Atelier Génie Logiciel
Prise en main de StarUML sur l'étude de cas `Bookinons` *(2ème partie)*

Exercice 3 : Axe statique : la vue Logique et le diagramme de classes

3.1 Organisation des classes en paquetages (package) `presentation, application, metier`

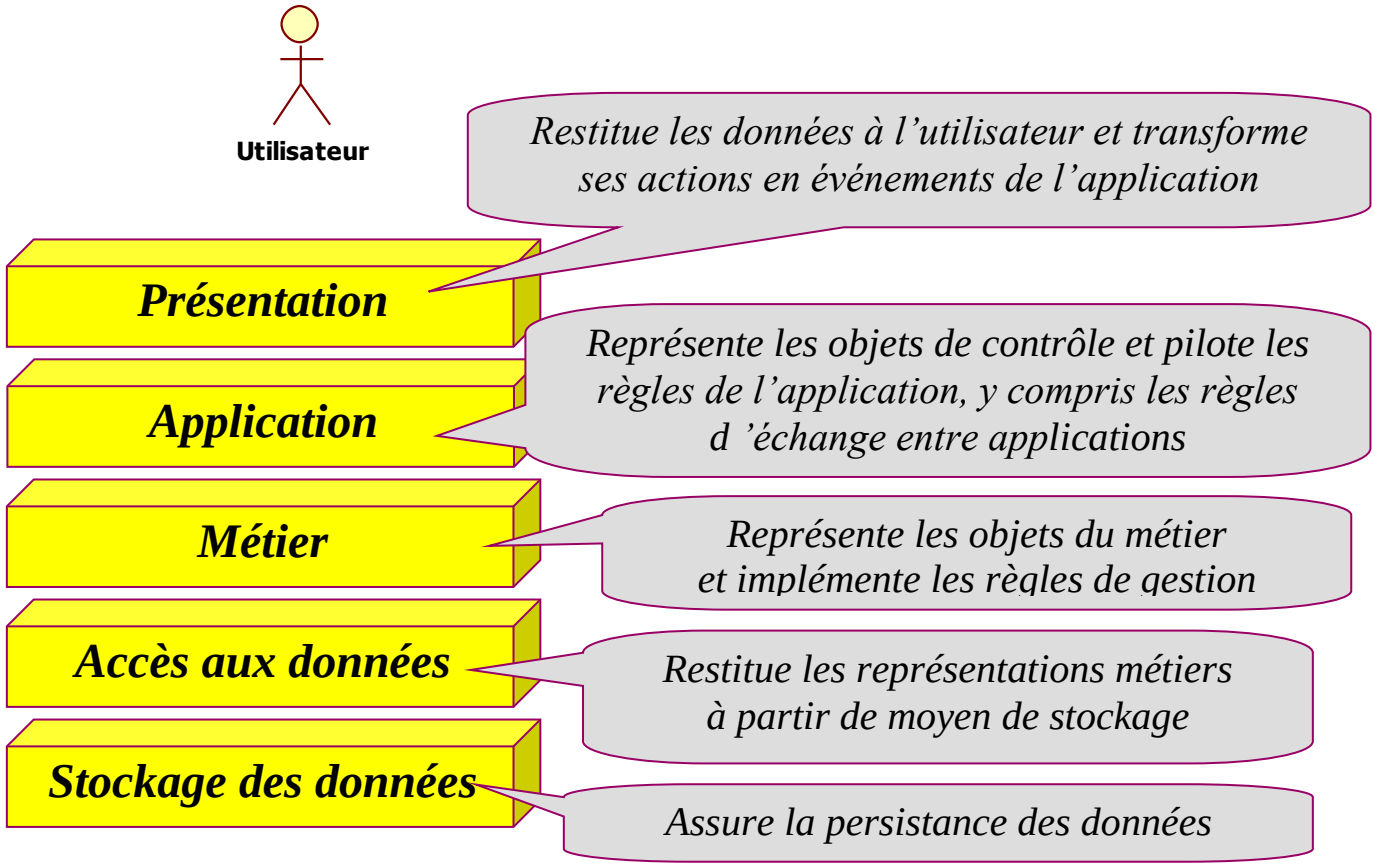
Nous avons décidé d'organiser notre ***vue logique (`Logical View`) en package*** et de créer un package par Use Case ***afin de pouvoir regrouper dans chaque package tous les diagrammes communs au même UC*** (diagramme d'activités, diagrammme de séquence et diagramme de classes participantes) et de permettre ainsi une meilleure lisibilité de la conception.

En revanche, certaines classes de l'application vont être utilisées par plusieurs Use Case.
Il n'est donc pas envisageable de mettre toutes les classes de l'application dans le package d'un UC particulier.
En effet, les classes vont être partagées par tous les UC : elles doivent donc être stockées dans un lieu accessible par tous les UC c-à-d soit directement à la racine de la **`Logical View`**, soit dans un **paquetage commum.**
Nous choisissons la solution du paquetage commun, et nous vous demandons de créer à la racine de la **`Logical View`** un nouveau paquetage que vous appelerez **`bookinons`**

En vue de la préparation de la phase d'implémentation, et afin de travailler de manière organisée, les classes ne seront pas toutes déposées telles quelles dans ce nouveau paquetage, mais elles seront réparties dans trois paquetages que nous appelerons : **`presentation, metier, application`** afin de respecter le découpage en couches logicielles préconisé dans les applications informatiques d'aujourd'hui.

3.1.a Quelques mots sur la notion de couche logicielle :

En effet, les systèmes informatiques sont de nos jours organisés en couches.
Le recours aux couches logicielles permet de diviser le problème en sous-parties indépendantes.
Par exemple, pour le développement d'application client/serveur, Rumbaugh préconise une architecture en 5 couches.



L'intérêt d'une architecture en couche est de pouvoir répondre à un ***critère d'évolutivité*** : *comme par exemple pouvoir modifier la* ***présentation*** *sans devoir modifier les règles* ***métier****, et/ou pouvoir changer de mécanisme de* ***stockage****(persistance) sans avoir à retoucher ni à la* ***présentation****, ni aux règles* ***métier…***

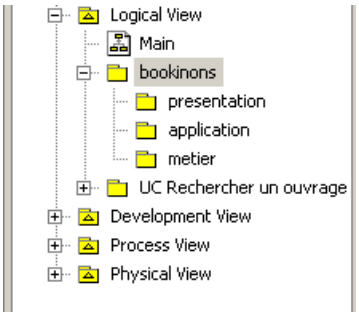
Ces notions de couches seront reprises plus tard dans le cours UML sur le diagramme de séquence.

3.1.b Mise en place des paquetages `presentation, application, metier`

Créer dans votre paquetage **`bookinons`** de la **`Logical View`**, les 3 sous-paquetages:

`presentation, application, metier`

Pour plus de faciliter par la suite, faîtes remonter à l'aide des flèches bleues le paquetage **`bookinons`** comme premier paquetage dans votre **`Logical View`**.



3.2 Mise en place des classes métiers

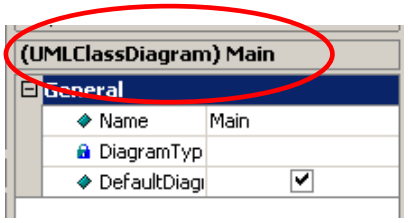
A l'issue de la description détaillée du Use Case, des *classes "pressenties"* ont pu être identifiées grâce au tableau des champs établi à partir des données représentées sur les maquettes.
Ces classes représentent ***les concepts métier*** de l'application : elles seront donc mémorisées dans le paquetage **`metier`**. Elles sont souvent persistantes et sont également appelées ***entités*** (ou **Entity**).
Pour l'instant, nous ne travaillerons donc qu'avec des **objets métiers**.
Remarque : Il est à noter que la ***couche métier*** pourrait tout-à fait être re-découpée en packages regroupant chacun des classes métier de forte cohérence et faible couplage...

3.2.a Création d'une classe **Livre.**

La première classe que nous allons créer est la classe **Livre**.
Placez-vous dans le **Model Explorer** sur le paquetage **metier**, puis d'un clic droit de souris, faites **Add → Class** et donner à cette classe le nom de **Livre**.

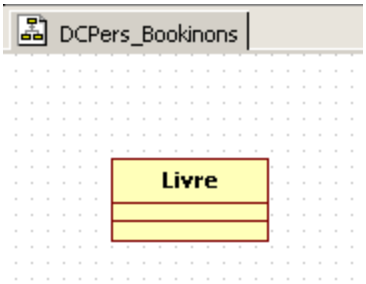
3.2.b Utilisation de la classe **Livre** dans un diagramme de classes

Pour visualiser la classe dans son formalise UML, nous devons la positionner dans un diagramme de classes.
Placez-vous sur le fichier **Main** disponible à la racine de la **Logical View** et consultez la fenêtre **Properties**.



Cette fenêtre vous indique que ce diagramme est un diagramme de classes.
Par défaut, à la création du projet un diagramme de classes a été créé à la racine de la **Logical View**. Renommez ce diagramme de la manière suivante : **DCPers_Bookinons :** ce diagramme nous servira dorénavant de ***Diagramme de Classes Persistantes*** pour l'application **Bookinons**.

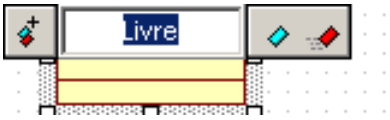
Cliquez dans le **Model Explorer** sur le diagramme **DCPers_Bookinons** afin qu'il devienne le diagramme actif de votre fenêtre de travail.
Cliquez alors sur la classe **Livre** dans le **Model Explorer** et faîtes la glisser jusqu'au diagramme **DCPers_Bookinons** , relâcher le bouton de la souris, la classe **Livre** apparaît sur le diagramme.



3.2.c Attributs et méthodes de la classe **Livre**

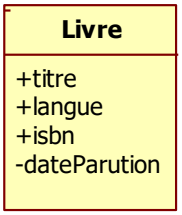
Il existe plusieurs manières pour renseigner les attributs et les méthodes de la classe **Livre** soit directement à partir du graphique (le plus simple) , soit à partir de la fenêtre **Properties**.

↳ Nous allons commencer par compléter la classe **Livre** directement **depuis le diagramme.**



Double-cliquez sur la classe **Livre**.
La classe **Livre** est alors modifiable.
Vous pouvez changer son nom.
Vous pouvez également ajouter des attributs et des méthodes via respectivement le rectangle bleu et le rectangle rouge à droite du bandeau gris proposé.

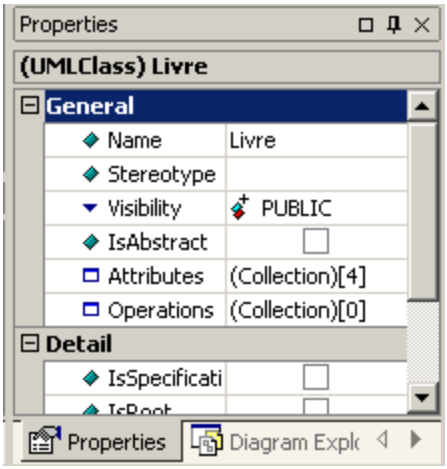
Pour rajouter un ***attribut***, il vous suffit de cliquer sur ***le rectangle bleu*** à droite du bandeau gris.
Vous pouvez alors saisir le nom de l'attribut. Commençons par entrer **titre.**
Pour saisir un nouvel attribut appuyez sur **+.**
Remarque : Lorsque vous entrez un nouvel attribut, vous pouvez choisir son niveau de visbilité en cliquant à gauche du bandeau gris sur le bouton représentant un + et 2 rectangles rouge et bleu.
Pour l'instant, nous commençons juste à rechercher les classes, le niveau de visibilité nous importe peu pour le moment (même s'il est par défaut à public).
Complétez la classe **Livre** avec les attributs suivants : **langue**, **isbn**,
Puis ajoutez un nouvel attribut et tapez cette fois-ci : **- dateParution**, puis cliquez n'importe où ailleurs sur le diagramme pour sortir de la modification de la classe Livre. Votre classe **Livre** apparaît conforme à la représentation ci-contre.
Vous pouvez alors constater qu'il est possible de saisir la visibilité directement en



même temps que le nom de la classe en faisant précéder le nom de la classe de **-** ou **#** (rappelons que si aucun niveau de visibilité n'est précisé, le niveau de visibilité est public (**+**) par défaut : ce qui fut le cas pour les premiers attributs saisis : **titre**, **langue** et **isbn**.

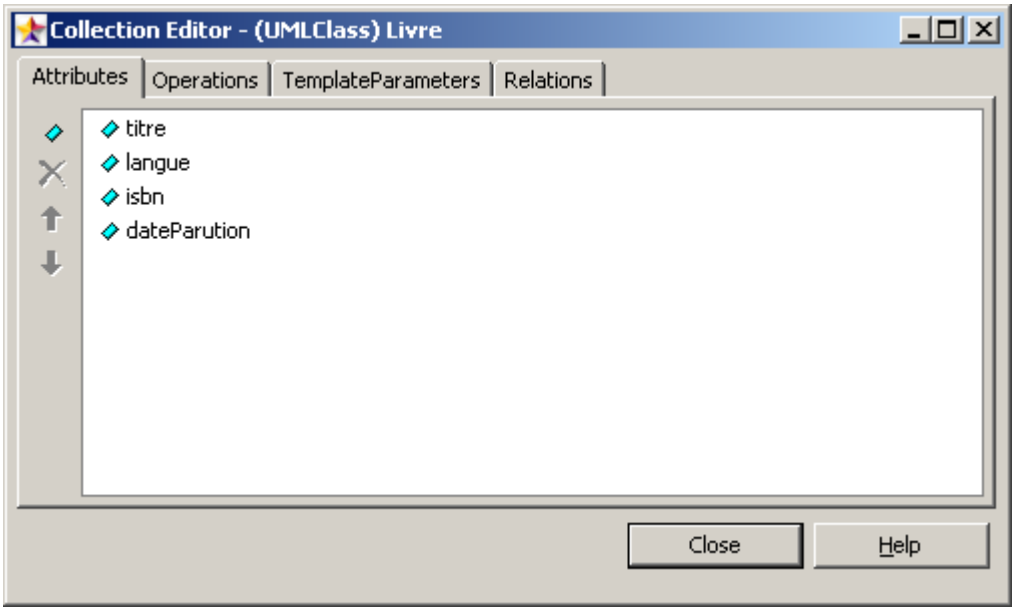
Nous allons maintenant ***modifier le niveau de visibilité*** de l'attribut **dateParution** afin de le rendre public comme les trois autres.
Pour cela double-cliquer directement dans le diagramme sur l'attribut **dateParution** de la classe **Livre**, puis cliquer sur le cadenas à gauche du bandeau gris afin de changer le niveau de visibilité. Choisir une visibilité publique (**+**) pour être en accord avec les autres attributs.

↳ Il nous reste plus qu'à rajouter l'attribut **prix**, ce que nous allons faire depuis la **fenêtre Properties**.
Cliquez dans le **Model Explorer** sur la classe **Livre** pour visualiser les propriétés de la classe dans la fenêtre **Properties**.
Pour l'instant, la propriété **Attributes** nous indique que la classe dipose de ***4 attributs***.



Pour rajouter un nouvel attribut depuis la fenêtre Properties, il suffit de cliquer sur la propriété attribut qui propose alors un bouton avec **...** qui va nous permettre de modifier les attributs.

Cliquer donc sur le bouton avec **...**
La fenêtre **Collection Editor** s'affiche alors à l'écran.



Pour ajouter un attribut, il suffit de cliquer sur le ***rectangle bleu*** proposé à gauche de la fenêtre portant l'onglet **Attributes**.
Un nouvel attribut est alors ajouté et vous pouvez alors le nommer **prix** à en renseignant l'attribut **Name** de la fenêtre **Properties.**

Remarque : Cette fenêtre appelée **Collection Editor** vous permet également :

- de supprimer un attribut : en le sélectionnant, puis en utilisant la croix
- ou de classer les attributs dans un ordre souhaité en utilisant les flèches bleues.

Pour revenir sur le diagramme, il suffit maintenant de cliquer sur le bouton **Close** pour fermer la fenêtre **Collection Editor.**

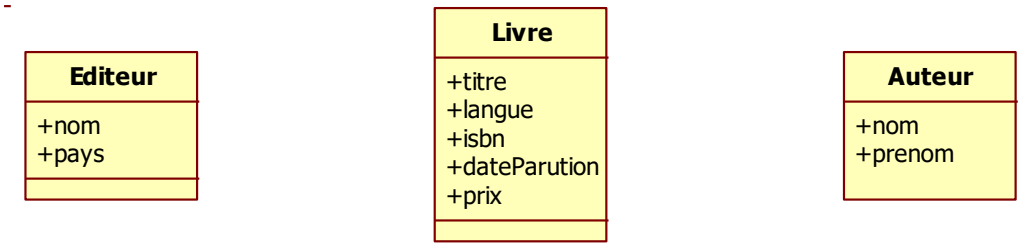
Remarque : Vous pouvez également accéder à la fenêtre **Collection Editor** directement depuis le diagramme. Pour cela cliquer une fois sur la classe **Livre** dans le diagramme, puis effectuer un clic droit de souris. Un menu apparaît où il est possible de choisir entre autres options : **Collection Editor**.

3.2.d Création des classes métiers : **`Auteur, Editeur`**

Dorénavant nous appelerons le paquetage **`bookinons.metier`**, le sous-paquetage **`métier`** du paquetage **`bookinons`**.

Depuis le **`Model Explorer`**, dans le paquetage **`bookinons.metier`**, créer 2 nouvelles classes : **`Auteur`** et **`Editeur`**.

Faites apparaître ses classes sur le diagramme de classes **`DCPers_Bookinons`** , et complétez-le de manière à obtenir :



3.3 Mise en place des associations

Nous allons maintenant représenter les associations entre classes.

3.3.a Création de l'association

Commençons par représenter sur le diagramme l'association entre **`Livre`** est **`Editeur`**.
Dans la **`Toolbox Class`**, cliquez sur **`Association`**. Cliquez ensuite sur la classe **`Editeur`** du diagramme et faîtes glisser la souris sans relâcher jusqu'à la classe **`Livre`**.
Une fois la classe **`Livre`** atteinte, vous pouvez relâcher : les deux classes sont alors reliées par une association.

3.3.b Nom de l'association

"Un **`Livre`** ***est edité par*** un **`Editeur`**"
Nous allons maintenant nous occuper de nommer l'association par : **`"<est edité par "`**
Deux solutions sont envisageables pour nommer l'association :

- soit renseigner la propriété **`Name`** dans la fenêtre **`Properties`** de l'association.
- soit double-cliquer au milieu de l'association sur le diagramme afin d'ouvrir une zone de saisie qui n'attend plus que le nom de l'association (il est important de se positionner au milieu de l'association pour ouvrir cette zone).

Choisissez l'une ou l'autre des solutions afin d'obtenir la représentation suivante :

3.3.c Multiplicité d'une association

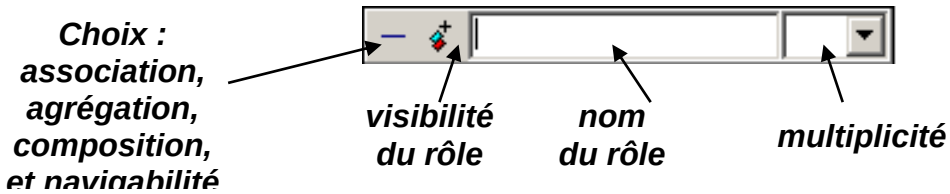
Rappelons la définition de la ***multiplicité*** : "La multiplicité indique le nombre d'instances qui participent à l'association".

"Un **`Livre`** est edité par **`1 Editeur`**"
Il n'y aura donc qu'une ***seule instance d'*****`Editeur`** par instance de **`Livre`**, c-a-d que la multiplicité du côté de l'**`Editeur`** doit être positioné à **1**.

Deux solutions sont envisageables pour indiquer la multiplicité :

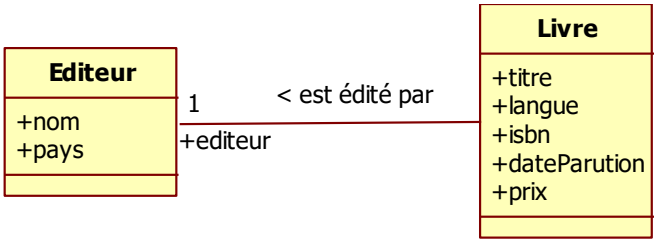
- soit depuis de la fenêtre **`Properties`**.
- soit directement depuis le diagramme.

Il est plus simple de passer par le diagramme pour indiquer les multiplicités.
Pour ouvrir sur le diagramme la bôite de dialogue permettant d'indiquer la multiplicité de l'association, vous devez vous ***positionner à l'extrémité de l'association côté*** **`Editeur`**, puis double-cliquez



La multiplicité peut être choisie via la liste déroulante à droite de la boîte de dialogue.
Sélectionner **1** dans la liste déroulante, puis cliquez ailleurs sur le diagramme afin de voir votre multiplicité apparaître.

Si vous souhaitez faire apparaître un rôle **`editeur`** sur votre diagramme. Ré-ouvrir la boîte de dialogue en cliquant sur l'extrémité de l'association et entrer **`editeur`** dans la zone de saisie pévue à cet effet. Vous obtenez alors la configuration suivante :

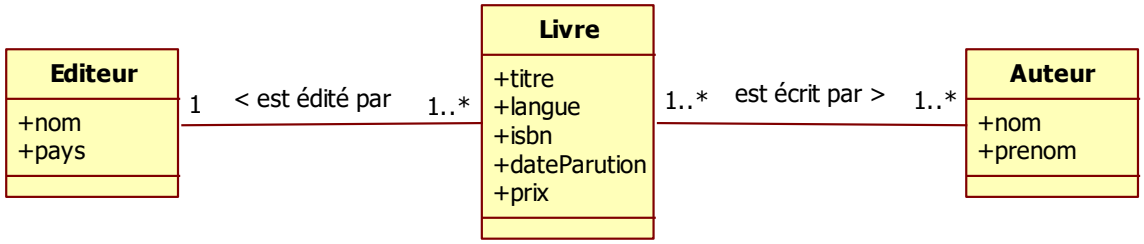


Mais est-il vraiment nécessaire que le rôle **`editeur`** apparaisse sur le diagramme ?

Pour le supprimer de notre diagramme, nous allons cette fois, utiliser la fenêtre **`Properties`**.
Sur le diagramme, cliquez une fois sur l'***association*** pour pouvoir visualiser les propriétés de l'association dans la fenêtre **`Properties`**.
La propriété **`End1.name`** contient le ***nom du rôle*** à l'extrémité de l'association que nous venons de saisir. Pour supprimer editeur du diagramme, effacez-le de la propriété **`End1.name`**.
La propriété **`End1.Multiplicity`** contient la ***multiplicité*** indiquée l'extrémité de l'association, côté **`Editeur`**.
`End1` concerne donc la ***première classe*** que nous avons sélectionnée lorsque nous avons créé l'association c-à-d dans notre cas Editeur. Cette information peut se retouver en consultant la valeur de la propriété : **`End1.Participant`** qui comme l'indique le petit cadenas est non modifiable …

`End2` concernera donc la seconde classe de l'association c-à-d la classe **`Livre`** comme l'indique la propriété **`End2.Participant`**.
Nous savons qu' **`1 Editeur`** peut éditer par **`1`** ou **`plusieurs Livre`**" .
La multiplicité **`1..*`** peut donc être directement tapée dans la propriété **`End2.Multiplicity`** ou sélectionnée à partir de la liste déroulante.

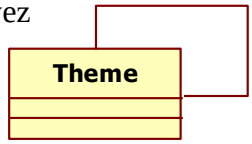
Créez de même une association entre **Livre** et **Auteu**r afin d'obtenir une représentation semblable à la représentation ci-dessous :



3.3.d Création d'une association réflexive

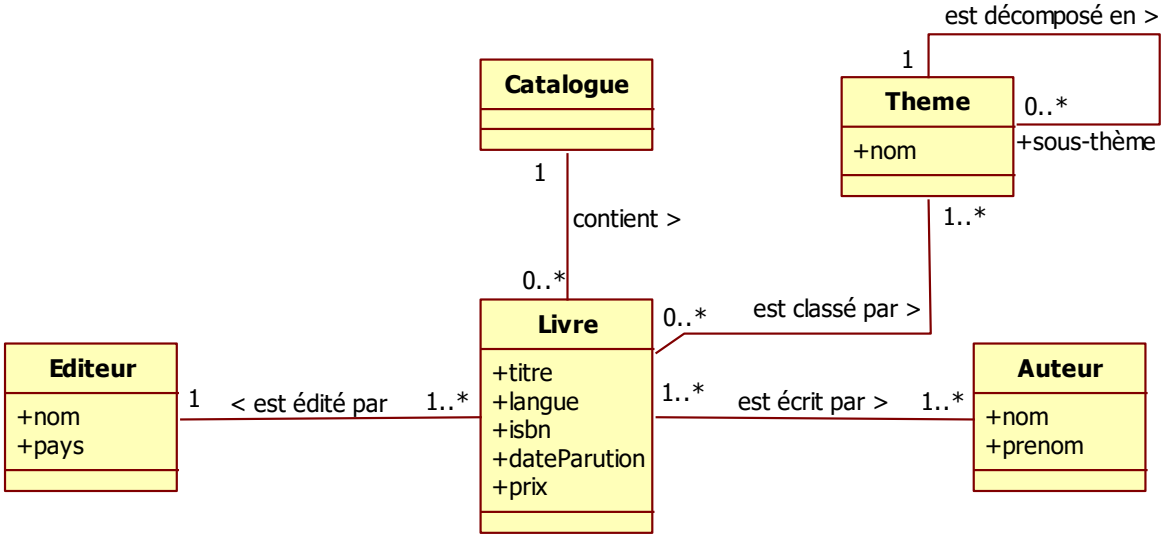
Dans le paquetage **bookinons.metier**, créer une nouvelle classe **Theme**.
Faîtes apparaître la classe **Theme** sur le diagramme de classes **DCPers_Bookinons**.

Pour représenter une association réflexive sur la classe Theme, vous devez sélectionner l'**Association** dans la **Toolbox** de **Class** puis cliquer une fois sur la classe **Theme** sur le diagramme.
L'association réflexive apparaît alors de la manière suivante :



3.3.e Première ébauche du diagramme de classes.

Après avoir rajouté dans le paquetage **bookinons.metier**, une nouvelle classe **Catalogue**, entraînez-vous à manipuler les éléments du diagramme de classes vus précédemment afin d'obtenir un diagramme de classes conformes à la représentation suivante :



3.4 Qualificateur

Dans le contexte du **Catalogue,** un **isbn** est attribué à **1** seul **Livre** (***ou à aucun***).
Il semble donc opportun de faire apparaître un qualificateur **isbn** pour affiner l'association entre **Catalogue** et **Livre**.
Un qualificateur est un ***attribut de l'association*** qui va permettre de sélectionner un sous-ensemble d'objets parmi l'ensemble des objets participant à l'association.
Faire apparaître un qualificateur dans notre modélisation signifie donc qu'il est aussi nécessaire de réduire la multiplicité indiquée.

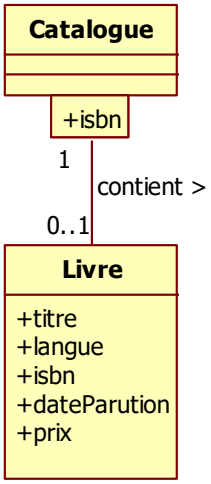
3.4.a Création d'un qualificateur :

Un qualificateur **est un attribut** de l'association.
Dans le diagramme de classes, nous allons donc commencer par ***sélectionner l'association*** qui relie la classe **`Catalogue`** à la classe **`Livre`**, puis après un clic droit, sélectionner **`Collection Editor`** pour faire apparaître la fenêtre correspondante.
Cette fenêtre propose 3 onglets : **`End1 Qualifiers`**, **`End2 Qualifiers`** et **`Relations`**

Le qualificateur **`isbn`** que nous souhaitons rajouter est un ***attribut de l'association*** qui apparaîtra à **l'extrémité de l'association côté `Catalogue`** (puisqu'on se place dans le contexte du **`Catalogue`**).
Au préalable, il est donc nécessaire de consulter la fenêtre **`Properties`** de l'association afin de savoir si la classe **`Catalogue`** correspond à **`End1.Participant`** ou à **`End2.Participant`**.

Si votre classe **`Catalogue`** est la valeur de **`End1.Participant`**,
restez dans l'onglet **`End1 Qualifiers`**.

Si votre classe **`Catalogue`** est la valeur de **`End2.Participant`**,
passez dans l'onglet **`End2 Qualifiers`.**



Cliquez alors sur le rectangle bleu et entrer **`isbn`** comme nom du qualificateur dans la fenêtre **`Properties`**. Appuyez sur **Close.** Le qualificateur **`isbn`** apparaît alors dans votre modélisation.

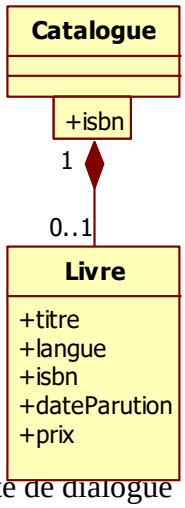
3.4.b Modification de la multiplicité :

Le qualificateur permet de réduire la multiplicité… Si on se place dans le contexte du **`Catalogue`,** un **`isbn`** peut être attribué à **`1 Livre`** ***ou à aucun*** : **`0..1`**

Vous devez donc changer la multiplicité du côté de la classe **`Livre`** et transformer le **`0..*`** en **`0..1`**

3.5 Association, Agrégation et Composition

Il est également possible d'affiner le diagramme de classes en faisant apparaître des agrégations et des compositions qui sont des cas particuliers d'association qui exprime une relation de contenance plus ou moins forte.



3.5.a Transformation d'une association en composition

Nous avons décidé de transformer l'association entre **`Catalogue`** et **`Livre`** en ***une composition*** puisque *"Un **`Catalogue`** **est composé** des **`Livres`**"* .
La composition devra donc être modélisée du côté de la classe **`Catalogue`**

Sur le diagramme de classes, placez-vous à l'extrémité de l'association entre **`Catalogue`** et **`Livre`**, côté **`Catalogue`** et double-cliquez sur cette extrémité. La boîte de dialogue concernant la modification de l'extrémité de l'association s'ouvre. Cliquez sur le trait à gauche de la boîte afin de faire apparaître les différents niveaux d'associations. Choisissez la **composition**.
Une composition signifie "***est composée de***", le nom l'association entre les classes **`Catalogue`** et **`Livre`** est maintenant explicite et superflu du fait de la présence de la notation graphique de la composition sur le diagramme. Supprimez donc **`"contient>"`** de votre diagramme afin d'en alléger les notations pour en faciliter sa lisibilité.

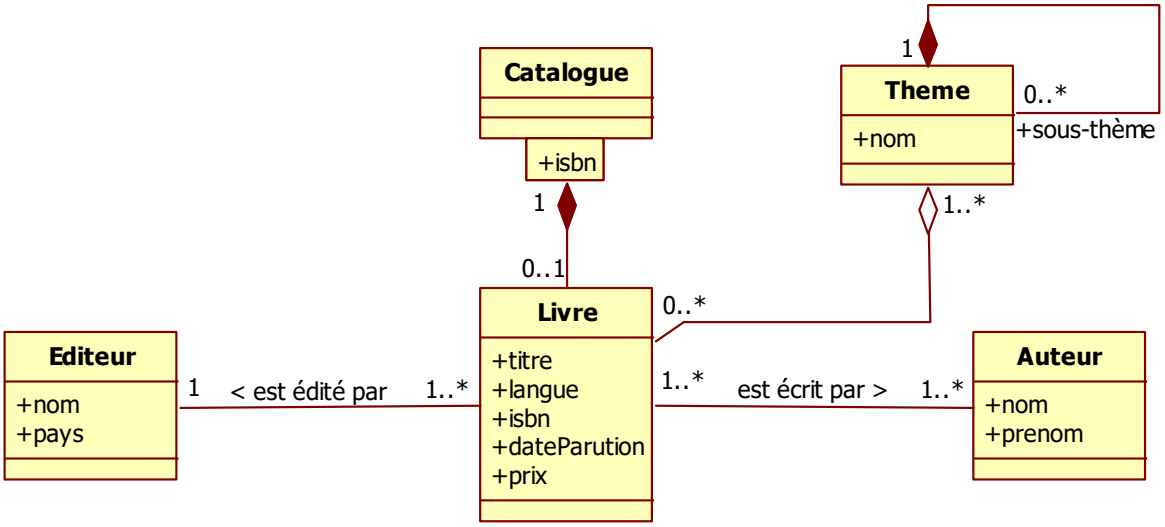
3.5.b Transformation d'une association en agrégation

De la même manière, vous pouvez ***transformer une association en agrégation***.
Pour vous entraîner, faîtes apparaître une ***agrégation*** entre la classe **`Theme`** et la classe **`Livre`** afin de modéliser le fait qu'un **`Theme`** ***contient*** des **`Livres`**

3.5.c Deuxième ébauche du diagramme de classes.

Rajouter également une ***composition*** sur la réflexive autour de la classe **`Theme`**.
En effet un **`Theme`** ***peut être décomposé*** en sous-thèmes.

Une fois toutes ces modifications effectuées, votre diagramme de classes doit être similaire au diagramme suivant :



3.6 Navigabilité

La ***navigabilité*** peut également être rajoutée sur le diagramme.
Par exemple, placez-vous sur l'extrémité de la composition entre **`Catalogue`** et **`Livre`** du côté de la classe **`Catalogue`** et double-cliquez afin d'ouvrir la boîte de dialogue permettant de modifier l'extrémité de cette association. Ouvrir la première zone (celle qui propose le choix entre association, agrégation et composition) et allez décocher l'option **`isNavigable`**.
Une flèche apparaît alors sur l'autre extrémité de l'association (c-a-d du côté de **`Livre`**).
L'association n'est plus navigable que dans un seul sens :

> Une instance de **`Catalogue`** a accès à une instance de **`Livre`** c-a-d que le catalogue "connaît" (voit) les livres…
> Une instance de **`Livre`** n'a plus accès à une instance de **`Catalogue`** c-a-d que les livres ne "connaissent" (voient) pas le catalogue…

Rappelons que si rien n'est indiqué l'association est par défaut navigable dans les 2 sens (c-a-d que c'est comme s'il y avait des flèches dans les deux sens).

3.7 Diagramme de classes participantes

La modélisation UML de nos applications sera en fait composée de plusieurs diagrammes de classes :
- ***un seul diagramme de classes de persistantes*** qui sera commun à tous les UC de l'application
- et ***autant de diagrammes de classes de participantes que de UC*** dans l'application.

Dans le paquetage **UC Rechercher un ouvrage**, ajouter un **D**iagramme de **C**lasses que vous appelerez **DC_RechercherUnOuvrage.** En fin d'analyse, ce **D**iagramme de **C**lasses sera censé regrouper ***toutes les classes participantes*** au UC **RechercherUnOuvrage**.

Faîtes glisser sur votre diagramme les classes **Livre, Auteur, Editeur, Catalogue** et **Theme** du paquetage **bookinons.metier** du **Model Explorer**.

Vous pouvez alors constater que les relations créées dans le diagramme de classes précédent ont été conservées : c'est pour cela qu'il est indispensable de sélectionner l'option **delete from Model** lorsqu'on souhaite supprimer définitivement un élément de la modélisation …

Le Diagramme Classes Participantes contiendra toutes les classes participants au Use Case y compris les classes d'analyse de Jacobson qui pourront être rajoutées lors de la modélisation du diagramme de séquence…

Exercice 4 : Axe dynamique : la vue Logique et le diagramme de séquence

Le diagramme de séquence est un diagramme d'**interaction** qui représente une **vue dynamique** du système. Le diagramme de séquence montre les interactions entre les acteurs et le système selon un point de vue temporel : il représente un ensemble de messages échangés entre les acteurs et le système, ordonnés chronologiquement.

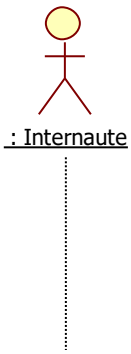
En général, un diagramme de séquence permet de représenter graphiquement **un scénario en particulier** , montrant ainsi la logique de déroulement des actions décrites.
Il est d'usage de modéliser d'abord le flot de base, puis le ou les flots alternatifs,
Chaque cas d'utilisation pourra être décrit par un ou plusieurs diagrammes de séquence.

Nous allons commencer par représenter le diagramme de séquence décrivant le flot de base du **UC Rechercher un ouvrage**. Ajouter dans le package **UC Rechercher un ouvrage** un diagramme de séquence (**Sequence Diagram**) que vous appelerez **DS_RechercherOuvrage**.
Ce diagramme devient alors le diagramme actif de votre fenêtre de travail.

4.1 Mise en place des objets sur le diagramme de séquence :

4.1.a Un acteur

L' acteur est le premier objet à positionner sur le diagramme de séquence.
Il s'agit de l'acteur principal du UC qui va interagir avec le système.
Pour le use case **Rechercher un ouvrage** c'est l'**Internaute.**
Il est donc maintenant nécessaire d'ouvrir le paquetage **UC des Internaute** de la **Cible 1** de la vue **Scenarios** afin de cliquer sur l'acteur **Internaute** et de le faire glisser à complètement à gauche du diagramme **DS_RechercherOuvrage** : un objet **Internaute** apparaît alors dans le diagramme avec sa ligne de vie.

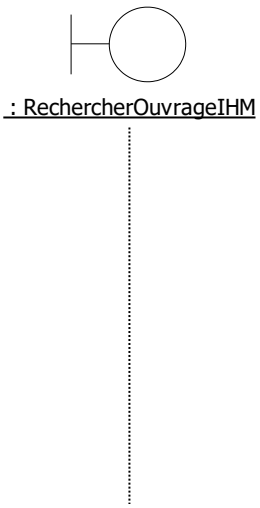


4.1.b Une interface graphique (IHM)

Le deuxième objet du diagramme de séquence doit être **une interface graphique**.
Par convention le nom de la classe de cet objet porte le ***nom du UC*** suivi de ***IHM***.
Cette classe permet à l'acteur d'interagir avec le système : c'est un dialogue qui doit se trouver dans la couche **presentation**. Dans le paquetage **bookinons.presentation** de la **Logical**

View, ajouter une classe que vous appelerez **RechercherOuvrageIHM.** Une fois la classe créée, cliquez sur cette classe et faîtes glisser votre souris sur le diagramme **DS_RechercherOuvrage** afin de créer sur ce diagramme un objet de la classe **RechercherOuvrageIHM.**

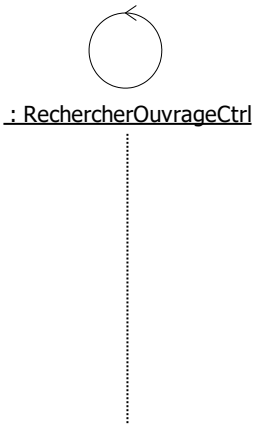
Remarque : Cet objet n'a pas de nom, on dit qu'il s'agit d'un ***objet anonyme***. Les diagrammes de séquence sont généralement réalisés avec des objets anonymes.



Dans un diagramme de séquence, il est plus convivial de visualiser les objets sous la ***forme graphique des stéréotypes de Jacobson***.
Dans un premier temps, il est donc nécessaire de stéréotyper la classe **RechercherOuvrageIHM** comme <<**boundary**>>**.**
Pour cela, dans le **Model Explorer,** cliquez sur la classe **RechercherOuvrageIHM** afin d'ouvrir sa fenêtre **Properties**.
Cliquez sur la propriété **Stereotype**, puis sur les 3 points (…) afin de sélectionner dans la boîte de de dialogue ouverte le stéréotype **boundary**, puis valider avec **OK**.
Il faut maintenant ***changer le format d'affichage*** de l'objet. Cliquez sur l'objet dans le diagramme afin de le sélectionner, puis choisissez l'option **Iconic** du menu **Format** → **Stereotype Display** pour obtenir une représentation graphique du stéréotype de Jacobson <<**boundary**>>**.**

4.1.c Un contrôleur de UC (Ctrl)

Le troisième objet du diagramme de séquence doit être **un contrôleur.**
Par convention le nom de la classe de cet objet porte le ***nom du UC*** suivi de ***Ctrl.***

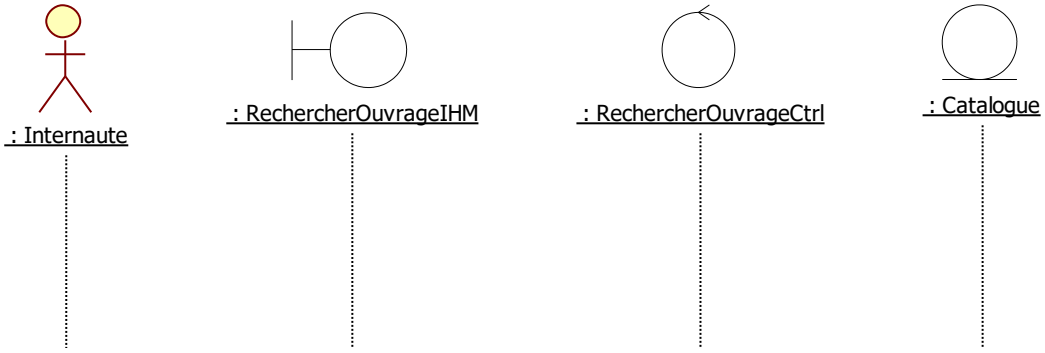


Cette classe va ***contrôler*** et orchestrer l'application : c'est un contrôle qui doit se trouver dans la couche **application**. Dans le paquetage **bookinons.application** de la **Logical View,** ajouter une classe que vous appelerez **RechercherOuvrageCtrl.**
Pour pouvoir visualiser cet objet dans le diagramme sous la ***forme graphique des stéréotypes de Jacobson***, vous devez attribuer à cette classe le stéréotype <<**control**>>**.**
Une fois la classe créée, cliquez sur cette classe et faîtes glisser votre souris sur le diagramme **DS_RechercherOuvrage** afin de créer sur ce diagramme un objet de la classe **RechercherOuvrageCtrl.**
Il ne reste plus qu'à changer le format d'affichage afin de la faire appraître sous forme **Iconic.**

4.1.d Une entité

Les objets de la couche ***métier*** seront représentés sur le diagramme de séquence à l'aide du stéréotype de Jacobson <<**entity**>>.
De la **Logical View,** cliquez dans le paquetage **bookinons.metier** sur la classe **Catalogue**. Pour pouvoir visualiser cet objet dans le diagramme sous la ***forme graphique des stéréotypes de Jacobson***, vous devez attribuer à cette classe le stéréotype **entity.**
Faîtes alors glisser la classe sur le diagramme **DS_RechercherOuvrage** afin de créer sur ce diagramme un objet de la classe **Catalogue.**
Il ne reste plus qu'à changer le format d'affichage afin de la faire appraître sous forme **Iconic.**

Vous venez de mettre en place le squelette du diagramme de séquence qui respecte un modèle **MVC** (une classe métier pour le **M**odèle, une classe IHM pour la **V**ue et une classe Ctrl pour le **C**ontrôle)
Nous allons maintenant pouvoir représenter sur ce diagramme l'enchaînement des messages.



4.2 Mise en place des messages sur le diagramme de séquence :

4.2.a Message pour marquer le début du diagramme

Un diagramme de séquence permet de modéliser ***un des scénarios d'un Cas d 'Utilisation***.
Nous souhaitons faire apparaître en début et en fin de diagramme des messages "conventionnés" pour bien montrer le début et la fin des actions du scénario. (Ceci est une convention qui propre à ce cours...)

Pour lancer le scénario, nous ferons toujours en sorte que l'acteur commence par lancer le message **debut** à l'IHM…
Du point de vue de la conception objet, lancer un ***message synchrone*** à un objet revient à provoquer l'exécution d'une opération définie dans la classe de cet objet.
La classe **RechercherOuvrageIHM** doit donc disposer d'une opération **debut()**.
Avant de représenter le message sur le diagramme, il faut donc ajouter l'opération dans la classe.

Depuis le **Model Explorer**, sélectionnez la classe **RechercherOuvrageIHM** du paquetage **bookinons.presentation**. Une fois la classe sélectionnée, effectuez un clic droit qui vous permettra de choisir le menu **Add → Operation** et de saisir ainsi directement le nom de l'opération ou de le taper dans le **Name** de la fenêtre **Properties**. Vous appelerez cette opération : **debut**.
Rappel : Par convention, une opération commencera par une minuscule.

Remarque : Pour ajouter cette opération, vous auriez également pu passer par le diagramme de classes participantes du use case **Rechercher Ouvrage** qui doit contenir à la fin de l'analyse toutes les classes qui apparaissent dans le Use Case.
Vous pouvez donc dès maintenant faire glisser la classe **RechercherOuvrageIHM** sur le diagramme de classes **DC_RechercherUnOuvrage**, ainsi que la classe **RechercherOuvrageCtrl**…
Nous nous occuperons de compléter ce diagramme avec les associations un peu plus tard …

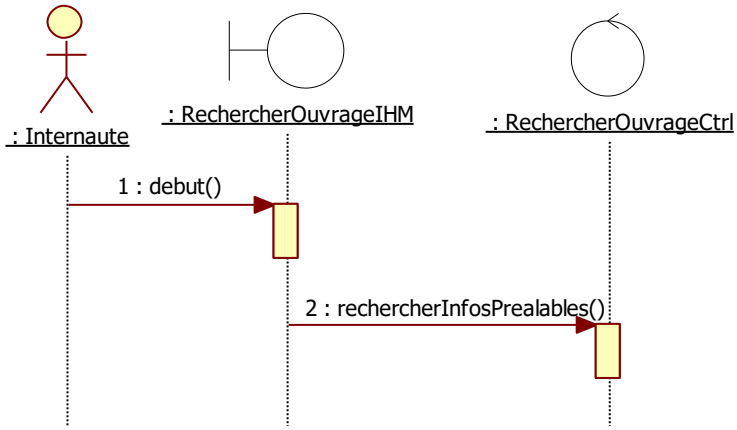
Dès lors que l'opération **debut** est disponible dans la classe **RechercherOuvrageIHM** , l'acteur peut envoyer un message **debut** vers l'objet IHM.
Pour représenter un message sur le diagramme, sélectionnez dans la **Toolbox** de **Sequence** un objet **Stimulus**, puis cliquez sur la ligne de vie de l'acteur et faîtes glisser la souris jusqu'à la ligne de vie de l'IHM, puis relâcher.
Une boîte de dialogue s'ouvre et vous demande de renseigner le nom du message.
Vous devez aller chercher le nom de l'opération **debut** dans la classe **RechercherOuvrageIHM**.
Pour cela, cliquez sur le signe **=** à gauche du bandeau gris afin de récupérer la méthode **debut** de la classe **RechercherOUvrageIHM**.

Ne tapez pas directement le nom du message dans la boîte de dialogue !!! : cela n'enrichirait en aucun cas la classe. En effet si le message est tapé directement dans la boîte de dialogue, aucune opération ne sera ajoutée à la classe … Or le but du diagramme de séquence est la découverte de nouveaux objets du système mais aussi ***l'enrichissement des classes en ajoutant notamment les opérations*** qui sont découvertes lors de l'ellaboration de ce diagramme…

A Retenir !!! Lorsque vous voulez rajouter un message dans le diagramme de séquence, commencez d'abord par rajouter une opération dans la classe de l'objet qui va recevoir ce message …

4.2.b Messages du diagramme de séquence …

Le second message que l'on souhaite représenter est le message **`rechercherDonneesPrealables`** qui va être envoyé de l'objet **`IHM`** vers l'objet **`Ctrl`**. Complétez la classe **`RechercherOuvrageCtrl`** avec l'opération **`rechercherInfosPrealables`**, puis rajoutez sur le diagramme ce nouveau message afin d'obtenir une représentation similaire à la représentation suivante :



4.2.c Paramétrage du diagramme de séquence : numérotation et activité

Par défaut, le diagramme de séquence numérote les messages et montre la période d'activité (ou "focus of control"). Il est tout à fait possible de modifier ce paramètrage.
Pour cela, depuis le **`Model Explorer`**, cliquer dans la **`Logical View`** sur le diagramme de séquence **`DS_RechercherOuvrage`** situé rappelons-le dans le paquetage du Use Case **`Rechercher un Ouvrage.`** La fenêtre **`Properties`** nous indique alors les propriétés du diagramme de séquence qu'il est possible de modifier.

- si vous décochez la propriété **`ShowSequenceNumber`**, la numérotation des messages ne sera plus visible sur le diagramme.
- si vous décochez la propriété **`ShowActivation`**, les périodes d'activités ne seront plus visibles sur le diagramme.
- la propriété **`MessageSignature`**permet quant à elle de modifier l'apparence de la signauture des messages.

…Pour continuer le tutoriel, vous pouvez par exemple choisir de visualiser la numérotation des messages, mais de ne pas visualiser l'activation…

4.2.d Message réflexif dans le diagramme de séquence

Ajouter dans la classe **RechercherOuvrageIHM**, une opération que vous appelerez **afficherEcranDeRecherche.**

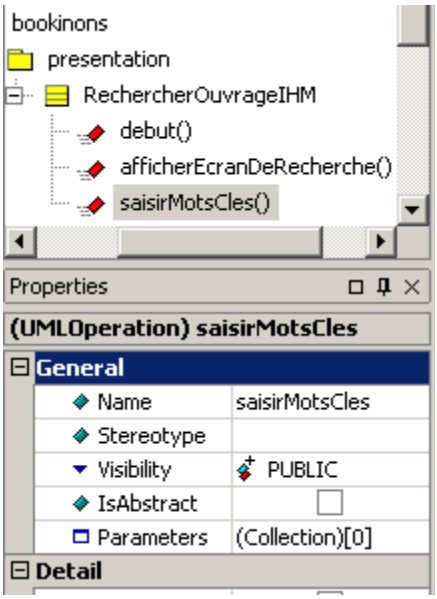
Il est tout à fait possible qu'un objet s'envoie un message à lui-même : c'est ce que l'on appelle un ***message réflexif*** : c'est ce que nous allons représenter maintenant sur la ligne de vie de l'objet IHM. Sélectionnez dans la **Toolbox** de **Sequence** un objet **Stimulus**, puis cliquez sur la ligne de vie de l'objet de la classe **RechercherOuvrageIHM** et relâcher la souris. La boîte de dialogue s'ouvre : il ne vous reste plus qu'à aller chercher la méthode **afficherEcranDeRecherche** en cliquant sur le signe **=** du bandeau gris.

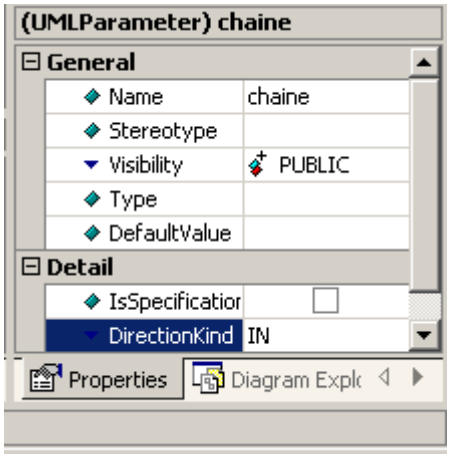
4.2.e Signature du message : visualisation des paramètres:

Ajoutez dans la classe **RechercherOuvrageIHM**, une opération que vous appelerez **saisirMotsCles.**

L'acteur est là pour interagir avec le système.
Sur votre diagramme, faites représentez maintenant un message **saisirMotsCles** envoyé de l'acteur vers l'objet IHM.

Pour passer le paramètre **chaine** à l'opération **saisirMotsCles**, il suffit maintenant de cliquer dans le **Model Explorer** sur la méthode **saisirMotsCles** afin de visualiser dans la fenêtre **Properties** les propriétés de l'opération. Cliquez sur la propriété **Parameters**, puis sur les 3 points (...) une boîte de dialogue s'ouvre. Dans l'onglet **Parameters**, cliquez sur la flèche jaune, et tapez **chaine** dans la propriété **Name** de la fenêtre **Properties**. Cliquez sur **Close**.





Rien n'est apparu sur le diagramme de séquence… C'est normal !
Il faut maintenant changer la propriété **MessageSignature** du diagramme de séquence et la passer par exemple à **NAMEONLY**, si vous voulez voir apparaître le nom du paramètre sur le diagramme.

Remarque : la fenêtre **Properties** d'un paramètre (comme **chaine**) permet entres autres, si on le souhaite, de pouvoir choisir un **Type** pour le paramètre, mais également d'indiquer s'il s'agit d'un paramètre d'entrée, de sortie ou de retour via la propriété **DirectionKind**, ...

4.3 Mise en place des interfaces sur le diagramme de séquence :

Il faut savoir qu'en plus des IHM **(<<boundary>>**), des contrôleurs (**<<control>>**) et des classes métiers(**<<entity>>)** , on peut trouver dans le diagramme de séquence des objets stéréotypés **<<interface>>**.
Nous allons regrouper les objets interface dans un nouveau paquetage que nous appelerons **service**.
Créez donc dans le paquetage **bookinons**, un nouveau paquetage appelé **service**.

4.3.a Mise en place de l'interface `IRechercherNouveautes` :

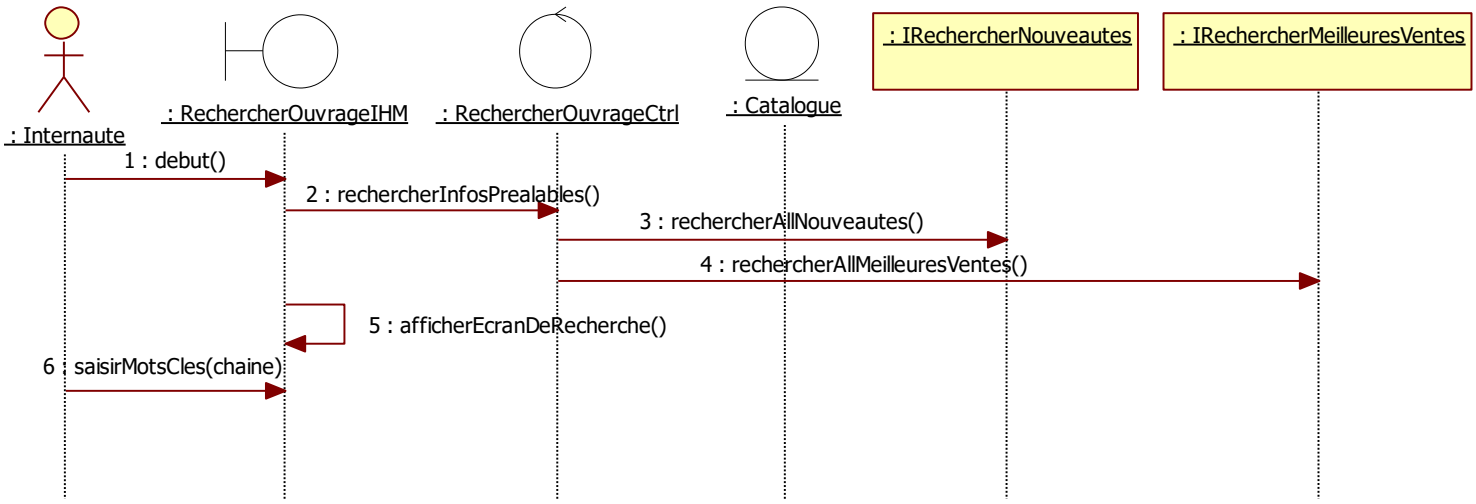
Placez-vous dans le **`Model Explorer`** sur le nouveau paquetage **`service`**.
Pour créer une interface dans ce paquetage, il suffit d'effectuer un clic droit puis de choisir **`Add → Interface`**.
La nouvelle interface est créée, entrez alors **`IRechercherNouveautes`** comme **`Name`** dans la fenêtre **`Properties`**.
Dans le **`Model Explorer`**, sélectionnez cette nouvelle interface et faîtes-la glisser sur le diagramme de séquence afin de créer un nouvel objet de type **`IRechercherNouveautes`**.

Ajouter dans l'interface **`IRechercherNouveautes`**, une opération que vous appelerez **`rechercherAllNouveautes.`**
Sur votre diagramme, ajoutez en 3ème position un message **`rechercherAllNouveautes`** qui part du contrôleur (objet de type **`RechercherOUvrageCtrl`**)et qui arrive sur l'objet de type **`IRechercherNouveautes`**.

Remarque: il est tout à fait possible d'insérer un message au milieu de la séquence déjà existante. Lors de l'ajout de ce nouveau message, vous avez sûrement constaté que les messages se sont renumérotés automatiquement en fonction de leur position dans la séquence …

4.3.a Création de l'interface `IRechercherMeilleuresVentes`

De la même manière, créez une interface **`IrechercherMeilleuresVentes`** avec une opération **`rechercherAllMeilleuresVentes`** et modifier le diagramme afin d'obtenir une représentation similaire à la représentation suivante



Remarques :

- ***Rappel définition d'une interface :*** Au sens UML, une ***interface*** est un ***ensemble d'opérations utilisé pour spécifier le service (contrat)*** d'une classe ou d'un composant. Une interface ne peut définir ***ni attribut***, ***ni association navigable vers d'autres classes*** et toutes les opérations d'une interface sont abstraites.
- En principe, on choisit de représenter une interface sur un diagramme de séquence lorsque le service demandé est un service extérieur à l'application.
- Il est souvent recommandé de faire commencer le nom de l'interface avec la lettre **I**.

Si vous le souhaitez, pour que votre diagramme soit encore plus explicite, vous pouvez faire apparaître le stéréotype **<<interface>>** sur les objets **IRechercherNouveautes** et **IrechercherMeilleuresVentes.**
Pour cela, cliquez sur le diagramme sur l'objet **IRechercherNouveautes** par exemple, et dans la fenêtre **Properties**, tapez directement **interface** comme valeur de la propriété **Stéréotype**.
Pour voir apparaître le stéréotype sur le diagramme, il faut que *le format d'affichage des objets soit* ***Textual.***

4.4 Construction complète du diagramme de séquence du flot de base

4.4.a Diagramme à représenter

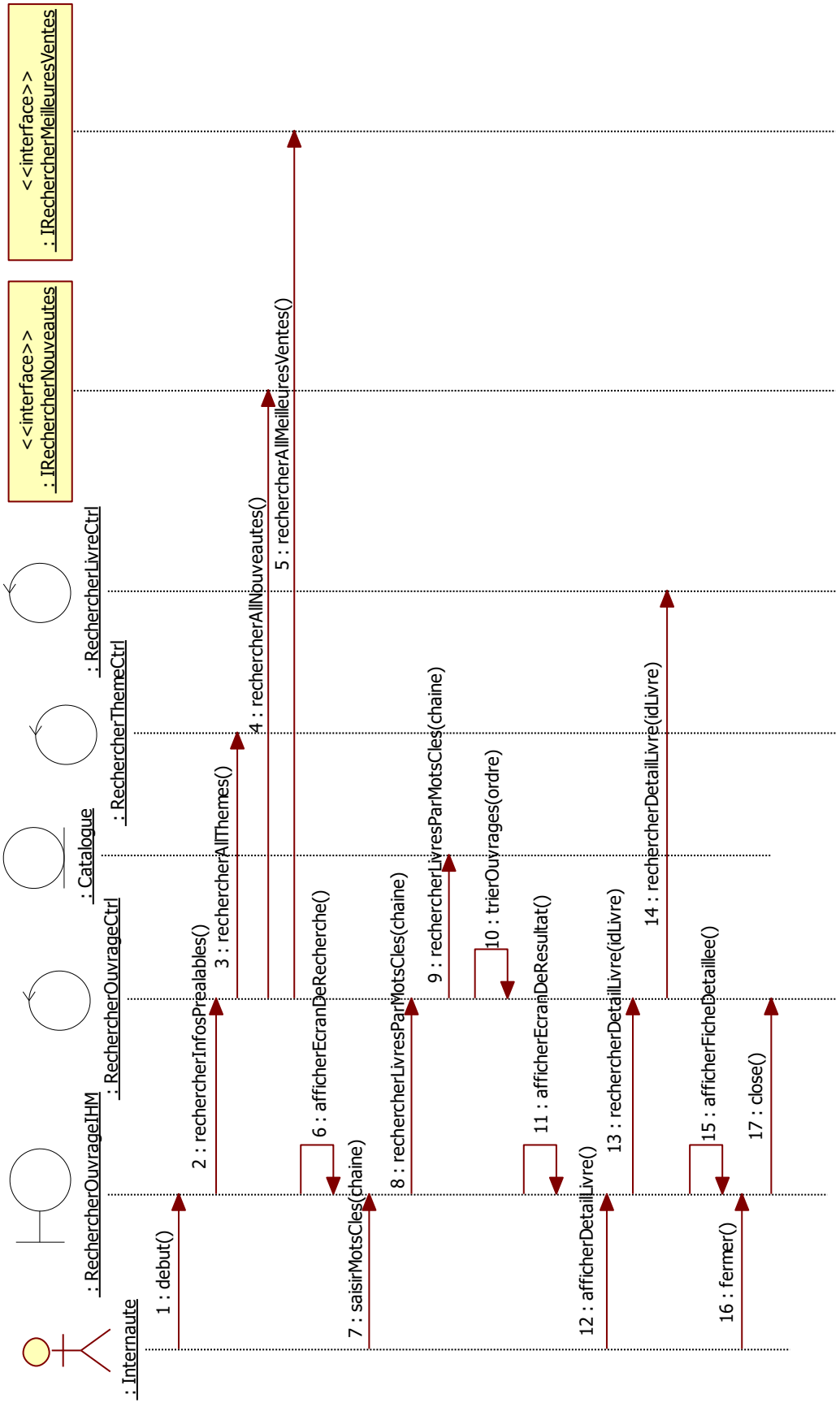
Pour vous entraîner, vous trouverez sur la page suivante une première ébauche du diagramme de séquence du **UC Rechercher un ouvrag**e que vous devez maintenant reproduire sous StarUML.

A propos des contrôleurs RechercherLivreCtrl et RechercherThemeCtrl
La classe **RechercherLivreCtrl** correspond à un contrôleur du UC du **CRUD** concernant le paramètre **Livre** de l'application.
De même, la classe **RechercherThemeCtrl** correspond à un contrôleur du UC du **CRUD** concernant le paramètre **Theme** de l'application.
Les classes **RechercherLivreCtrl** et **RechercherThemeCtrl** devront donc être stockées dans le paquetage **application** de **bookinons**.

A propos du type de message envoyé … :
Sur ce diagramme, les messages de retour ne sont volontairement pas indiqués afin de ne pas alourdir le diagramme. En effet, les messages représentés sont tous des messages synchrones (puisqu'il s'agit d'appel d'opérations) : le retour est donc implicite !
Rappel: Le retour est implicite avec un appel synchrone !

Si toutefois, vous vouliez représenter d'autres messages que des ***messages synchrones*** (**CALL**), vous pouvez le faire à partir des propriétés d'un objet de type **Stimulus** en intervenant sur la propriété **Action Kind** de la fenêtre **Properties** d'un ***Stimulus*** qui vous permet alors de choisir entre :

- **CALL** (***synchrone***),
- **SEND** (***asynchrone***),
- **RETURN** (***message de retour***),
- **CREATE** et
- **DESTROY**.


: Internaute
: RechercherOuvrageIHM
: RechercherOuvrageCtrl
: Catalogue
: RechercherThemeCtrl
: RechercherLivreCtrl
<<interface>>
: IRechercherNouveautes
<<interface>>
: IRechercherMeilleuresVentes
1 : debut()
2 : rechercherInfosPrealables()
3 : rechercherAllThemes()
4 : rechercherAllNouveautes()
5 : rechercherAllMeilleuresVentes()
6 : afficherEcranDeRecherche()
7 : saisirMotsCles(chaine)
8 : rechercherLivresParMotsCles(chaine)
9 : rechercherLivresParMotsCles(chaine)
10 : trierOuvrages(ordre)
11 : afficherEcranDeResultat()
12 : afficherDetailLivre()
13 : rechercherDetailLivre(idLivre)
14 : rechercherDetailLivre(idLivre)
15 : afficherFicheDetaillee()
16 : fermer()
17 : close()

4.4.b Messages de fin de diagramme : `fermer` et `close`

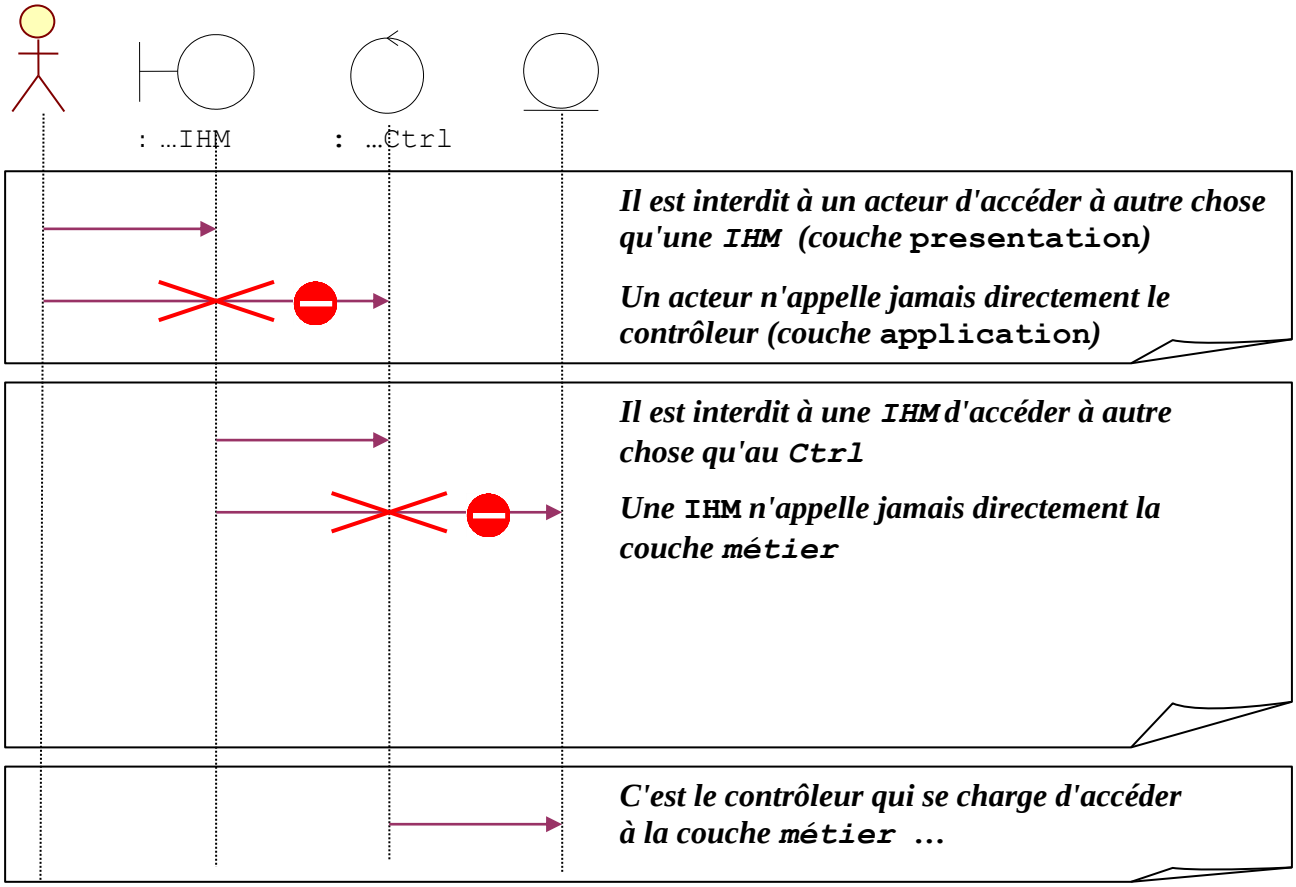
Notez bien que pour montrer la fin du scénario, notre convention demande à ce que les diagrammes de séquence se finissent toujours par 2 messages : **`fermer`** et **`close`**. En effet, l'acteur devra envoyer un message **`fermer`** à l 'IHM pour signifier qu 'il souhaite quitter le scénario et l'IHM devra envoyer un message **`close`** au contrôleur pour bien montrer la fin des messages.

4.4.c Rappel des règles de construction dans un diagramme

Les diagrammes de séquence que vous écrirez dorénavant devront respecter des règles de construction précises afin de respecter un pattern d'analyse **`MVC`**

(**`M`** pour **`M`**odèle c-à-d les classes **`<<entity>>`**,
`C` pour **`C`**ontrôleur c-a-d les classes **`<<control>>`**
et **`V`** pour **`V`**ue c-à-d les classes **`<<boundary>>`**)

La mise en place de l'application sous forme de couches logicielles (**`MVC)`** permet d'isoler les comportements/actions de chacune des couches. Il vous faut bien garder à l'esprit que les messages ne doivent pas "sauter" de couches…



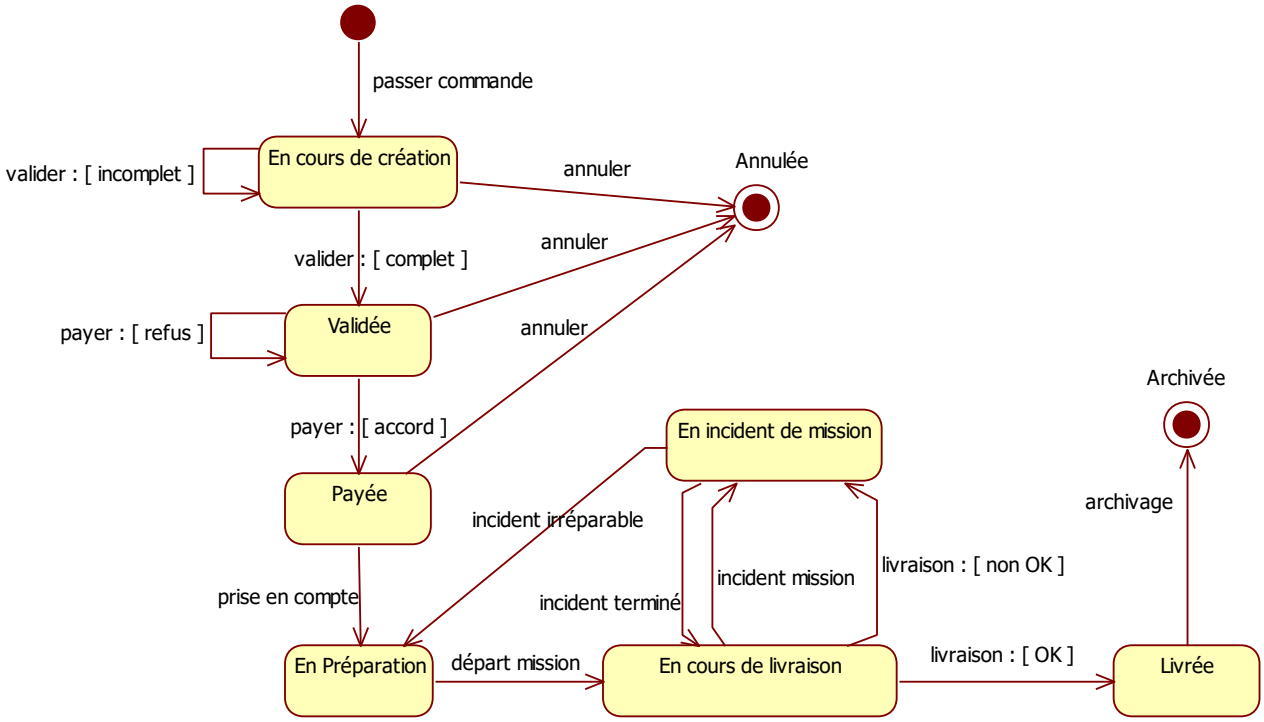
Exercice 5 : Axe dynamique : La vue logique et le diagramme d'états-transitions

En UML, ***un diagramme d'états-transitions*** représente une vue sur un objet : il décrit ***le Cycle de Vie de l'Objet*** c-à-d qu'il montre comment l'objet réagit à certains événements en fonction de son état courant et comment il passe dans un nouvel état. Le diagramme d'états-transitions UML vise donc à montrer les ***différents états*** et les ***transitions possibles des objets d'une classe à l'exécution.***
Il y aura donc ***autant de diagrammes d'états-transitions que d'objets complexes*** dans le système.

De plus, comme un même objet peut intervenir dans plusieurs cas d'utilisations, le ***diagramme d'états-transitions*** est considéré comme un diagramme ***inter Use-Case.*** Les diagrammes d'états-transitions se trouveront donc directement **à la racine de la vue logique au même niveau que les paquetages des Use Case**. Cependant, pour une meilleure lisibilité, il est possible de créer un paquetage **`DET`** dans lequel pourront être stockés tous les **D**iagrammes d'**E**tats-**T**ransitions.

Placez-vous dans le **`Model Explorer`** sur sur **`Logical View`** et créez un nouveau **`Package`** que vous appelerez **`DET`**. Dans ce nouveau paquetage, créez un diagramme d'états-transitions (**`StatechartDiagram`**) que vous appelerez **`DET_Commande.`** Ce diagramme devient alors le diagramme actif de votre fenêtre de travail.

A partir de votre connaissance actuelle de StarUML et en utilisant l'aide sur le logiciel, réalisez maintenant le diagramme d'états-transitions suivant.

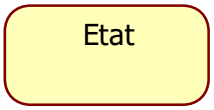


Remarques :

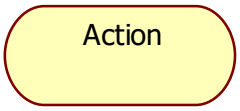
- Vous pouvez constater que le diagramme d'activité est très proche du diagramme d'états-transitions. Les 2 diagrammes se distinguent par leur domaine de prédilection :
  - dynamique et plutôt orienté utilisateur pour le diagramme d'activité
  - plus proche des objets métiers pour le diagramme d'états-transitions.

- De plus, les briques de base ont une représentation graphique différentes :

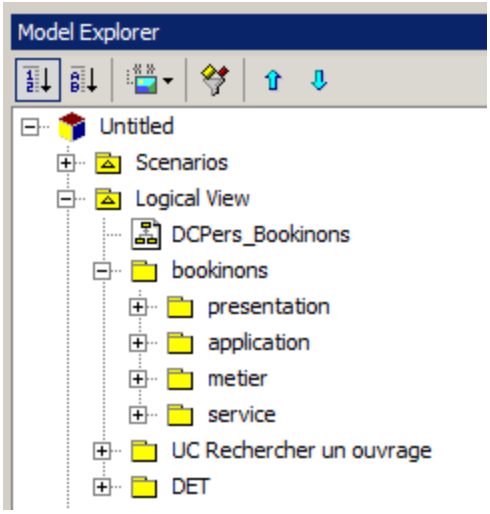
  - un **état** sera modélisé par un rectangle à coins arrondis



  - une **action (ou activité)** sera modélisé par :



Rappel de l'organisation de la vue logique:



→ La vue logique contiendra **1 paquetage par Use Case**.
→ La vue logique contiendra également un paquetage portant par exemple le nom du projet et destiné à recevoir les classes modélisées. Un découpage en couches sera utilisé pour stocker les **classes**.
Ce découpage comprendra au mimum les 3 paquetages suivants : **`presentation, application, metier`**.

→ Il y aura ***1 seul diagramme d'activité par Use Case***. Ce diagramme d'activité se trouvera dans la vue logique, dans le package correspondant au cas d'utilisation modélisé.

→ Il y aura ***1 diagramme de séquence par Use Case représentant le scénario nominal du Use Case***
Evéntuellement des diagrammes de séquence pour des flots alternatifs "délicats".
Ce(s) diagramme(s) de séquence se trouvera(ont) dans la vue logique, dans le package correspondant au cas d'utilisation modélisé.

→ Il y aura également ***1 diagramme de classes participantes*** par Use Case qui se trouvera dans le package correspondant au cas d'utilisation modélisé.

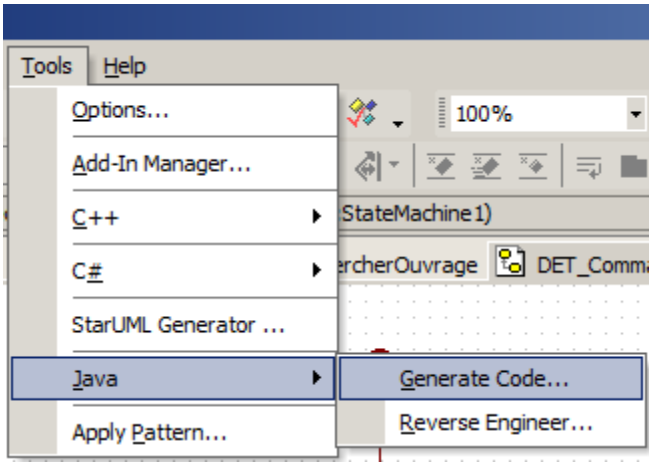
→ Il y aura ***1 diagramme de classes persistantes*** qui sera commun ***à tous les Use Case*** de l'application et qui sera stocké directement à la racine de la vue logique au même niveau que les paquetages des Use Case car ce diagramme concerne tous les UC.

→ Il y aura également ***1 diagramme d'états-transitions***, directement stocké à la racine de la vue logique au même niveau que les paquetages des Use Case, puisque le diagramme d'état transition est considéré comme un diagramme ***inter Use-Case.*** Pour une meilleure lisibilité, il est possible de créer un paquetage **`DET`** dans lequel pourront être stockés tous les **D**iagrammes d'**E**tats-**T**ransitions.

Exercice 6 : Génération de code Java : Forward Engineering

A partir entre autres des diagrammes de classes, les AGL peuvent produire directement du code source (squelettes des classes) dans les principaux langages objets (Java, C++, C #, Phyton…) :
c 'est du ***forward engineering***

Sous Star UML, pour générer le squelette des classes Java modélisées cliquez sur :
Tools → Java → Generate Code ...



Une erreur peut apparaître.... En effet, il faut savoir qu'avant la génération du code, il faut au préalable **avoir inclus au moins une fois le profil adéquat** dans les modèles.
Pour cela, cliquez **Model → Profiles** puis choisissez le profil que vous souhaitez : cliquez par exemple sur **Java Profile** puis **Include >**.
Fermez ensuite le **Profile Manager** en cliquant sur **Close.**

Une fois le profil ajouté et l'option **Tools → Java → Generate Code ...** choisie, une fenêtre **Java Code Generation** s'ouvre.

Sélectionnez le paquetage **bookinons** de la **Logical View** et cliquez que **Next**.
La fenêtre suivante permet de sélectionner les classes à générer. Par défaut, elles sont toutes sélectionnées, il vous suffit donc de cliquer simplement sur **Next**.

La fenêtre suivante permet de sélectionner le répertoire dans lequel seront enregistrées les classes à génerer. Créez à partir de votre explorateur de fichiers un répertoire du type **BookinonsGenerationCode**. Une fois ce répertoire créé, sélectionnez-le comme **OutputDirectorySetup** depuis la fenêtre **Java Code Generation** de Star UML. Cliquez sur **Next**. Laissez pour l'instant les options de **l'Option Setup** par défaut et contentes-vous seulement de cliquer sur **Next**

La génération de code est alors lancée...Une fois terminée, il ne vous reste plus qu'à aller consulter le code généré dans votre répertoire **BookinonsGenerationCode**..

Remarque : vous pouvez relancer une nouvelle génération de code en modifiant cette fois-ci les options de l'**Option Setup**

Remarques pour aller plus loin :

- **Star UML**, comme tout bon AGL, peut également produire des diagrammes de classes à partir du code source orienté objet : c 'est du ***reverse engineering (ou rétro-conception).*** Pour cela, il vous suffira de sélectionner: **Tools → Java → Reverse Engineer ...**

  En quoi la rétro-conception peut-elle être utile ?

  Il arrive (trop souvent!) que des ***méthodes et/ou des attributs aient été oubliés*** lors des phases d'analyse et de conception : projet mal estimé, analyse bâclée ou bien tout simplement ***analyse absente*** (dans le cas de la reprise d'un projet existant par exemple...) L'application est fonctionnelle pour l'utilisateur, mais l'application n'est plus en phase avec l'analyse ... La rétro-conception permet à partir d'un code orienté objet de recréer les classes de l'analyse UML en réinjectant les attributs et méthode omis à la conception ou modifiés durant la phase d'implémentation !

- **Star UML** peut également ***générer de façon automatique des diagrammes de classes grâce aux 26 patrons de conception*** qu'il connaît (3 patterns pour **EJB** et 23 patterns pour **GoF** (Gang of Four – ensemble de patrons de conception réutilisables, minimisation des interactions entre les diverses classes). Pour cela, à partir d'un diagramme actif de type diagramme de classes, il vous suffira de sélectionner: **Tools → Apply Patterns** et de choisir le pattern souhaité.

- **StarUML** peut également génèrer une documentation UML au format Word.