qu'il peut rendre, =des services qu'on peut lui demander)
- les caractéristiques de ses **relations avec le monde extérieur**

Caractéristiques de ce qu'il est

Tous les téléviseurs ont une marque, un écran, des haut-parleurs, des circuits pour capter et transmettre l'image et le son, etc.

Caractéristiques de ce qu'il fait (les services qu'il rend)

Tous les téléviseurs ont un mécanisme de mise sous tension et d'arrêt, de changement de chaînes, de réglage des images et du son, etc.

Caractéristiques de ses relations (interfaces) avec le monde extérieur

Le téléviseur fonctionne si **la prise de courant** est branchée et si **l'antenne ou le câble** sont connectées.

La télécommande permet d'adresser des signaux de changement d'état à la télévision pour utiliser ses services : allumer, éteindre, monter le son, baisser le son, choisir une chaîne, etc.

Remarques :

Un téléviseur peut être déplacé d'une pièce à l'autre et continuer à fonctionner tant que ses relations avec le monde extérieurs sont maintenues.

L'utilisateur d'un téléviseur doit uniquement savoir brancher la prise de courant, brancher l'antenne ou le câble, utiliser la télécommande. Il ne sait pas comment fonctionne un téléviseur.

Abstrait et concret : classe et objet

Cette description présente le concept de téléviseur. C'est une présentation abstraite (générale, théorique) : il n'est pas question d'un téléviseur réel, concret. Cette description correspond à la notion de **classe** (analogue à la notion de **type** en programmation procédurale).

Tous les téléviseurs concrets sont construit à partir du même plan de base. Ce sont les **objets** (analogue à la notion de **variable** en programmation procédurale).

Gestion des pannes

En cas de panne du téléviseur, on cherche le problème soit au niveau de ses dépendances avec le monde extérieur, soit au niveau des services rendus par le téléviseur.

Idée de la programmation objet

L'idée de la **programmation objet** est de mimer ou **simuler les objets du monde réel**, que ce soit des objets statiques ou des objets coopératifs. Les objets statiques sont des objets qui seront utilisés par d'autres objet

Déterminer quels sont les objets du monde réel à considérer constitue la base de la démarche objet. C'est une démarche radicalement différente de la démarche procédurale qui consiste à séparer les données passives et les instructions de calcul sur ces données. **Le paradigme objet unifie les données et les opérations dans un même module : l'objet.**

La **compréhension d'un objet** est obtenue à travers **les informations qui le caractérise**, mais aussi à travers **les opérations ou les services qu'on peut lui demander**, qu'on appelle encore ses responsabilités. Un objet est défini par un mode d'emploi, c'est-à-dire la liste des opérations qu'on peut appliquer à tous les objets de même nature.

Cette compréhension est formalisée dans un modèle qui est la **classe de l'objet**.

Petite méthode de conception objet

A partir d'un cahier des charges :

- Dans un premier temps on recense les noms communs et les verbes qu'on juge important.
- Dans un second temps, on regroupe **les opérations correspondant à des verbes** avec **les objets désignés par un nom commun**. C'est la que se trouve la principale difficulté. La relation entre l'opération et l'objet doit valoir pour tous les objets de même nature (mettreSousTension () vaut pour tous les téléviseurs). En général, d'un point de vue grammatical, **l'objet est complément d'objet (direct ou indirect) de l'opération (mettre sous tension quoi : le téléviseur, changer la chaîne de quoi : du téléviseur, etc**.).

2. Définition d'une classe

Une classe est un **modèle** à partir duquel on pourra construire les objets. C'est un analogue de la notion de **type**.

Les attributs

Les attributs d'une classe peuvent être vus comme les champs d'un type structuré. Dans la classe, on peut donner des valeurs par défaut aux attributs. Une valeur par défaut peut être non modifiable : ces des constantes. Quand on crée un objet, les valeurs par défaut sont automatiquement affectées aux champs concernés.

De plus, la visibilité et l'accès aux champs peuvent être réglementée. En général, les attributs sont privés : ils ne seront visibles que par les opérations de la classe.

La classe comme type abstrait

Le téléviseur a une alimentation, un ampli audio, un tube vidéo, etc. Ces caractéristiques sont des attributs du téléviseur. Le type de ces attributs est la classe correspondant à un objet alimentation, un objet ampli audio, etc.

Cette approche permet de concevoir de façon à la fois abstraite et concrète : l'abstraction de la classe Alimentation sera finalement réalisée.

Les opérations

Une classe contient des opérations. Les opérations sont des procédures et des fonctions.

Distinction entre en-tête et implémentation d'une opérations

- **L'en-tête** décrit les paramètres en entrée et en sortie de l'opération, c'est-à-dire la façon dont on peut utiliser une opération. On parle de **spécifications d'une opération**.
- **Le corps** de l'opération est son implémentation.

Dans un premier temps, on ne précise que l'en-tête des opérations.

Distinction entre paramètres publiques et paramètres privés

- **Les paramètres privés** sont les attributs de la classe de l'opération. Ils n'apparaissent pas dans l'en-tête.
- **Les paramètres publics** ne sont pas attributs de la classe. Ils apparaissent dans l'en-tête.

Distinction entre opérations publiques et des opérations privées

- **Les opérations publiques** représentent les fonctionnalités, ou responsabilités, ou services offerts par la classe. Elles disent à quoi sert la classe. Elles permettent de comprendre ce qu'est la classe. Elles permettent d'utiliser la classe. Les opérations publiques sont visibles par tout le monde.
- **Les opérations privées** ne sont pas utiles pour le monde extérieur. Ce sont des opérations qui servent aux opérations publiques.

Dans un premier temps, on ne précise que les opération publiques.

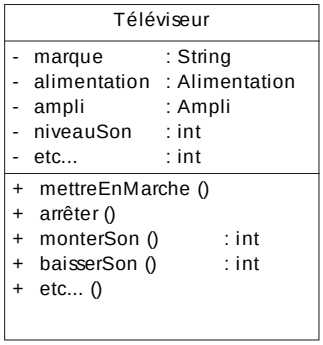
Distinction entre fonctionnel et organique

Les 3 distinctions précédentes sont des distinctions entre le fonctionnel et l'organique (le quoi et le comment, l'externe et l'interne, le visible et le caché, le public et le privé).

Dans un premier temps, on s'intéresse à l'en-tête des opérations publiques en se limitant aux paramètres formels qui ne sont pas attribut de la classe de l'opération.

Formalisme UML

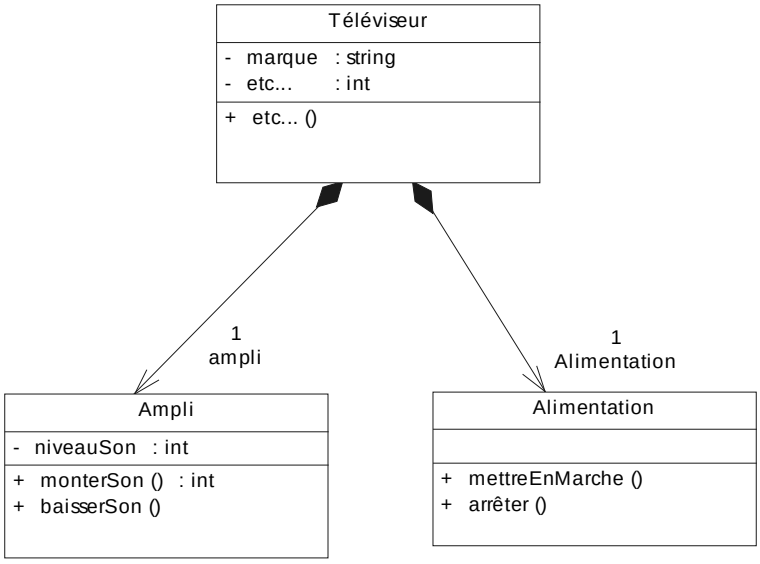
Classe avec des attributs de type Classe



- le nom des attributs et des opérations commence par une minuscule.
- Les types simple commencent par une minuscule (int), les types abstraits commencent par une majuscule (Alimentation, String, etc.)
- Les attributs sont privés : « - » devant
- Les opérations sont publiques : « + » devant.

Composition à la place des attributs de type classe

Le modèle suivant est équivalent au précédent.



3. Programmation objet vs. programmation procédurale

Principe de la distinction

Approche procédurale ou fonctionnelle : ce que le système fait

- L'approche procédurale propose une méthode de décomposition basée uniquement sur **ce que le système fait**, c'est-à-dire aux fonctions du système **On s'intéresse d'abord aux fonctions puis aux données manipulées par les fonctions**. Les fonctions sont identifiées puis décomposées en sous-fonctions, et ainsi de suite jusqu'à l'obtention d'éléments simples programmables.
- C'est la **méthode cartésienne ou analytique**.
- L'architecture du système est le reflet des fonctions du système. C'est une **architecture hiérarchique**.

Approche objet : démarche systémique : ce que le système est et fait

- L'approche objet propose une méthode de décomposition non pas basée uniquement sur ce que le système fait, mais sur **ce que le système est et fait**. Autrement dit, **on s'intéresse à des ensembles de fonctions associées chacun à un ensemble de données, le tout formant un objet.** L'objet est une unité de services rendus (ses fonctions). Les fonctions et sont identifiées par objet et pour un objet et non plus à partir de l'ensemble du système. Les données sont identifiées par objet et pour un objet et non plus à partir des fonctions ou de l'ensemble du système.
- C'est la **méthode systémique** : les objets collaborent entre eux pour constituer le système. On peut en rajouter ou en supprimer sans perturber les collaborations entre les objets non concernés.
- L'architecture du système est le reflet des objets du systèmes. C'est une **architecture réticulaire**.

Avantages de l'approche objet

- **Stabilité de la modélisation** par rapport aux entités du monde réel.
- Construction itérative facilitée par le **couplage faible** entre les composants.
- Possibilité d'une **réutilisation les composants** d'un développement à un autre.
- **Simplicité du modèle**. 5 concepts fondateurs :
  1. **Objet**
  2. **Message**
  3. **Classe**
  4. **Héritage**
  5. **Polymorphisme**

4. La classe comme interface

2 types d'interface : proposée et utilisée

Interface proposée

La télécommande du téléviseur est l'interface par laquelle on peut utiliser le téléviseur. C'est **l'interface proposée** par l'objet. On peut changer d'interface sans changer de téléviseur. L'interface

La modélisation objet permet de définir une classe spéciale appelée «interface » et qui permet de définir uniquement l'en-tête des opérations. Les interfaces sont des classes abstraites car aucun objet ne les réalisera. Le corps des opérations n'est pas définit non plus. C'est la classe qui réalise l'interface qui définira le corps des opérations.

Dans notre exemple, les usages du téléviseur : allumer, éteindre, monter le son, etc., font partie de son interface. Ils sont réalisée par l'objet téléviseur.

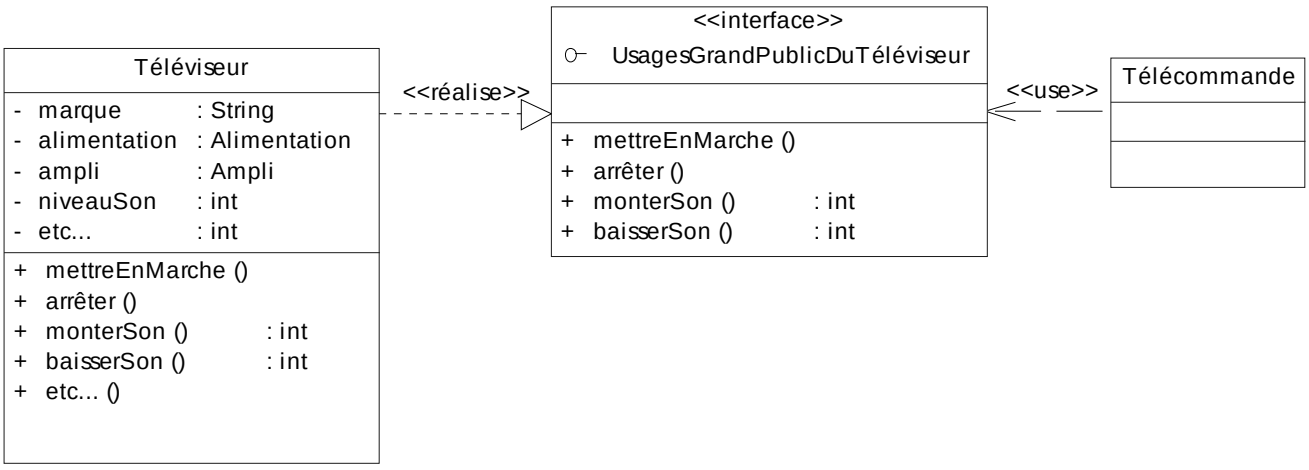
La télécommande est un nouvel objet (qui donne lieu à une nouvelle classe) qui va utiliser l'interface des « usages grand public » du téléviseur.

Interface utilisée

Les prises électriques et les prises câble ou antenne sont les interfaces proposés par les objets « réseau électrique », « réseau de câble » ou « antenne ». Ce sont les **interfaces utilisés** par le téléviseur pour qu'il puisse fonctionner.

Formalisme UML

Interface proposée

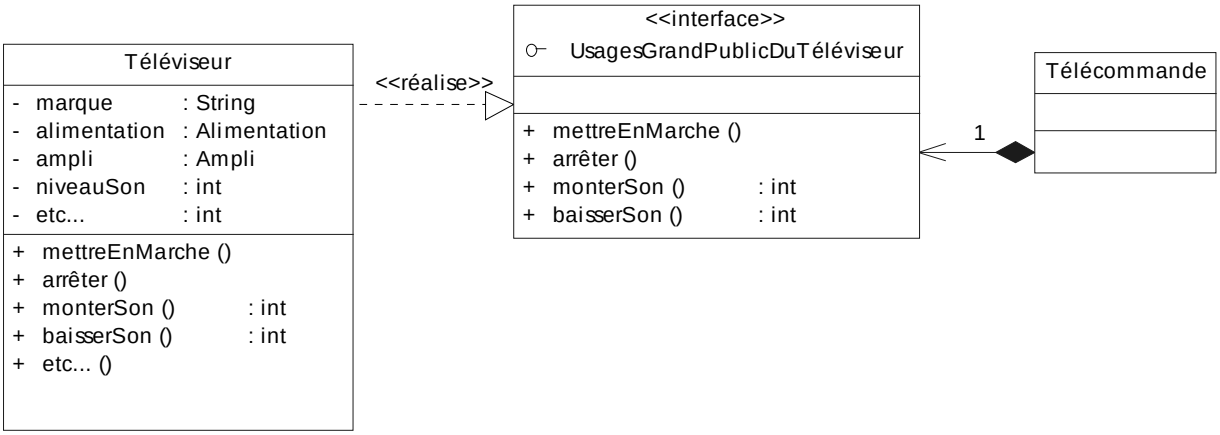


La classe « UsagesGrandPublicDuTéléviseur » est stéréotypée « interface ». Le Téléviseur « réalise » l'interface UsagesGrandPublicDuTéléviseur .

Le lien de réalisation est en pointillé. La flèche est un triangle creux (comme pour l'héritage). Le lien peut être stéréotypé « réalise ».

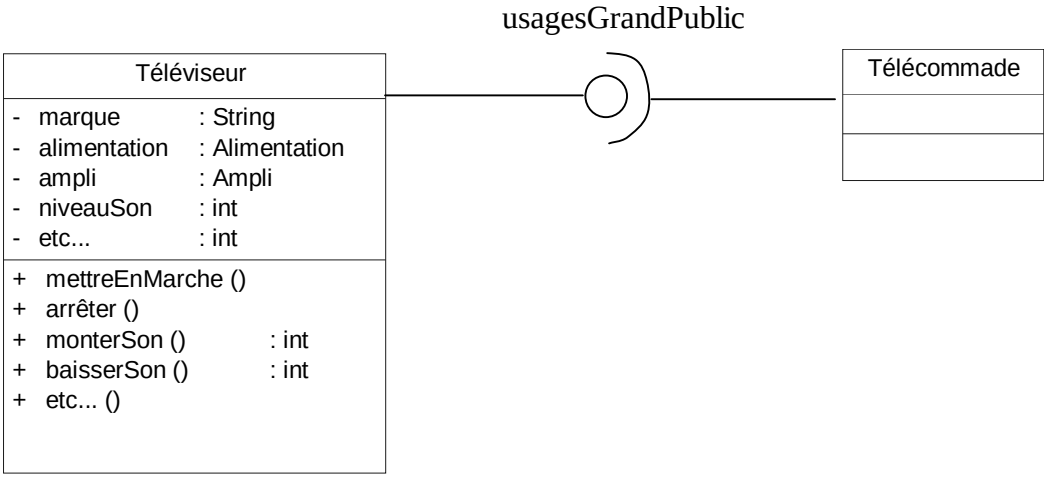
La classe « télécommande » utilise l'interface : la télécommande à besoin de l'interface pour fonctionner.

autre représentation



Le « use » est en réalité une composition.

autre représentation



Le cercle correspond à la classe interface

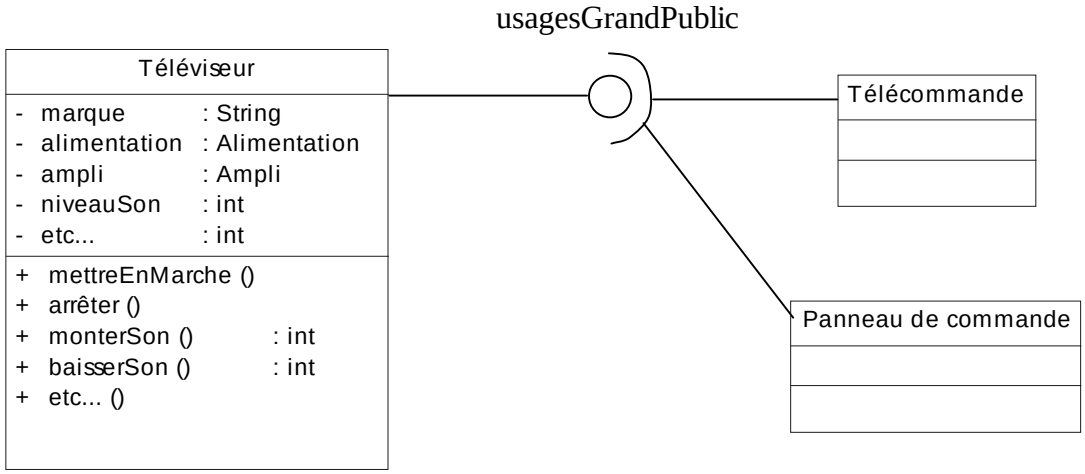
Le lien entre le téléviseur et le cercle correspond à la réalisation de l'interface.

Le lien avec le demi-cercle correspond à un lien d'utilisation entre classe qui spécifie l'utilisation d'une interface.

Un deuxième usage de l'interface

La télévision peut aussi être commandé via son panneau de commandes intégré au poste.

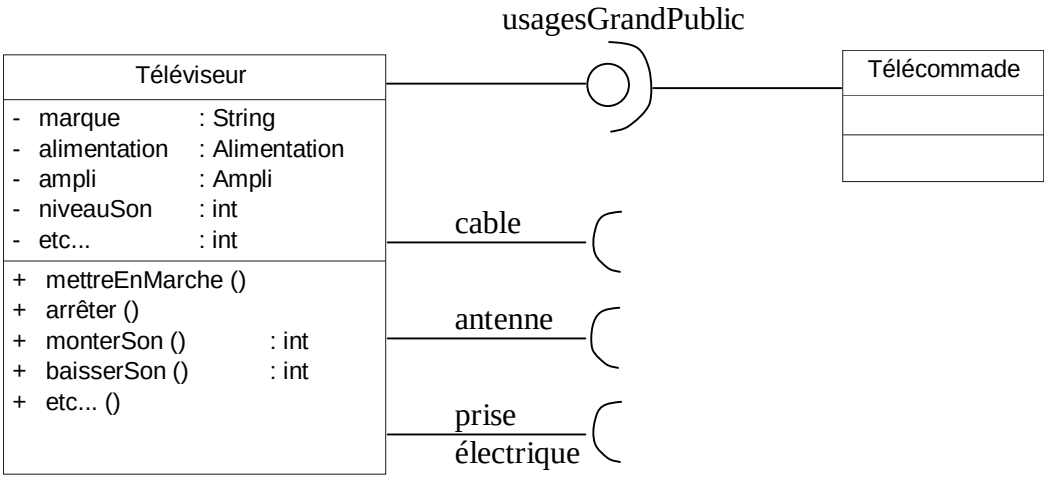
Le modèle devient alors :



L'intérêt est donc de bien dissocier, dans la conception, la partie interface utilisateur de la partie objets du système.

interface utilisée

Si on ajoute les interfaces utilisées par le téléviseur, à savoir un cable, une antenne et une prise électrique, on obtient le modèle suivant :



Conception par les interfaces

Pour trouver les interfaces du système, il faut **examiner toutes les paires : utilisateur – usage du système.** On a intérêt à créer dans un premier temps au moins une classe interface par usage.

Les interfaces se trouvent à un haut niveau d'abstraction. On commence par renseigner les besoins en interfaces utilisateur sans les implémenter. Ces classes seront affinées au fur et à mesure.

5. Composant logiciel

Présentation

Modularité, **abstraction** et **encapsulation** permettent de construire des composants logiciels réutilisables pouvant aller d'un simple bouton à une application toute entière comme un traitement de texte.

Deux types de modularité

La modularité dynamique : l'objet

- **Un objet est un module** cohérent regroupant des données et des opérations.
- Ces modules sont créés dynamiquement et peuvent être passés en paramètre.
- Par l'intermédiaire du nom, on a accès aux opérations.

Exemple : un objet polynôme aura deux opérations : calculerRacines( ) et afficherRacines( )

La modularité statique : le package

L'architecture logicielle va consister à regrouper les classes dans des modules (les packages). C'est la modularité statique.

Abstraction

L'abstraction permet d'utiliser un objet général sans se préoccuper des différences et des spécialisations entre différents types de cet objet. Seul le concept général est utilisé ce qui revient à abstraire les différences.

Exemple : 3 objets de type voiture : **break**, **berline** et **coupé**, auront chacun des méthodes démarrer( ), couperContact( ), etc. Un utilisateur pourra utiliser un objet **voiture** sans se préoccuper des différences de nature entre les voitures.

Encapsulation

L'objet cache la plupart de ses caractéristiques à ses utilisateurs. Les données de l'objet sont cachées. **Seules les spécifications (en-tête) des opérations dites publiques sont visibles des utilisateurs.** On retrouve là la **distinction entre fonctionnel et organique** (le quoi et le comment, l'externe et l'interne, le visible et le caché).

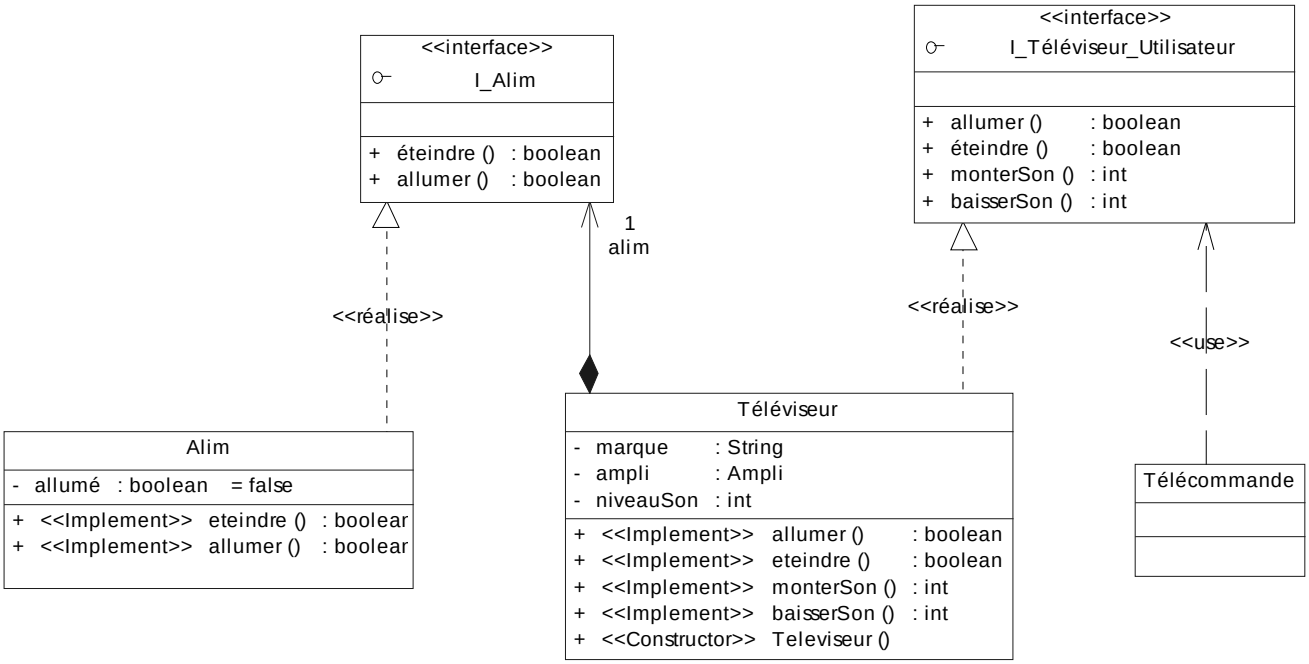
Les spécifications sont simplifiées car tous les paramètres correspondant à des données de l'objet étant cachés, il n'est pas nécessaire de les préciser.

Les données modifiées par les opérations sont gardées dans la mémoire de l'objet.

Exemple : un objet adhérent avec 4 opérations publiques : envoyer(unCourrier : Courrier), afficher(), mettreAJourDossier(), recevoir(unCourrier : Courrier). L'opération afficher permet de consulter les donnée de l'adhérent. L'opération mettreAJourDossier permet de modifier les données de l'adhérent.

Exemple

On traite le bloc d'alimentation comme un composant à part :



La classe « Alimentation » réalise deux méthodes d'interface. Le téléviseur accède à la classe Alimentation par son interface qu'il utilise en tant que composant logiciel.

Concrètement, le constructeur de la classe « Téléviseur » instancie un objet « Alim » qu'on associe à l'attribut « alim ».

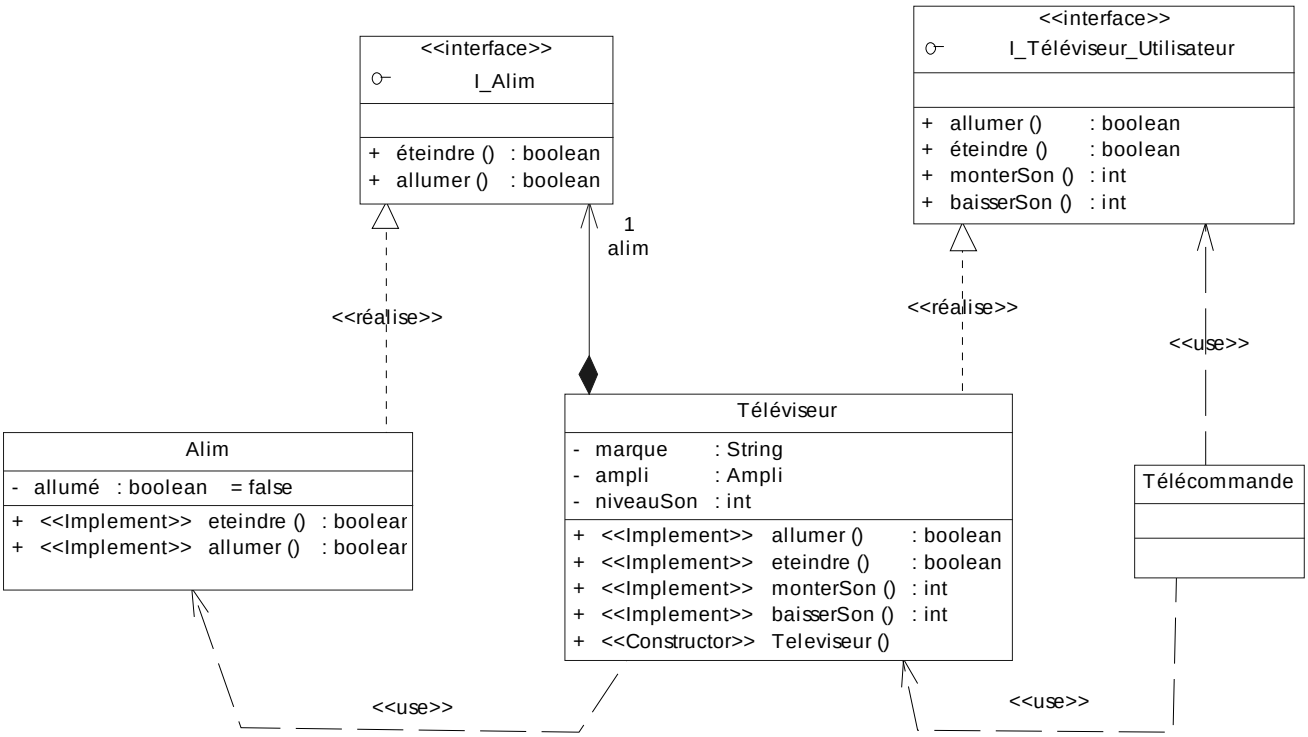
Code Java du constructeur :

```
public Televiseur() {
   alim= (IAlimentation) new Alim();
   alim.allumer();
}
```

De ce fait, il existe une dépendance de type « use » entre le téléviseur et l'ampli.

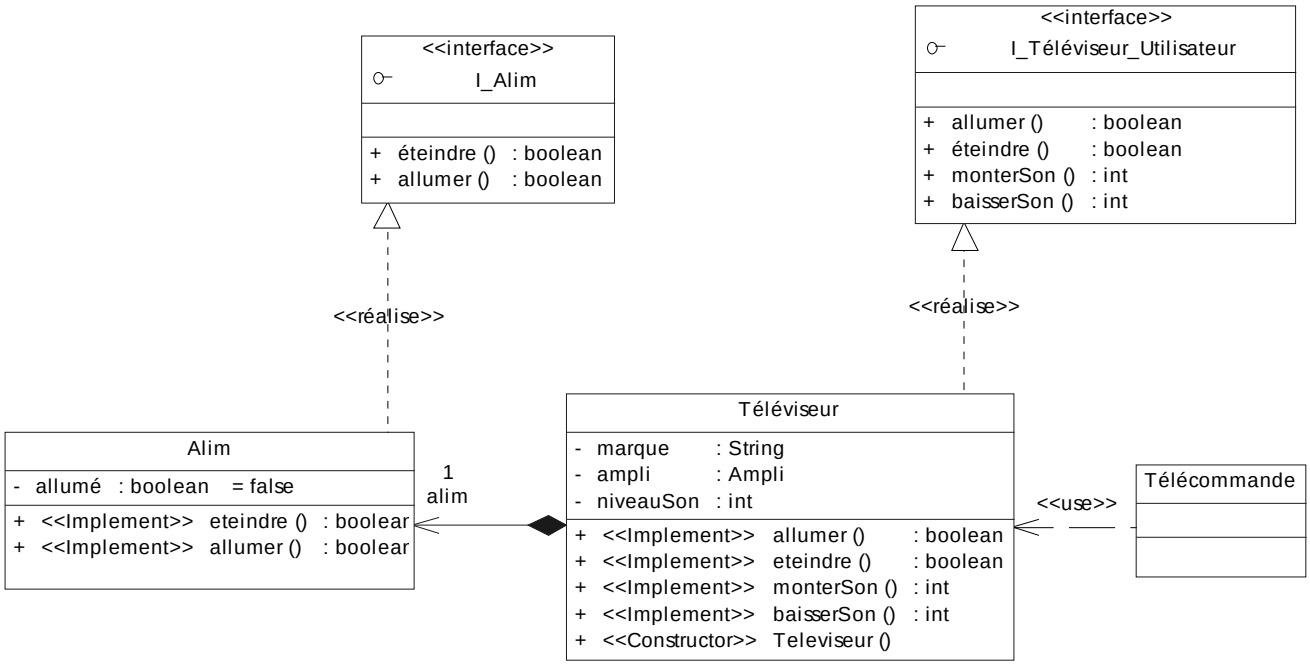
C'est le même principe entre la télécommande et l'ampli.

On pourrait donc avoir le diagramme de classes suivant :



Toutefois, on évite, en général, de mettre les relations de dépendance autres que celles des interfaces.

A noter aussi que les schémas précédents sont équivalents à :



Toutefois, on préfèrera la première version pour des raisons de lisibilité.

6. L'objet

Définition

Un objet est un exemplaire d'une classe.

Principes techniques

Déclaration

Pour pouvoir utiliser un objet, c'est-à-dire utiliser une de ses opérations publiques, il faut le déclarer une variable ayant comme type la classe de l'objet en question. Techniquement, la déclaration d'un objet est toujours la **déclaration d'un pointeur** sur un objet. La déclaration ne crée pas l'objet mais seulement le pointeur vers un futur objet.

Construction

On distingue donc entre déclaration et construction de l'objet. La construction s'effectue par l'opérateur « **new** » qui va allouer dynamiquement un objet et renvoyer l'adresse de cette allocation, adresse qui sera affectée à une variable ayant comme type la classe de l'objet en question.

Comme toute variable allouée dynamiquement, l'objet n'a donc pas à proprement parler de nom. Quand on déclare un objet, le nom de la variable est le nom du pointeur qui permet d'accéder à l'objet. Le pointeur pourra référencer un autre objet : on peut donc changer d'objet sans changer de variable.

Initialisation

Un objet peut être initialisée lors de sa construction, via des opérations particulières appelées « **constructeurs** ».

En conception, la construction-initialisation n'est pas abordée. C'est cependant un élément centrale de la programmation.

Lieux de déclaration – construction des objets

La déclaration et la construction d'un objet peut se faire à deux endroits différents :

- Au niveau des attributs d'une classe
- Au niveau d'une variable locale d'une opération

Provenance des objets d'une méthode

Un objet, dans le corps d'une méthode, provient de trois lieux :

- C'est un attribut de l'objet de la méthode : il a été construit avec l'objet.
- C'est une variable locale à la méthode : il est construit dans la méthode.

- C'est un paramètre formel de la méthode : il a été fournit par la fonction appelante. Toutefois, en dernière analyse, on retombera sur les deux premiers cas.

Ce dernier cas fait que : si un objet d'une classe 1 fait appel à une opération dont un paramètre est un objet d'une classe 2 alors la classe 1 dépend de la classe 2.

Syntaxe UML

Les objets sont soulignés et placés dans un rectangle. Le nom de l'objet commence par une minuscule. Le nom de la classe commence par un majuscule.

Objets nommés : olivier bertrand

Objets sans noms : : Eleve : Professeur

7. La communication entre les objets : envoi de message

Principe général de l'envoi de message

Envoyer un message à un objet c'est utiliser une de ses opérations.

Un objet 1 envoi un message à un objet 2 quand une opération de l'objet 1 utilise une opération de l'objet 2.

Pour cela, l'objet 2 doit exister dans l'environnement de l'opération appelante de l'objet 1 :

soit c'est une variable locale de l'opération 1,

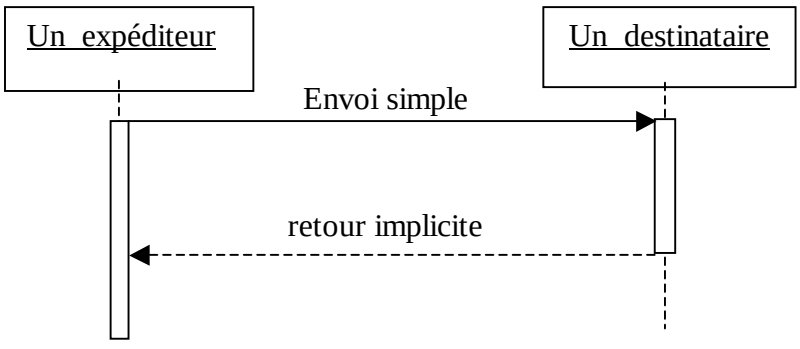
soit c'est un objet passé en paramètre de l'opération 1,

soit c'est un attribut de l'objet 1.

Envoi de message et héritage

En programmation objet, **la manière d'associer du code à un message** est un mécanisme dynamique qui n'est pas établi à la compilation. Autrement dit, l'envoi d'un message **ne correspond pas à un débranchement vers une adresse de code prédéfinie** comme c'est le cas en programmation procédurale classique. **C'est la relation d'héritage entre classes qui permet de gérer ce mécanisme**.

Syntaxe UML du diagramme de séquence



8. Exemple complet : la bibliothèque

Cahier des charges

On considère l'exemple d'une bibliothèque. On considère 2 usages : l'emprunt de livres et le retour d'un livre.

Pour emprunter un ou plusieurs livres, l'adhérent se sert dans les rayonnages, puis donne sa carte au bibliothécaire qui l'identifie. Ensuite, l'adhérent donne ses livres au bibliothécaire qui les identifie. Ensuite, le bibliothécaire redonne la carte et les livres à l'adhérent. L'adhérent ne peut pas emprunter plus de 3 livres en même temps. Les livres ne doivent pas être emprunter plus de 14 jours. En cas de retard d'un livre, l'adhérent ne peut plus emprunter d'autres livres.

Pour rendre un livre, n'importe qui donne le livre au bibliothécaire qui l'identifie.

Analyse des interfaces

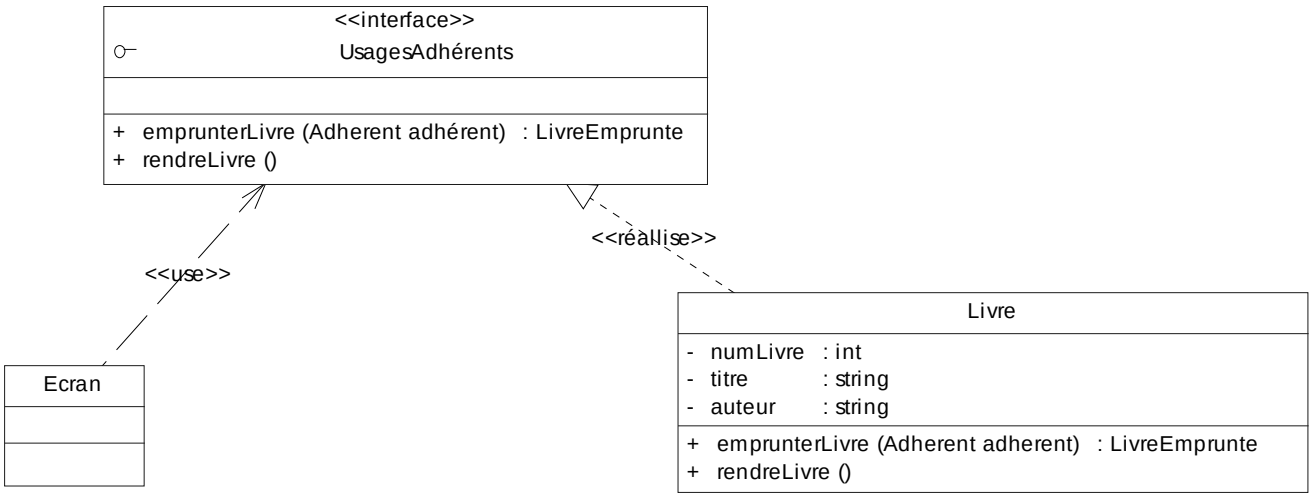
Pour trouver les interfaces du système, il faut **examiner toutes les paires : utilisateur – usage du système.**

L'utilisateur, c'est le bibliothécaire. Il y a deux usages :

rendreLivre( ). Pour cela, il suffit d'identifier le livre à rendre.

emprunterLivre( ). Pour cela, il faut commencer par identifier l'adhérent pour ensuite identifier le livre à emprunter.

Ces usages sont des **méthodes de la classe Livre**. On rend quoi : un livre. On emprunte quoi : un livre. Ces usages caractérisent l'objet livre.



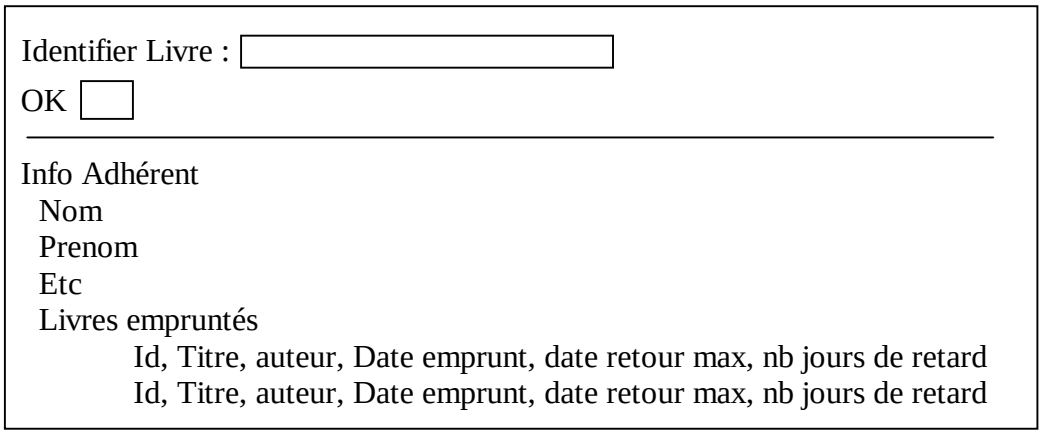
Remarque :

Les paramètres des méthodes seront découverts avec l'analyse du diagramme de séquence.

Analyse des écrans

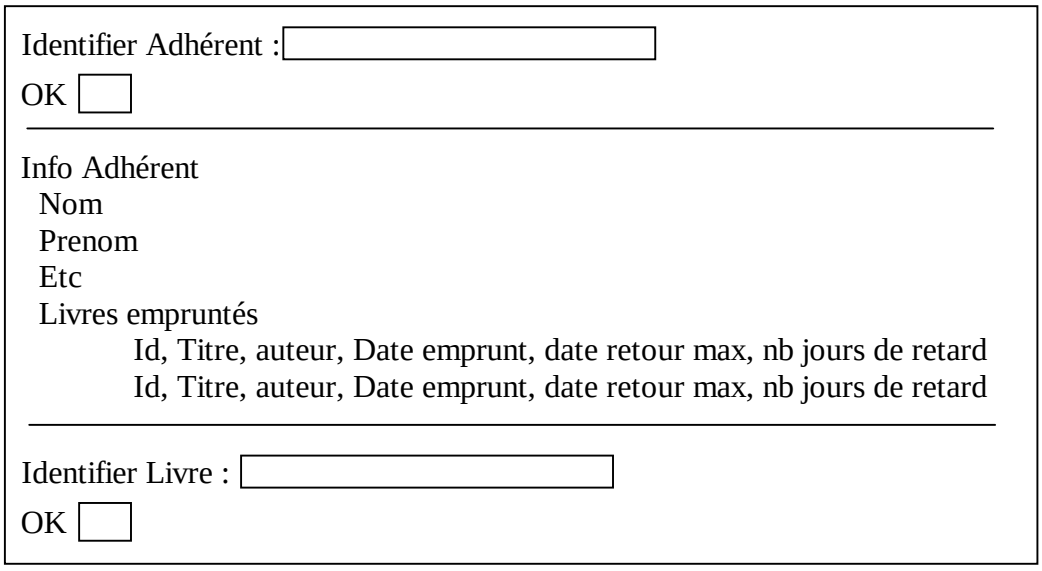
On a intérêt à imaginer les écrans pour concrétiser le système :

Ecran « enregistrerUnRetour( )



On saisie l'id du livre et on valide. On récupère les infos sur l'adhérent et les livres en cours d'emprunt.

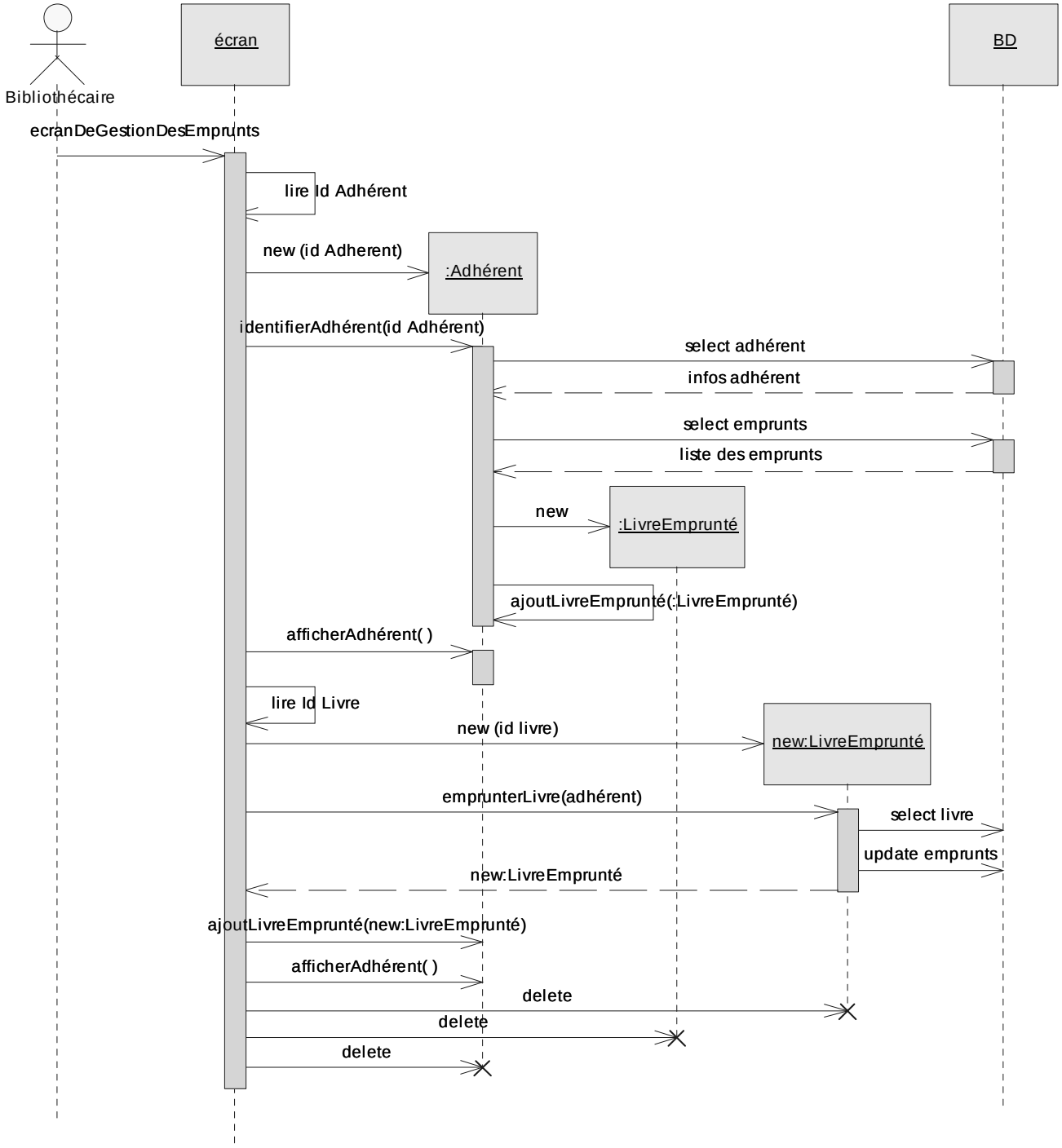
Ecran « enregistrerDesEmprunts( )



On saisie l'id de l'adhérent et on valide. On récupère les infos de l'adhérent. On saisie l'id du livre et on valide. La liste des emprunts de l'adhérent est mise à jour.

Analyse des scénarios

Scénario d'emprunt d'un livre (diagramme de séquence) :



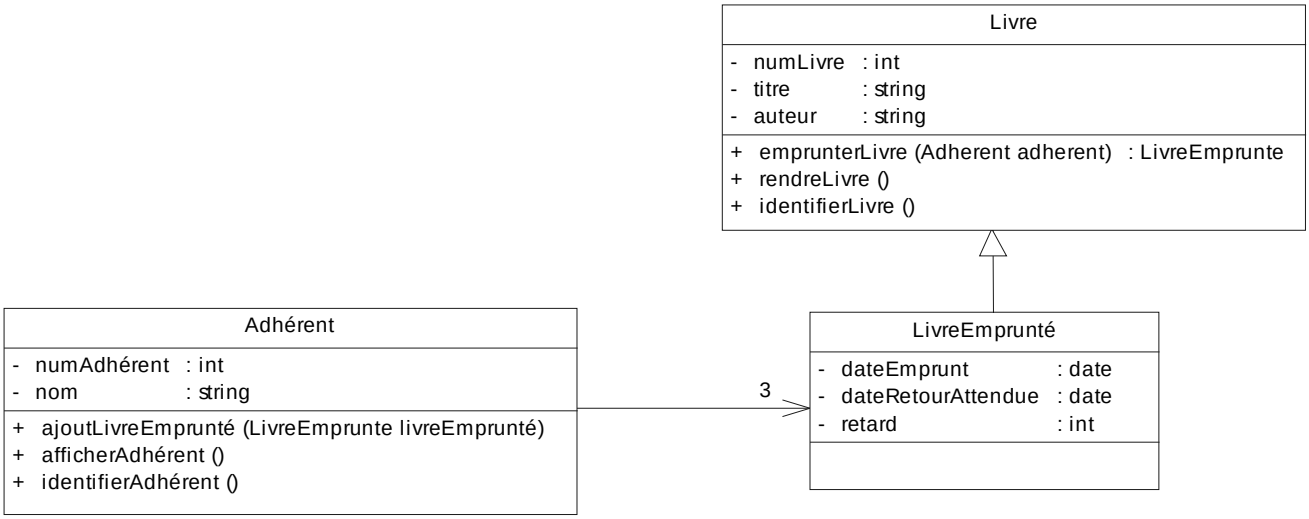
Remarques :

- On crée un pseudo objet écran qui nous permet de clarifier l'interface avec l'utilisateur, et un pseudo objet « BD » pour clarifier l'interface avec la BD. Ces objet n'apparaîtront pas dans le diagramme de classes.

- Les objets « :Adhérent », « :LivreEmprunté » et « new:LivreEmprunté » sont détruits à la sortie de la méthode « EnregistrerDesEmprunts »

- La méthode « identifierAdhérent » pourrait être intégré dans le constructeur de l'adhérent et ne pas être détaillée au niveau du diagramme de séquence.

Diagramme des classes



Remarques :

On met 3 pour le nombre de livres empruntés par l'adhérent pour obtenir un tableau 3 livres empruntés dans la table des adhérents.

Le livre emprunté est une espèce de livre : il hérite des attributs et des méthodes du lives et porte des attributs supplémentaires. A noter qu'on devrait descendre la méthode : « emprunterLivre » au niveau de la classe « LivreEmprunté »

9. Diagramme de classes et base de données : les classes « métier ».

Classes-métier et classes organiques

On distingue entre « classe-métier » et classe organique.

Les classes-métier sont celles qui correspondent directement au métier de l'application traitée tandis que les classes organiques sont des classes plus techniques.

Les classes-métier correspondent reprennent *grosso modo* le modèle conceptuel des données (le MCD).

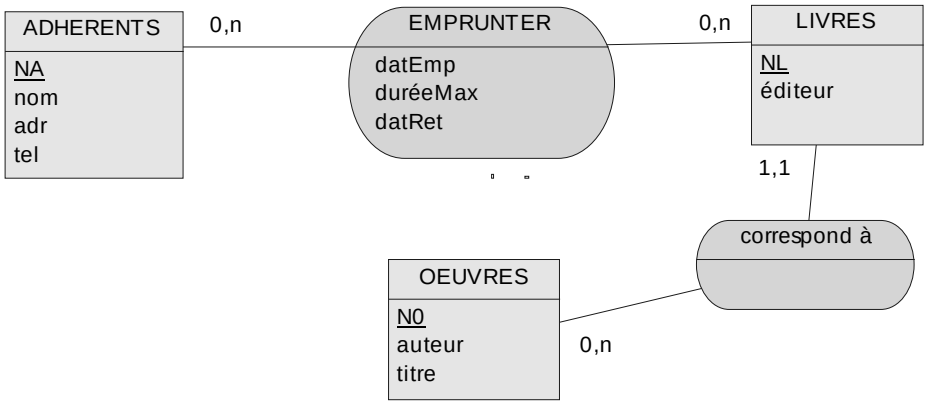
Elles permettent une première approche du système.

Classes-métier et base de données

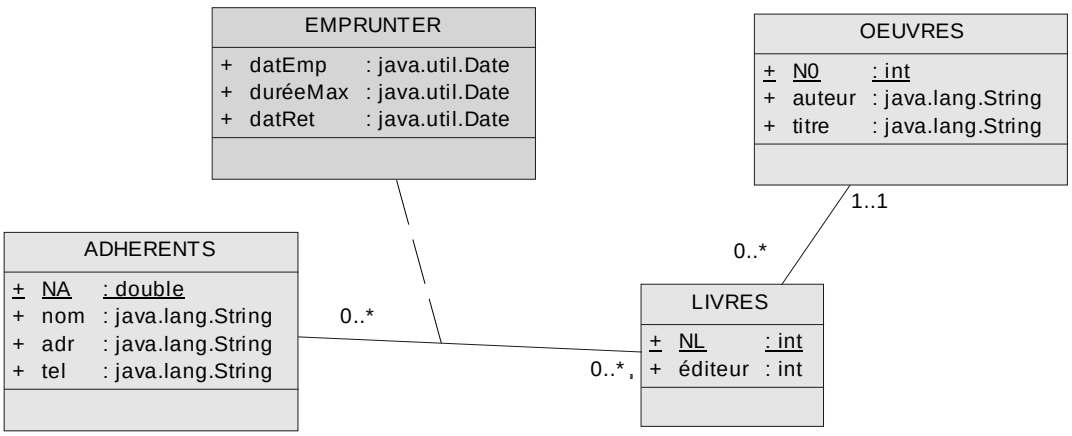
On peut utiliser le MCD ou les tables de la base de données pour commencer à réaliser un diagramme de classes-métier.

Relations entre le modèle BD et le modèle objet

MEA



UML



Explications de technique UML

- Tous les attributs sont « + » c'est-à-dire public. En réalité, ils devraient passer « - » , c'est-à-dire privés. Il faut donc faire attention aux résultats des traductions automatiques !

- En UML, on ne peut pas mettre d'attributs sur les associations. Les associations non hiérarchiques avec attributs du MEA donnent lieu dans le modèle de BD UML à des classes-associations.

- Une classe-association est une association classique à laquelle est raccrochée une classe dont les attributs proviennent de l'association non hiérarchique du MEA.

Explications sur la modélisation

- Les classes « livres » et « oeuvres » ont été fusionnées. Au niveau de l'application, leur distinction est inutile.

- La classe association « emprunter » devient une simple classe : « LivreEmprunté » : ça permet de gérer plus finement les cardinalités et les navigabilités.

- Les navigabilités sont limités à « Adhérent » vers « LivreEmprunté » et « LivreEmprunté » vers « Livre ». Cela suffit gérer les usages.

- L'association entre « LivreEmprunté » et « Livre » est une composition : on aura un attribut de type « Livre » dans la classe « LivreEmprunté ».

- L'association entre « Adhérent » et « LivreEmprunté » est de cardinalité 3 : on aura attribut livreEmrunte qui sera un tableau de 3 « LivreEmprunté ». On n'a besoin que de ça.

- On a ajouter la classe « Bibliothécaire » qui porte les méthodes de l'interface.

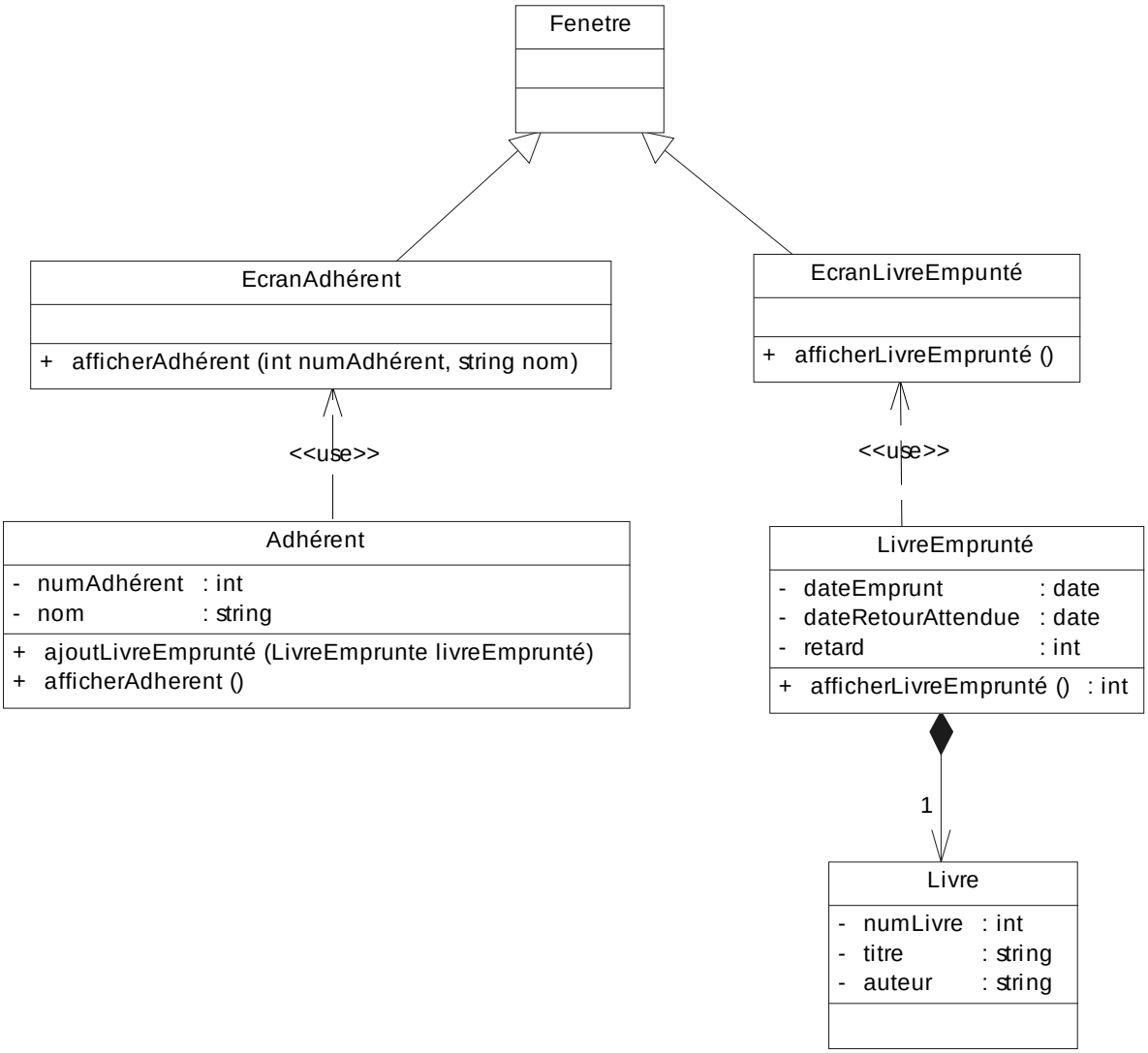
- La classe « Bibliothécaire » utilise la classe « Adhérents »

Remarque générale

On peut partir d'un MCD pour commencer le diagramme des classes. Toutefois, le diagrammes des classes doit être retravaillé en fonction des usages du système.

10. L'affichage et les écrans

On peut aussi avoir un modèle qui gère l'affichage au niveau de chaque classe :



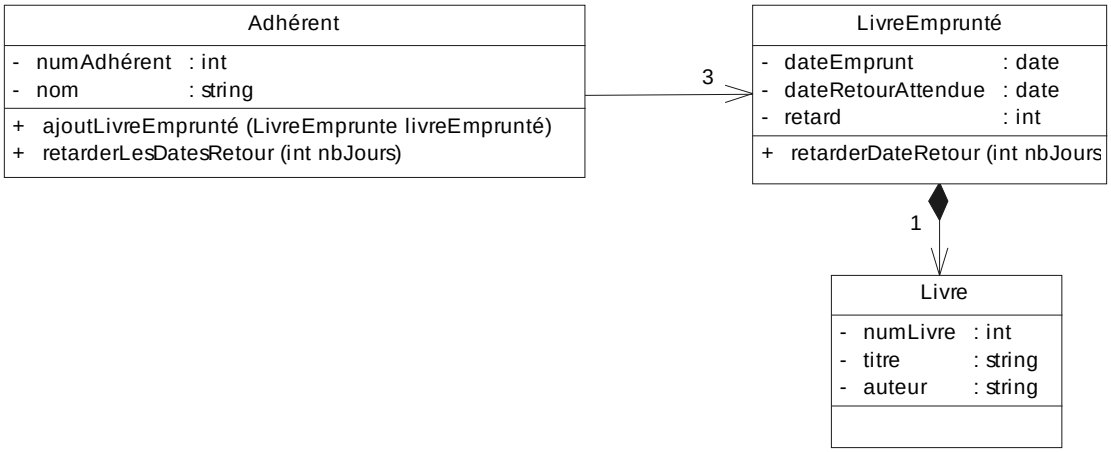
Avec un tel modèle, la notion d'interface est toujours utile pour présenter les services rendus par le système, mais ne sert plus à dissocier le calcul de l'interface utilisateur.

A noter que dans un diagramme de classes global, on évitera la présentation des fenêtres et des écrans.

11. La gestion des collections

Quand on a des collections (ici le tableau des 3 livres empruntés), il peut y avoir des traitements qui s'appliquent à toute la collection. Dans notre exemple, les adhérents pourrait appliquer des traitement à tous leurs livres empruntés. Par exemple, on pourrait vouloir ajouter une semaine de délai à tous les livres empruntés d'un adhérent

Le modèle devient le suivant :

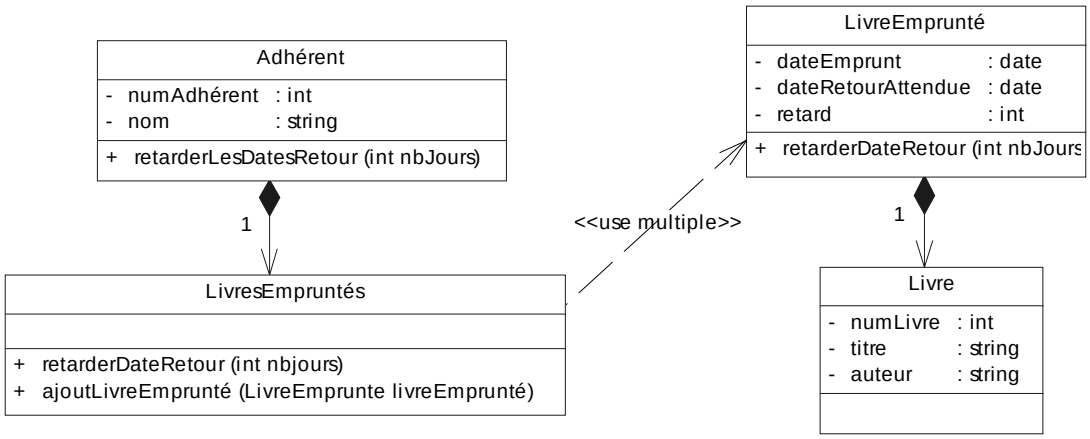


L'algorithme de « retarderLesDatesRetour » est

Pour i de 1 à 3
LivreEmprunté[i].retarderDateRetour(nbJours)
Fin pour

Pour généraliser le traitement, c'est-à-dire éviter que la boucle de traitement soit dans la classe appelante « Adhérent », on définit une classe « plurielle » ou « collection » : « LivesEmpuntés ». C'est cette classe qui fera le traitement alors que la classe adhérent ne fera qu'appeler la méthode de la classe plurielle. Ainsi le traitement sera accessible à n'importe quelle autre classe.

Le modèle devient le suivant :



Les itérations sur l'ensemble des livres empruntés ont été encapsulées dans la classe plurielle « LivresEmpruntés ». La méthode « retarderLesDatesRetour » de « LivresEmpruntés » contient la boucle.

Algorithme de « retarderLesDatesRetour » : LivresEmpruntés.retarderDateRetour(nbJours)

12. Les 3 relations fondamentales entre les objets (et les classes) : composition, utilisation, héritage

Il y a trois types de relations entre les objets : la composition, l'utilisation et l'héritage.

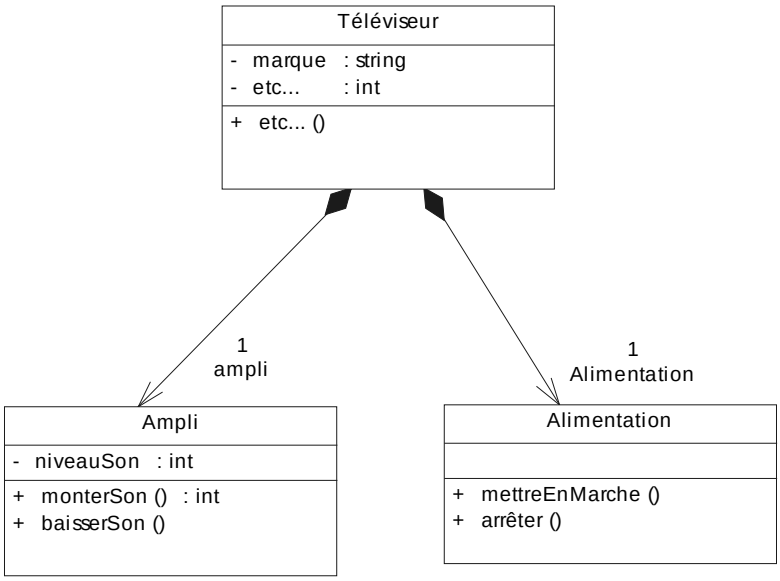
Ces trois relations sont aussi les trois relations fondamentales entre les classe

Ces relations génèrent des dépendances entre les objets.

La composition

Un objet 2 est composant d'un objet 1 si c'est un **attribut** de l'objet 1.

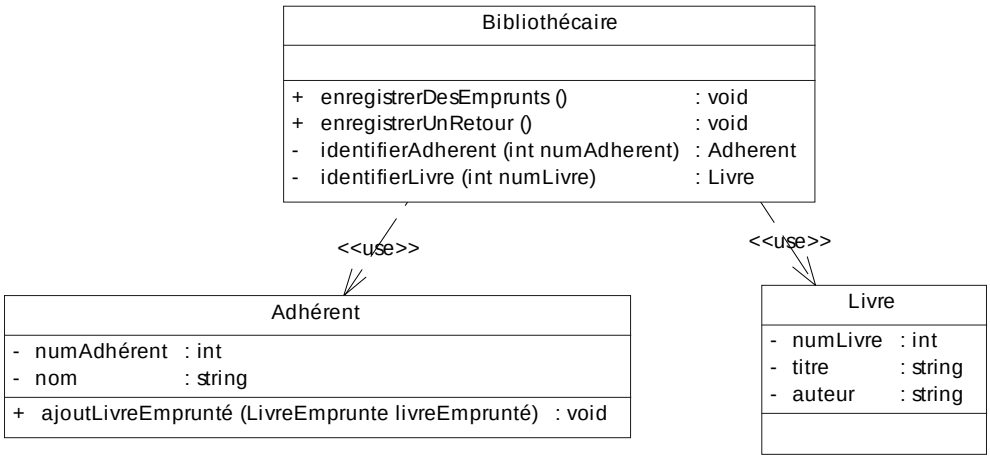
La relation de composition génère une **dépendance** : l'objet 1 dépend de l'objet 2.



L'utilisation (ou dépenndance)

Un objet 2 est utilisé par l'objet 1 si il est **déclaré localement** dans une opération d'un objet 1 ou **passé en paramètre formel** d'une opération d'un objet 1.

La relation d'utilisation génère une **dépendance** : l'objet 1 dépend de l'objet 2.



L'héritage

Une classe peut hériter des attributs et des opérations d'une autre classe. La classe héritière est appelée **sous-classe**, la classe dont elle hérite est appelée **classe parente**. Une classe parente peut à son tour hériter. Si une classe a une seule classe parente, on parle d'**héritage simple** sinon d'**héritage multiple**.

Héritage d'attributs : l'ensemble des **attributs d'un objet** d'une classe est constitué de l'ensemble de ses attributs et de l'ensemble des attributs de ses classes parentes.

En ce sens, l'héritage est une **façon d'organiser les concepts** dans un rapport d'espèce à genre : l'espèce (sous-classe) est une sorte du genre (classe parente).

Héritage d'opérations : l'ensemble des **opérations applicables à un objet** est constitué de l'ensemble de ses opérations et de l'ensemble des opérations de ses classes parentes.

En ce sens l'héritage est une **technique de partage et de réutilisation du code existant**.

La manière d'associer du code à un message est un mécanisme dynamique qui n'est pas établi à la compilation. L'envoi d'un message ne correspond pas à un débranchement vers une adresse de code prédéfinie comme c'est le cas en programmation procédurale classique. La programmation objet permet de trouver dans la hiérarchie des classes, le code du message à exécuter.

L'algorithme qui permet d'associer du code à un message peut donc être vu ainsi :

```
ExecRécursif (message, classe)
    Si ilExiste (message, classe)
        Exectuer (message, classe)
        Return
    Finsi
    Si parent (classe) = null
        Afficher (« erreur »)
        Return
    Finsi
    ExecRécursif (message, parent (classe) )
Fin
```

Formalisme UML