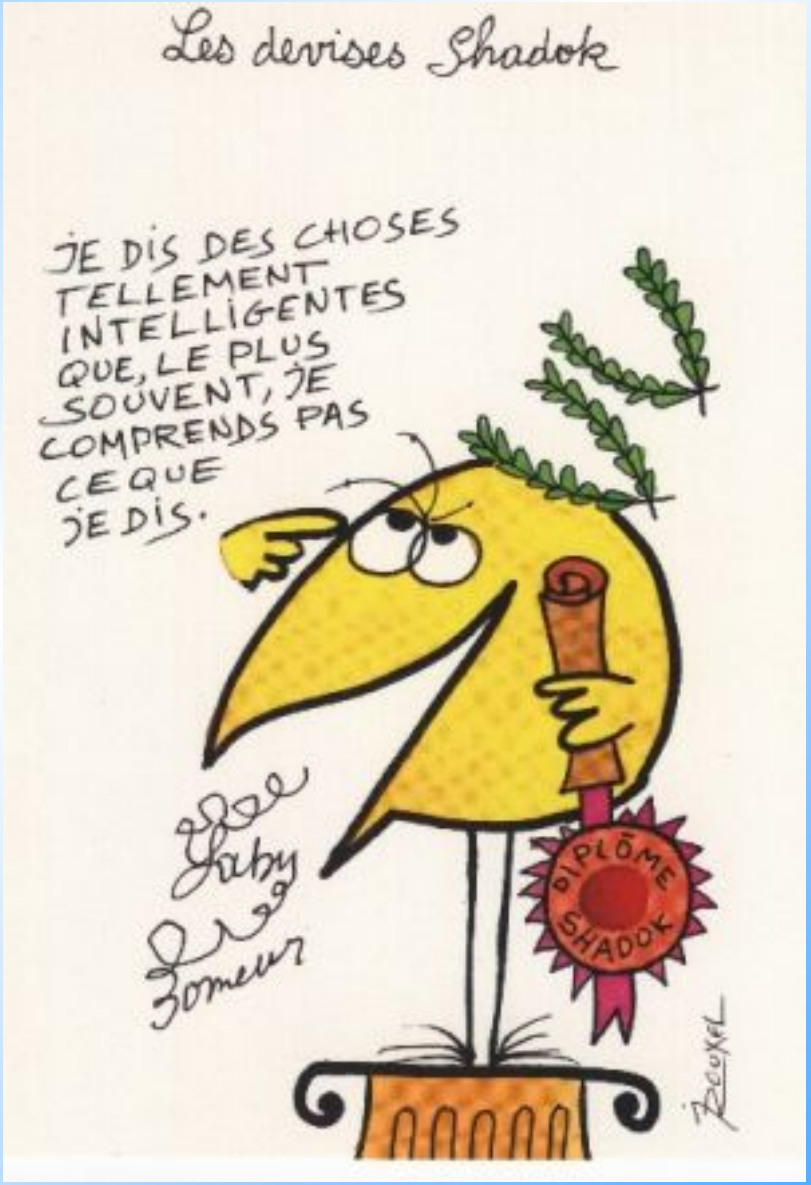
Conception objet préliminaire



Sommaire

- **Introduction**
- Diagrammes d’interaction
- Diagrammes pour le site Web
- Classes de conception préliminaire

Introduction

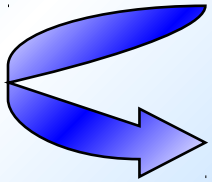
- Les diagrammes de cas d'utilisation montrent des interactions entre des acteurs et les grandes fonctions d'un système.

- Cependant, les interactions ne se limitent pas aux acteurs : les objets au cœur du système interagissent en s'envoyant des messages.

Introduction

- Apprentissage d'**attribution des responsabilités** de comportement des classes d'analyse (*entités* + *dialogues* + *contrôles*).

- Présentation du résultat de cette étude dans des diagrammes d'interactions UML (*séquence ou communication*).

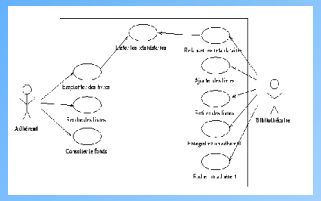


Vue statique complétée

(*sous forme de diagrammes de classes de conception préliminaire*)

indépendante des choix technologiques

Démarche



Cas d'utilisation

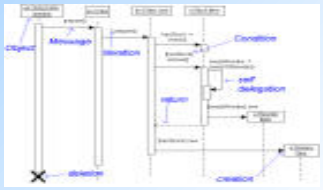
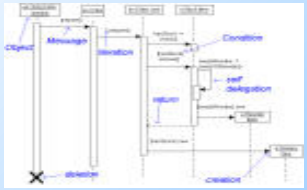


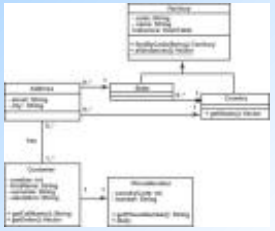
Diagramme de séquence système



Diagrammes d'interaction

Maquette

Diagrammes de classes participantes



Modèle du domaine

Diagrammes d'activités de navigation

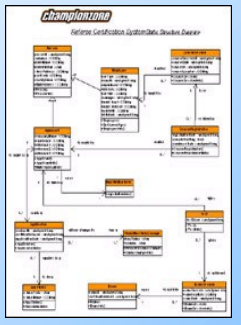


Diagramme de classes de conception

Code

Introduction

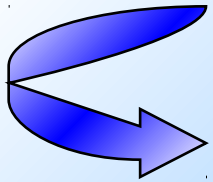
Déjà réalisé :

- Identification d'un certain nombre d'opérations potentielles dans les classes "*dialogues*" et "*contrôles*" :

Premier jet, une base de travail.

Actuellement :

- Conception d'un ensemble de classes faiblement couplées entre elles et fortement cohérentes.



Étude détaillée de la dynamique des objets grâce aux diagrammes d'interaction.

Introduction

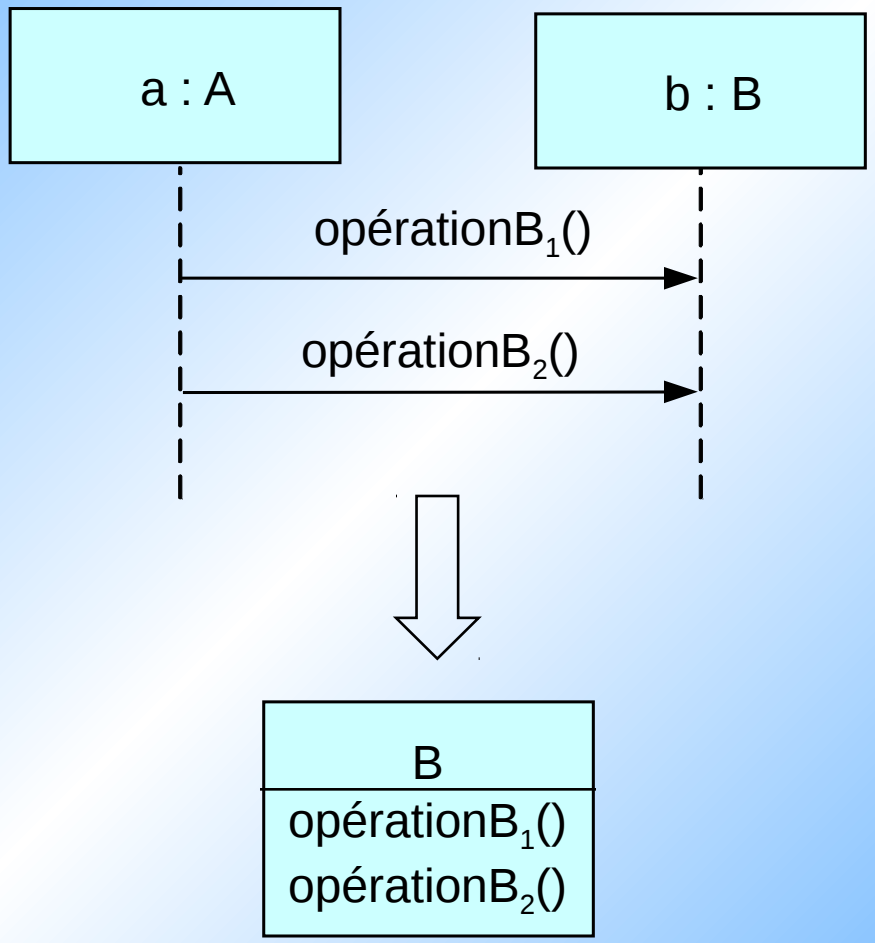
- L'attribution des bonnes responsabilités aux bonnes classes = un des problèmes les plus délicats de la conception orientée objet.

- Pour chaque service ou fonction, il faut décider quelle est la classe qui va le contenir.

- Les diagrammes d'interaction permettent au concepteur de représenter graphiquement ces décisions d'allocation de responsabilités.

- Chaque diagramme va représenter un ensemble d'objets de classes différentes collaborant dans le cadre d'un scénario d'exécution du système.

Introduction



Introduction

- Les objets communiquent en s'envoyant des messages qui invoquent des opérations (*ou méthodes*) sur les objets récepteurs.

- Suivi visuel des interactions dynamiques entre objets, et les traitements réalisés par chacun.

- Par rapport aux diagrammes de séquence système, on va remplacer le système vu comme une boite noire par un ensemble d'objets en collaboration.

- Utilisation des trois types de classes d'analyse, à savoir les dialogues, les contrôles et les entités.

Introduction

- Respect des règles fixées sur les relations entre classes d'analyse

✓ les acteurs ne peuvent interagir (*envoyer des messages*) qu'avec les dialogues.

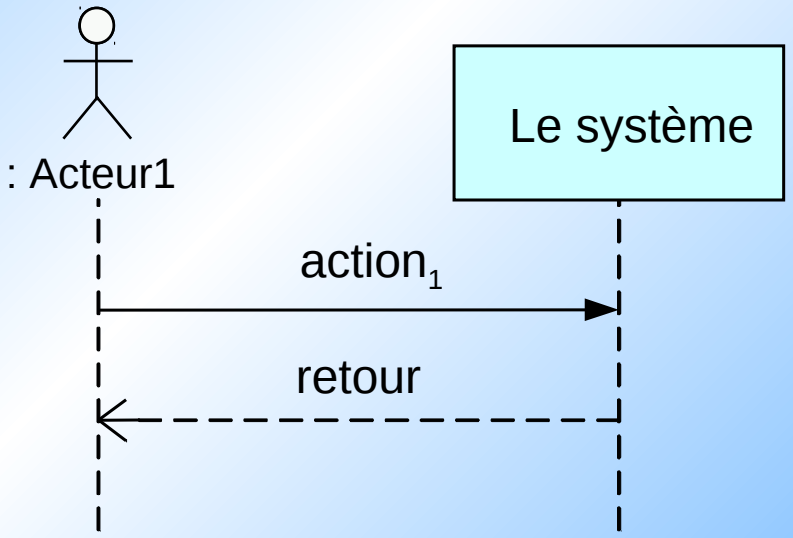
✓ les dialogues peuvent interagir avec les contrôles ou, exceptionnellement, avec d'autres dialogues.

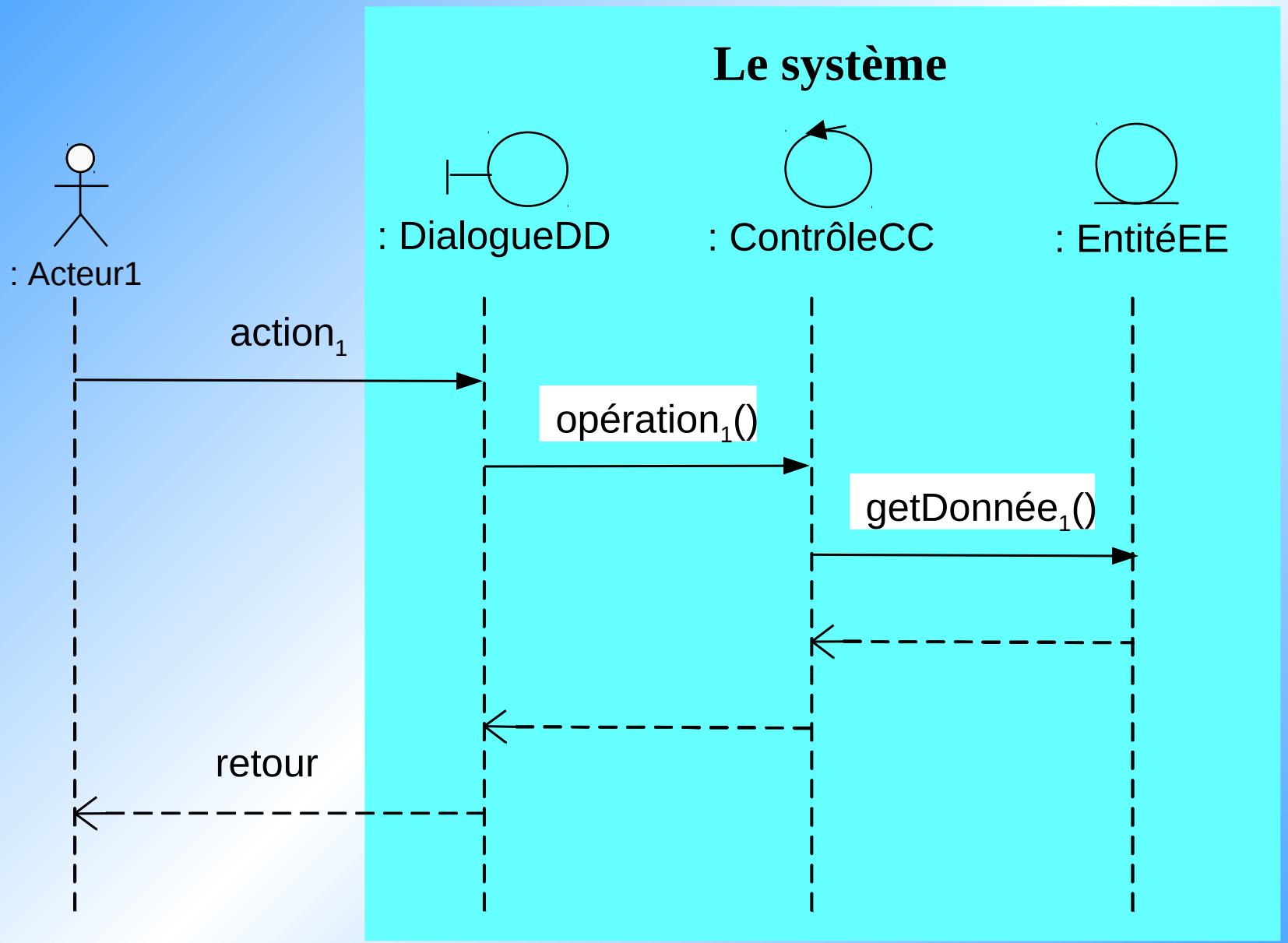
✓ les contrôles peuvent interagir avec les dialogues, les entités, ou d'autres contrôles.

✓ les entités ne peuvent interagir qu'entre elles.

Introduction

- Changement du niveau d'abstraction par rapport au diagramme de séquence système :




Le système
: Acteur1
: DialogueDD
: ContrôleCC
: EntitéEE
action1
opération1()
getDonnée1()
retour

Sommaire

- Introduction
- **Diagrammes d’interaction**
- Diagrammes pour le site Web
- Classes de conception préliminaire

Types de diagrammes d'interaction

- *2* types de diagrammes d'interaction :
  - ✓ les diagrammes de séquence,
  - ✓ les diagrammes de communication (UML 2) ou de collaboration (UML 1).

- Chaque type de diagramme a ses points forts et ses points faibles.

- Si on manque de place en largeur : les diagrammes de communication sont préférables (*ils permettent l'extension verticale des nouveaux objets*).

En revanche, la lecture des séquences de messages y est plus difficile.

diagrammes de séquence

- Les **diagrammes de séquence** représentent les interactions dans un format où chaque nouvel objet est ajouté en haut à droite.

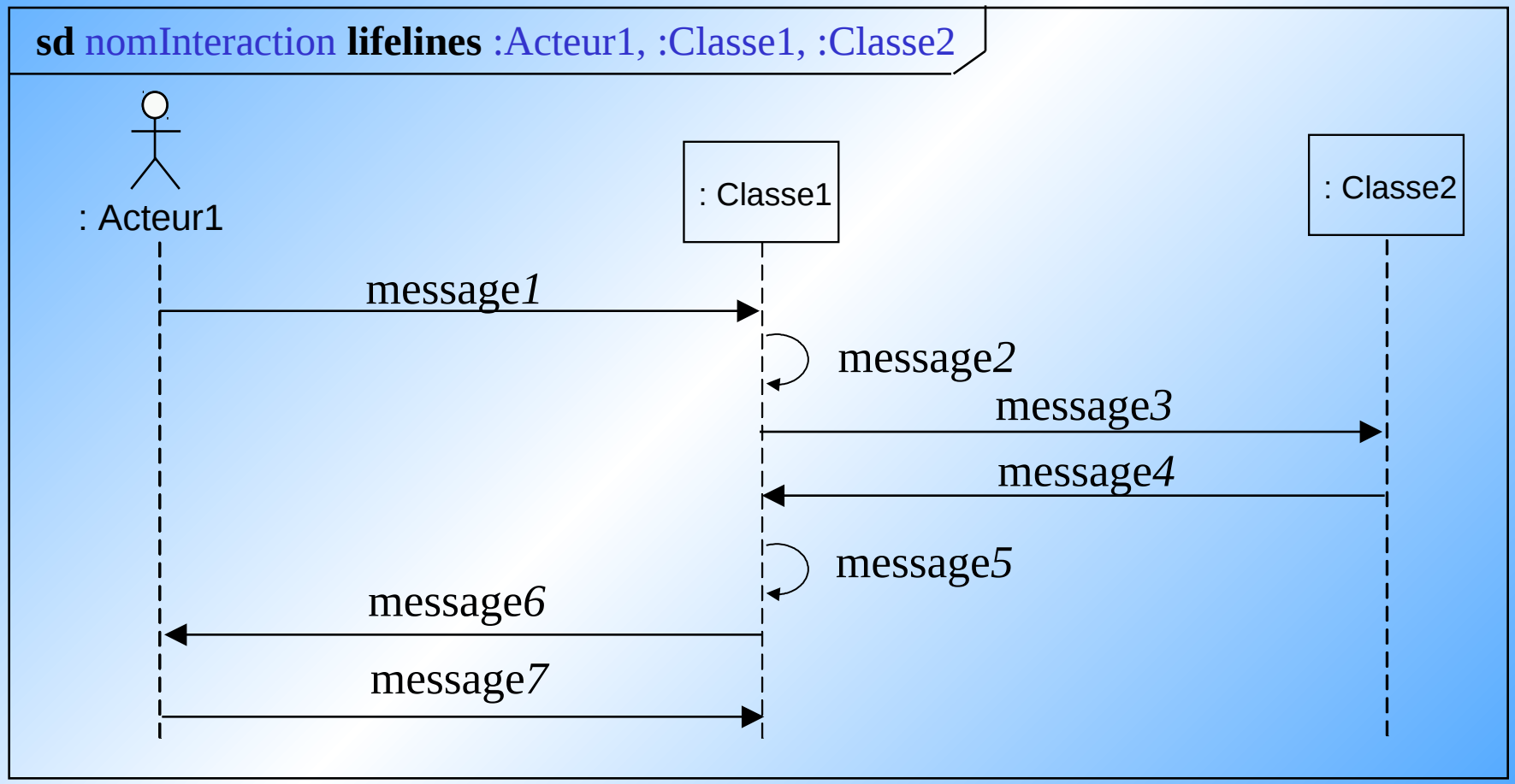
- On représente la ligne de vie de chaque objet par un trait pointillé vertical.

- Chaque ligne de vie sert de point de départ ou d'arrivée à des messages représentés eux-mêmes par des flèches horizontales.

- Par convention, le temps coule de haut en bas.

- Il indique ainsi visuellement la séquence relative des envois et réceptions de messages, d'où la dénomination : diagramme de séquence.

diagrammes de séquence



diagrammes de séquence

Syntaxe des messages

[<attribut> =] <message> [: <valeurRetour>]

optionnel *optionnel*

nombreLivres = chercher(" Tintin ") : 42

diagrammes de communication

- Les **diagrammes de communication** illustrent les interactions entre objets sous forme de graphes ou de réseaux.

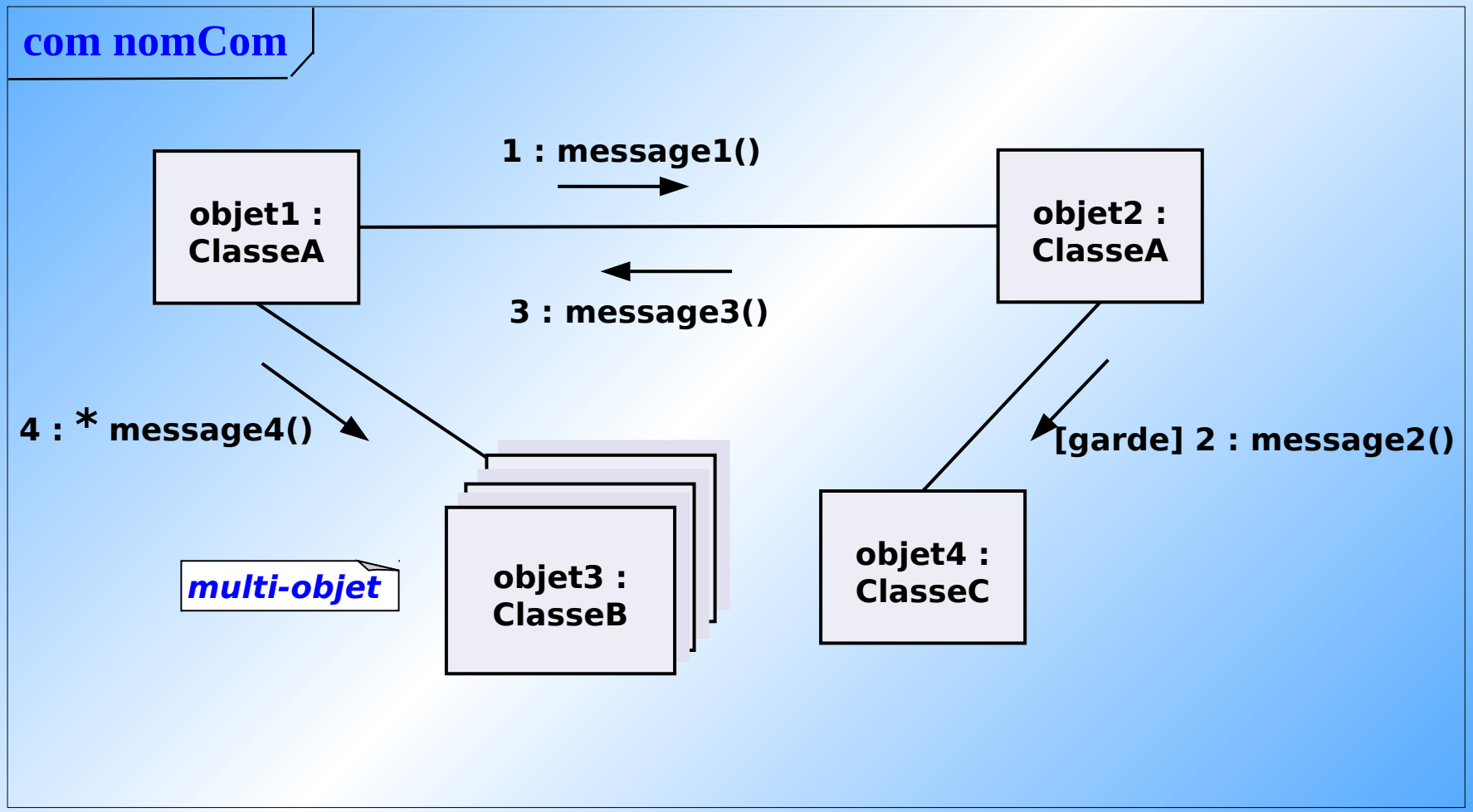
- Les objets peuvent être placés en tout point du diagramme.

- Ils sont connectés par des liens qui indiquent qu'une forme de navigation et de visibilité entre ces objets est possible.

- Tout message entre objets est représenté par une expression et une petite flèche indiquant son sens de circulation.

- Chaque lien permet le trafic de plusieurs messages et chaque message est assorti d'un numéro d'ordre.

diagrammes de communication



diagrammes de communication

Syntaxe des messages

[<cond>] <seq> [<iter>] : [<var>] <message> [<par>]

optionnel *optionnel* *optionnel* *optionnel*

- <cond> : condition sous forme d'expression booléenne entre crochets.

- <seq> : numéro de séquence du message.
  Les messages 1.2a et 1.2b sont envoyés en même temps.

diagrammes de communication

[<cond>] <seq> [<iter>] : [<var>] <message> [<par>]

- <iter> : spécifie, en langage naturel et entre crochets, l'envoi séquentiel (* <iter>) ou en parallèle (* || <iter>) de plusieurs messages.
La spécification <iter> peut être ignorée.

- <var> : valeur de retour du message, qui sera, par exemple, transmise en paramètre à un autre message.

- <message> : nom du message.

- <par> : paramètres (*optionnels*) du message.

diagrammes de communication

Exemple de messages

[<cond>] <seq> [<iter>] : [<var>] <message> [<par>]

- 1 : mes()
- 2.1 : mes(args)
- 3.2b : var = mes()
- [a > b] 3 : mes()
- 2.1 *[i : =0 .. 10] : mes()
- 4 *|| [pour toutes les portes] : fermer()
- [heure = heureDépart] 1.1a * || [i:= 0 .. 10] : ok[i] = fermer()

diagrammes d'interaction

- Les diagrammes d'interaction montrent des instances (*des occurrences*) et non pas des classes :

✓ à chaque élément UML (*classe*, *acteur*, *etc*.) correspond une instance qui utilise le même symbole graphique que le type.

✓ on peut identifier l'instance par un nom univoque (*exemple objet1:ClasseA*). En l'absence d'un tel nom, on fait précéder le nom de classe par le symbole ″:″ .

Types de messages

- Envoi d’un signal
- Invocation d’une opération
- Création ou destruction d’une instance

Envoi d'un signal

- L'envoi d'un signal déclenche une réaction chez le récepteur, de façon asynchrone.

- L'émetteur du signal ne reste pas bloqué le temps que le signal parvienne au récepteur et il ne sait pas quand, ni même si le message sera traité par le destinataire.

Invocation d'une opération

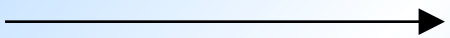
- Type de message le plus utilisé en programmation objet.
- L'invocation peut être synchrone (*l'émetteur reste bloqué le temps que dure l'invocation de l'opération*) ou asynchrone.
- La plupart des invocations sont synchrones.

Représentation des messages

- UML fait la différence entre un message synchrone et asynchrone.

ATTENTION NOTATION

flèche pleine



message synchrone

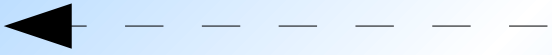
flèche évidée

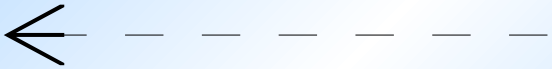
message asynchrone

diagrammes d'interaction

Représentation des messages

message réponse



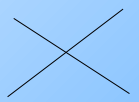


message de création

<<create>>

message de suppression

rien d'indiqué



Types de messages

- **message simple** : message dont on ne spécifie aucune caractéristique d'envoi ou de réception particulière.

- **message minuté** (*timeout*) : bloque l'expéditeur pendant un temps donné (*qui peut être spécifié dans une contrainte*), en attendant la prise en compte du message par le récepteur. L'expéditeur est libéré si la prise en compte n'a pas eu lieu pendant le délai spécifié.

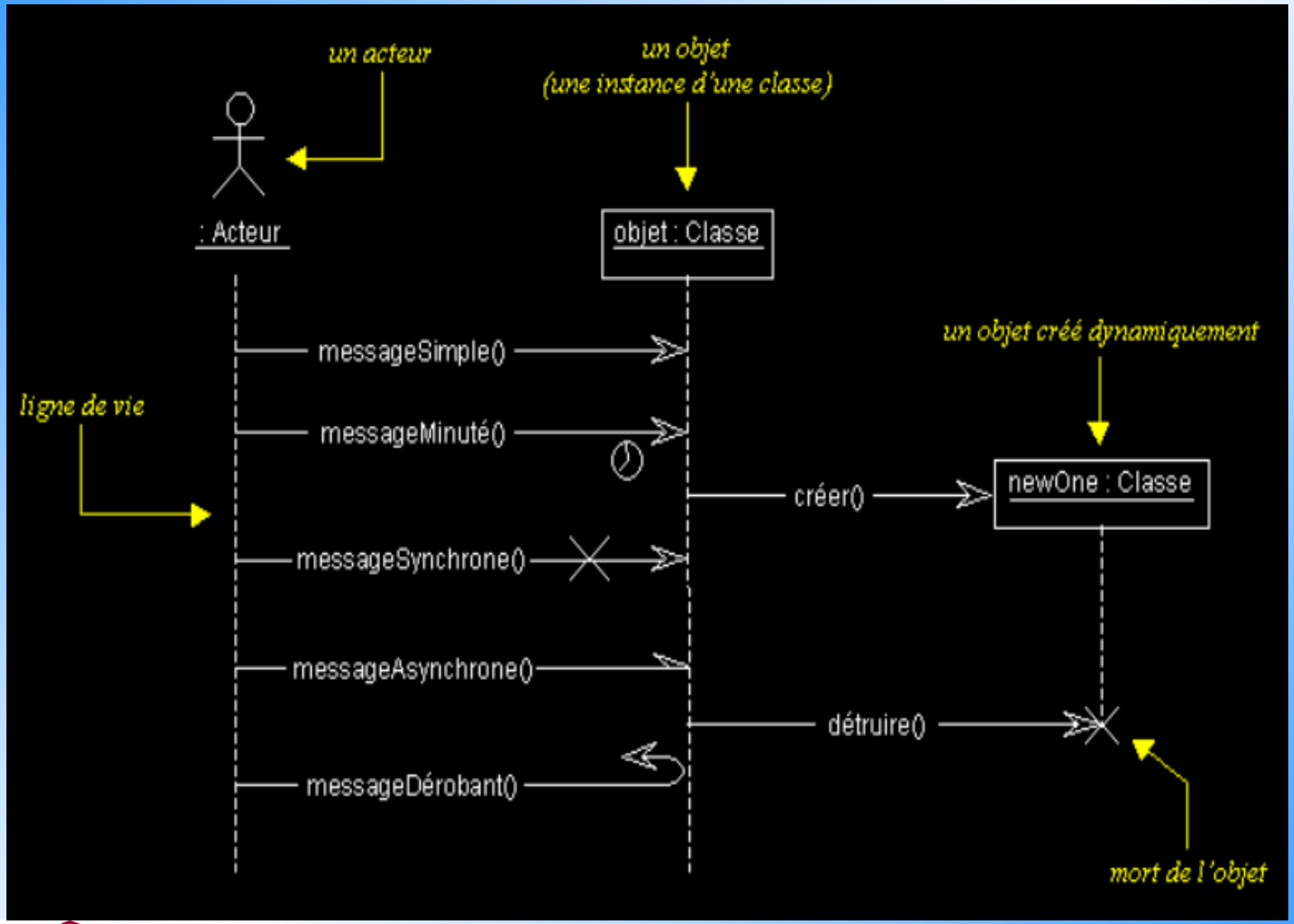
Types de messages

- **message synchrone** : bloque l'expéditeur jusqu'à la prise en compte du message par le destinataire. Le flot de contrôle passe de l'émetteur au récepteur (*l'émetteur devient passif et le récepteur actif*) à la prise en compte du message.

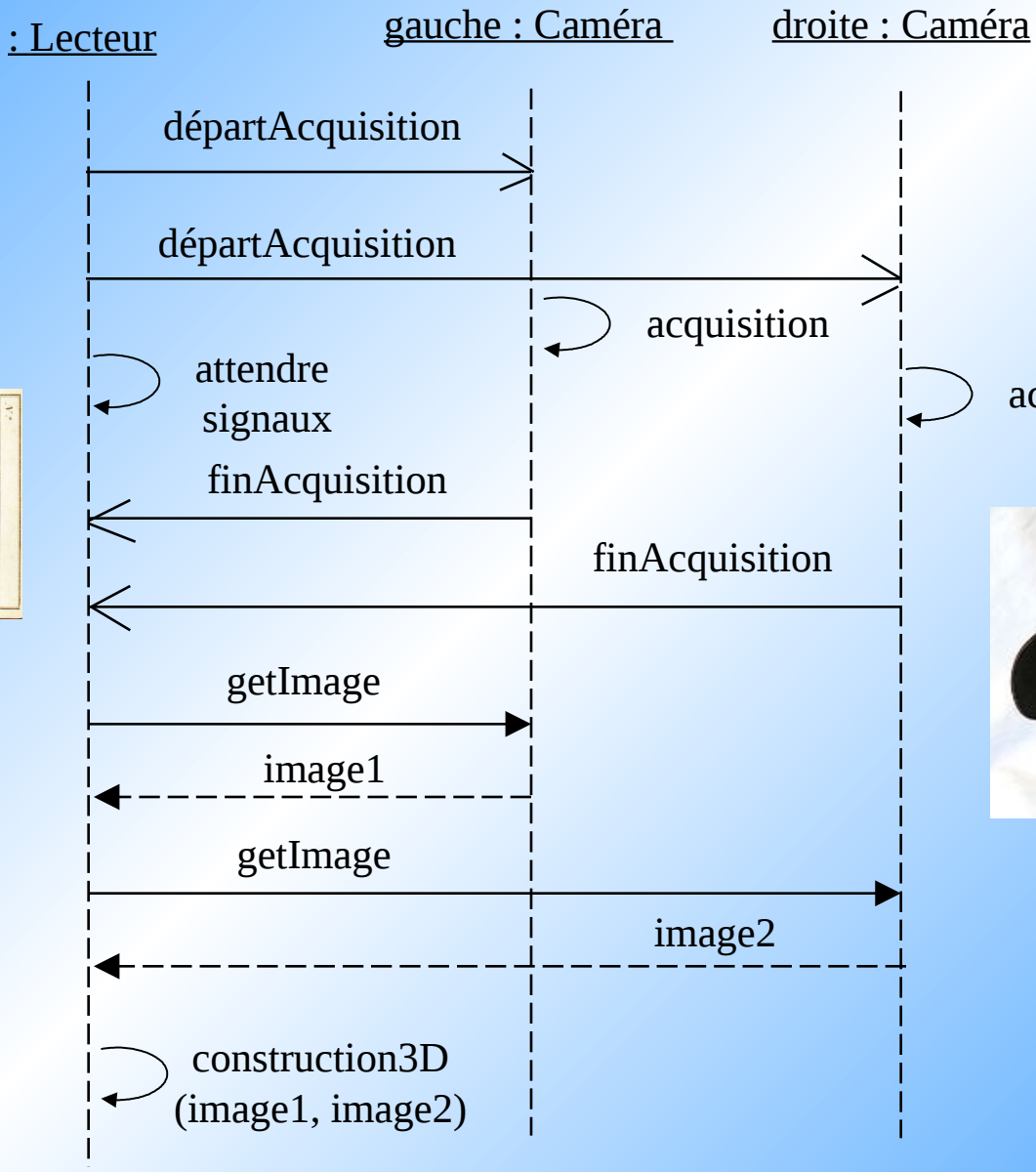
- **message asynchrone** : n'interrompt pas l'exécution de l'expéditeur. Le message envoyé peut être pris en compte par le récepteur à tout moment ou ignoré (*jamais traité*).

Types de messages

- **message dérobant** : n'interrompt pas l'exécution de l'expéditeur et ne déclenche une opération chez le récepteur que s'il s'est préalablement mis en attente de ce message.


un acteur
un objet
(une instance d'une classe)
: Acteur
objet : Classe
un objet créé dynamiquement
ligne de vie
messageSimple()
messageMinuté()
créer()
newOne : Classe
messageSynchrone()
messageAsynchrone()
détruire()
messageDérobant()
mort de l'objet

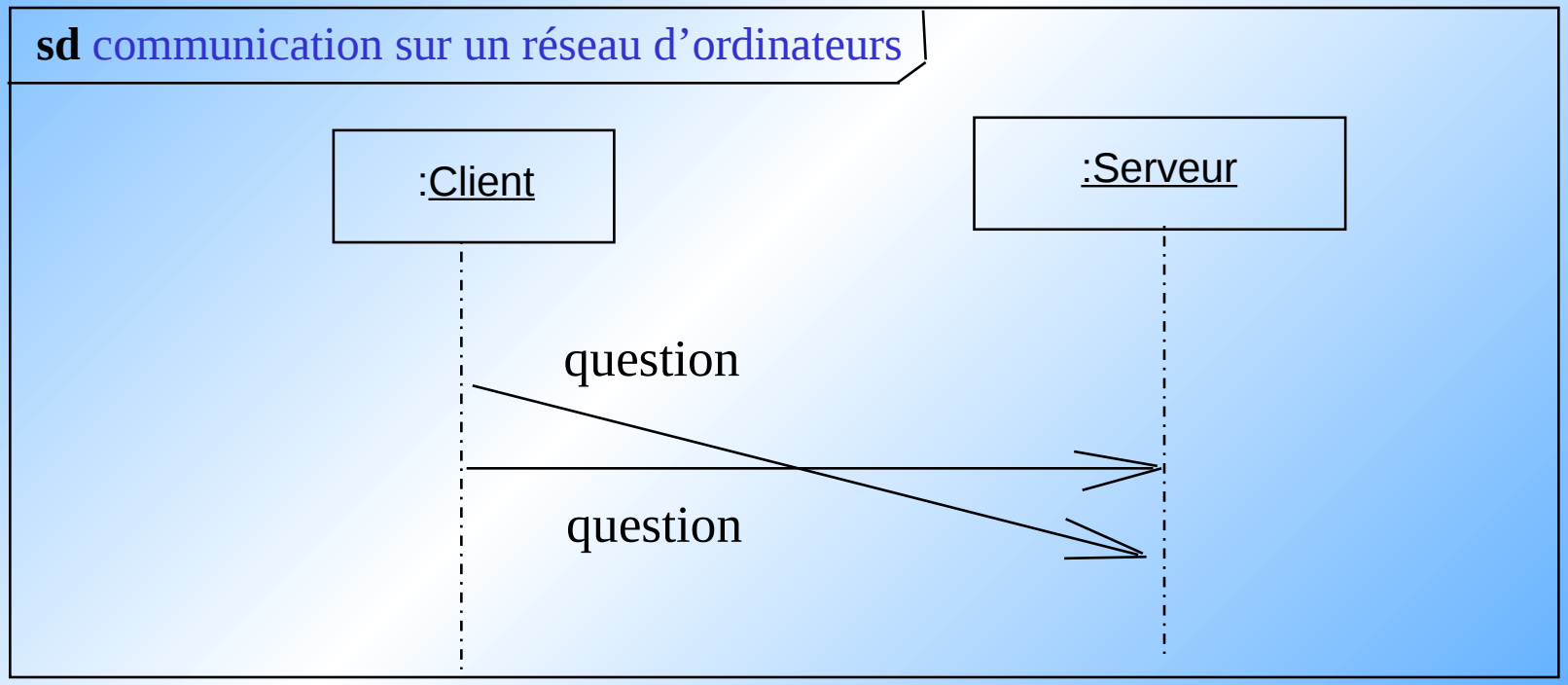
Lecture stéréoscopique



diagrammes d'interaction

Message asynchrone

- Les messages peuvent être reçus dans un ordre différent de l'ordre d'envoi.



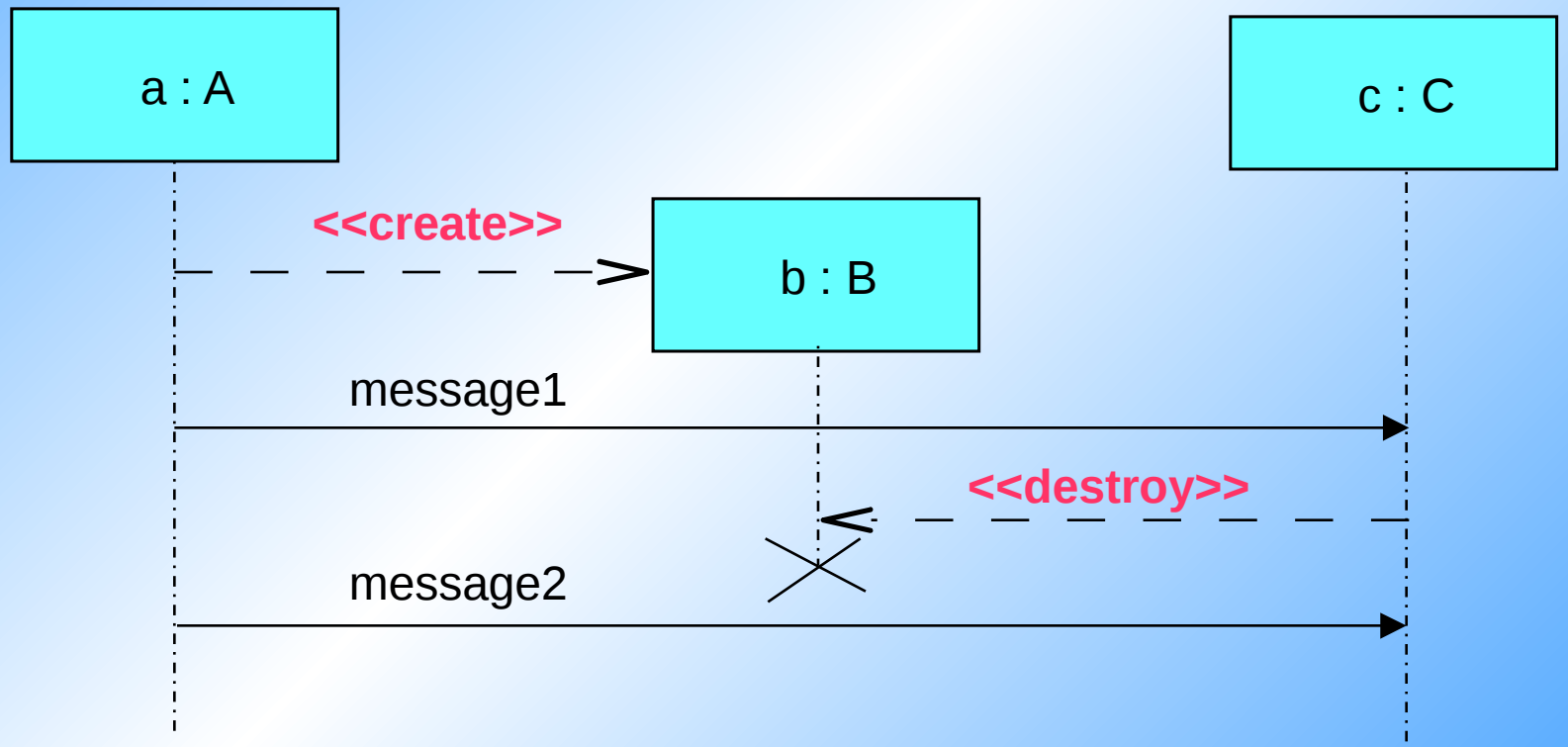
Certains protocoles de communication ne garantissent pas l'ordre d'arrivée des messages

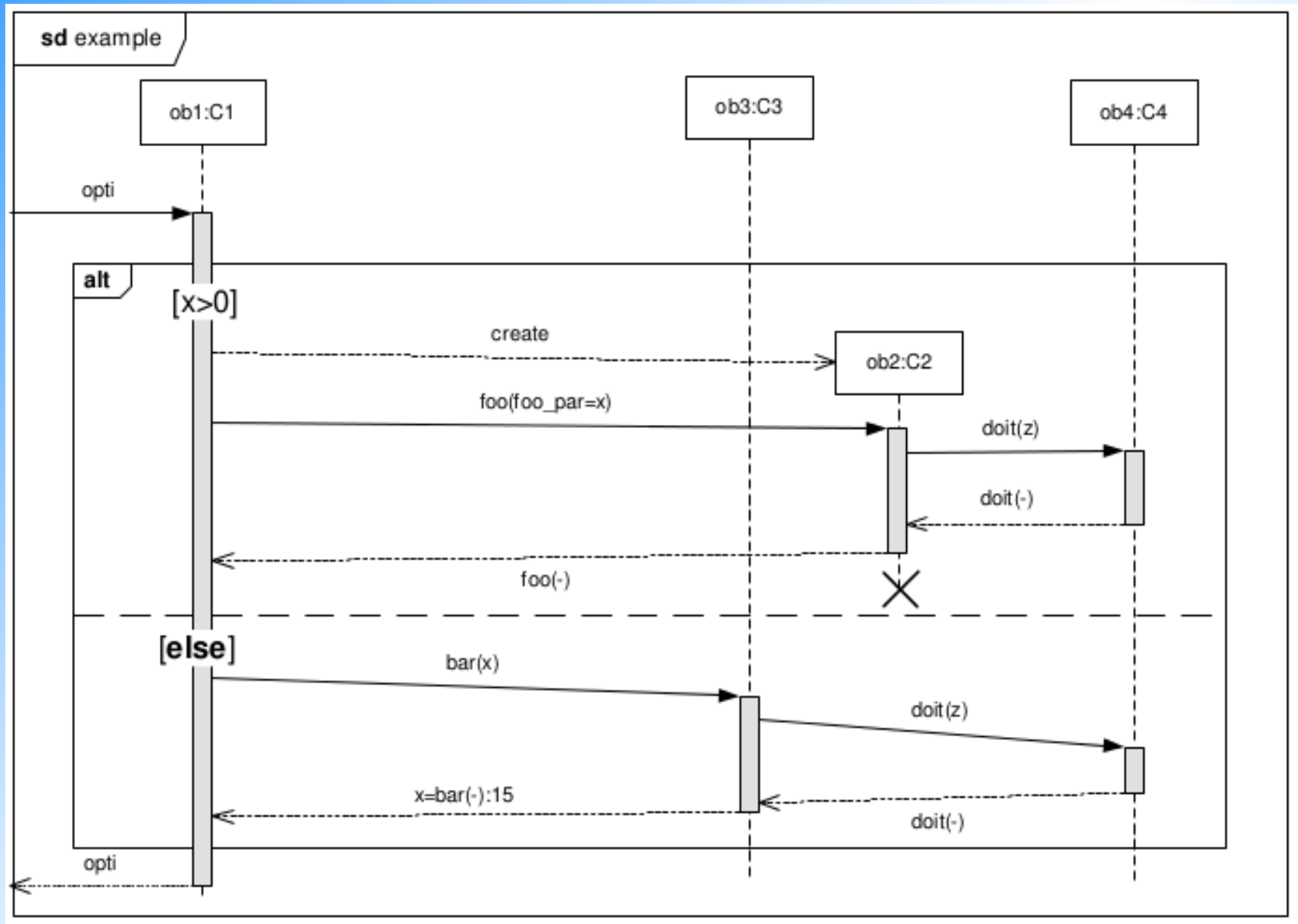
Création ou destruction d'une instance

- Création d'une instance : message standardisé *"create"*.
- Le message *"create"* peut comprendre des paramètres d'initialisation.
- Destruction d'un instance : message standardisé *"destroy"*.
- La création et la destruction d'instances ont des représentations particulières sur le diagramme de séquence, mais pas sur le diagramme de communication.

diagrammes d'interaction

- L'objet *b* est créé et détruit durant le scénario, contrairement aux objets *a* et *c* qui préexistent et survivent au scénario concerné.




sd example
ob1:C1
ob3:C3
ob4:C4
opti
alt
[x>0]
create
ob2:C2
foo(foo_par=x)
doit(z)
doit(-)
foo(-)
[else]
bar(x)
doit(z)
x=bar(-):15
doit(-)
opti

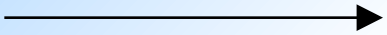
Remarque

- Certains langages objet comme Java et C# ont un "*garbage collector*" qui détruit automatiquement les objets qui ne sont plus utilisés.

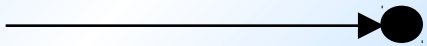
- Ce n’est pas le cas en C++ où les objets doivent être détruits explicitement.

Types de messages

- **Message complet** : les événements d'envoi et de réception sont connus.



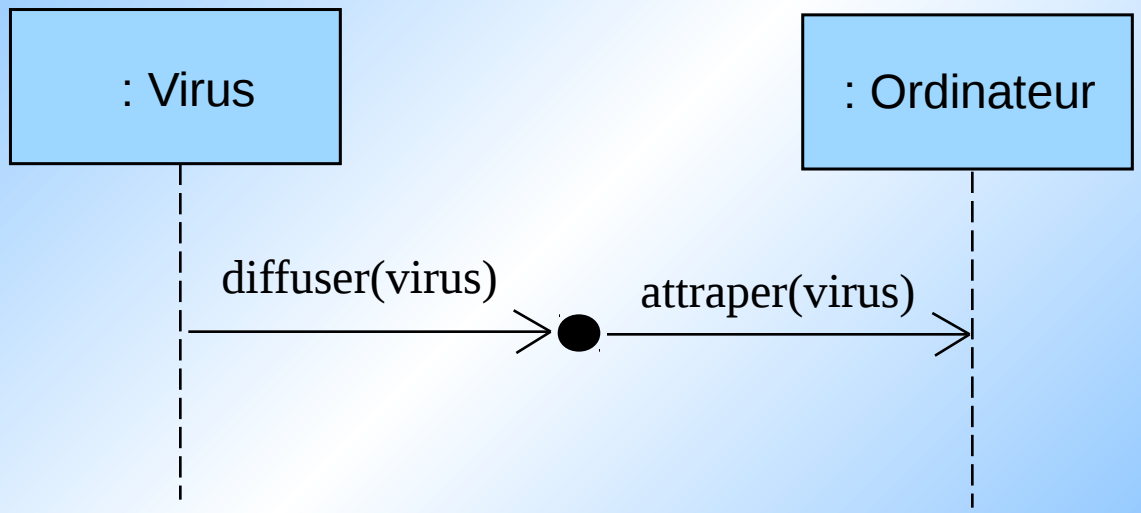
- **Message perdu** : événement d'envoi connu mais pas l'événement de réception.



- **Message trouvé** : événement de réception connu mais pas l'événement d'émission.

Types de messages

Exemple de message perdu et trouvé



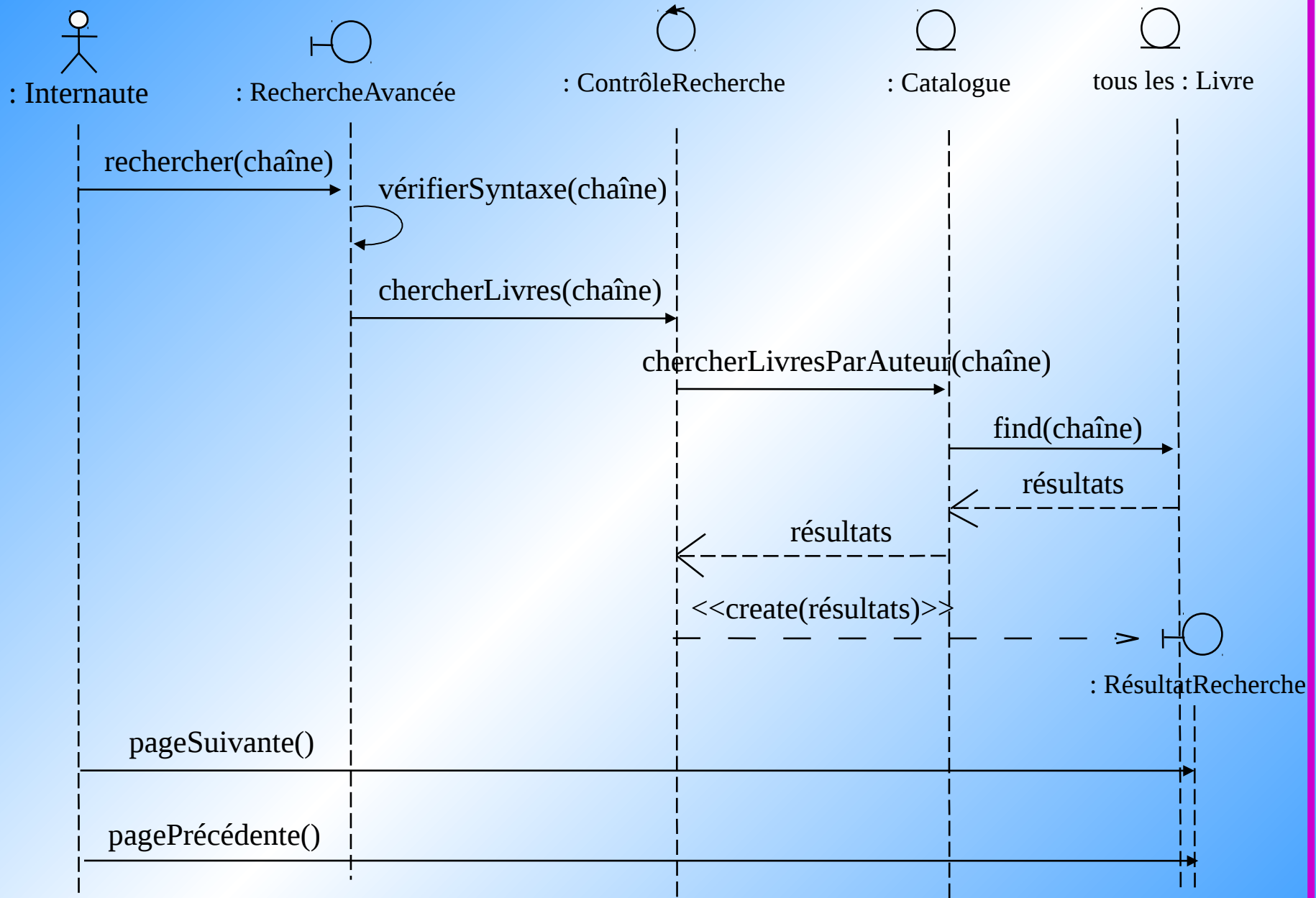
Sommaire

- Introduction
- Diagrammes d’interaction
- **Diagrammes pour le site Web**
- Classes de conception préliminaire

diagrammes d’interaction site web

Faire un diagramme de séquence
Rechercher des ouvrages

Scénario nominal de recherche avancée par nom d’auteur.


: Internaute
: RechercheAvancée
: ContrôleRecherche
: Catalogue
tous les : Livre
rechercher(chaîne)
vérifierSyntaxe(chaîne)
chercherLivres(chaîne)
chercherLivresParAuteur(chaîne)
find(chaîne)
résultats
résultats
<<create(résultats)>>
: RésultatRecherche
pageSuivante()
pagePrécédente()

Solution

- Sur la page d'accueil du site, l'internaute choisit le lien vers la page de recherche avancée.

- Il saisit une chaîne de recherche (*par exemple ici un nom d'auteur*).

- La vérification syntaxique de la phrase de recherche est de la responsabilité de la classe "*dialogue*" elle-même et non du "*contrôle*" associé qui n'est invoqué que dans le cas favorable où il n'y a pas d'erreur de syntaxe.

- Le "*contrôle*" délègue ensuite à une "*entité*" la recherche proprement dite.

Solution

- Quelle est la classe la mieux placée pour effectuer une recherche parmi l'ensemble des ouvrages du catalogue ?
C'est le catalogue lui-même puisqu'il contient tous les livres.

- Le catalogue construit alors une collection dynamique de livres correspondant à la recherche, qu'il renvoie au contrôle.

- Celui-ci initialise le "*dialogue*" chargé du résultat, en lui passant la collection en paramètre.

- L'internaute peut ensuite naviguer dans les différentes pages du résultat.

Solution

- Un objet peut s'envoyer un message à lui-même via un lien.

C'est le cas du dialogue *RechercheAvancée* qui s'envoie le message *vérifierSyntaxeRecherche*.

- Il s'agit d'un traitement interne à l'objet, mais important car son résultat influe sur la suite du scénario. On a donc envie de le représenter, même s'il ne s'agit pas d'une interaction entre plusieurs objets.

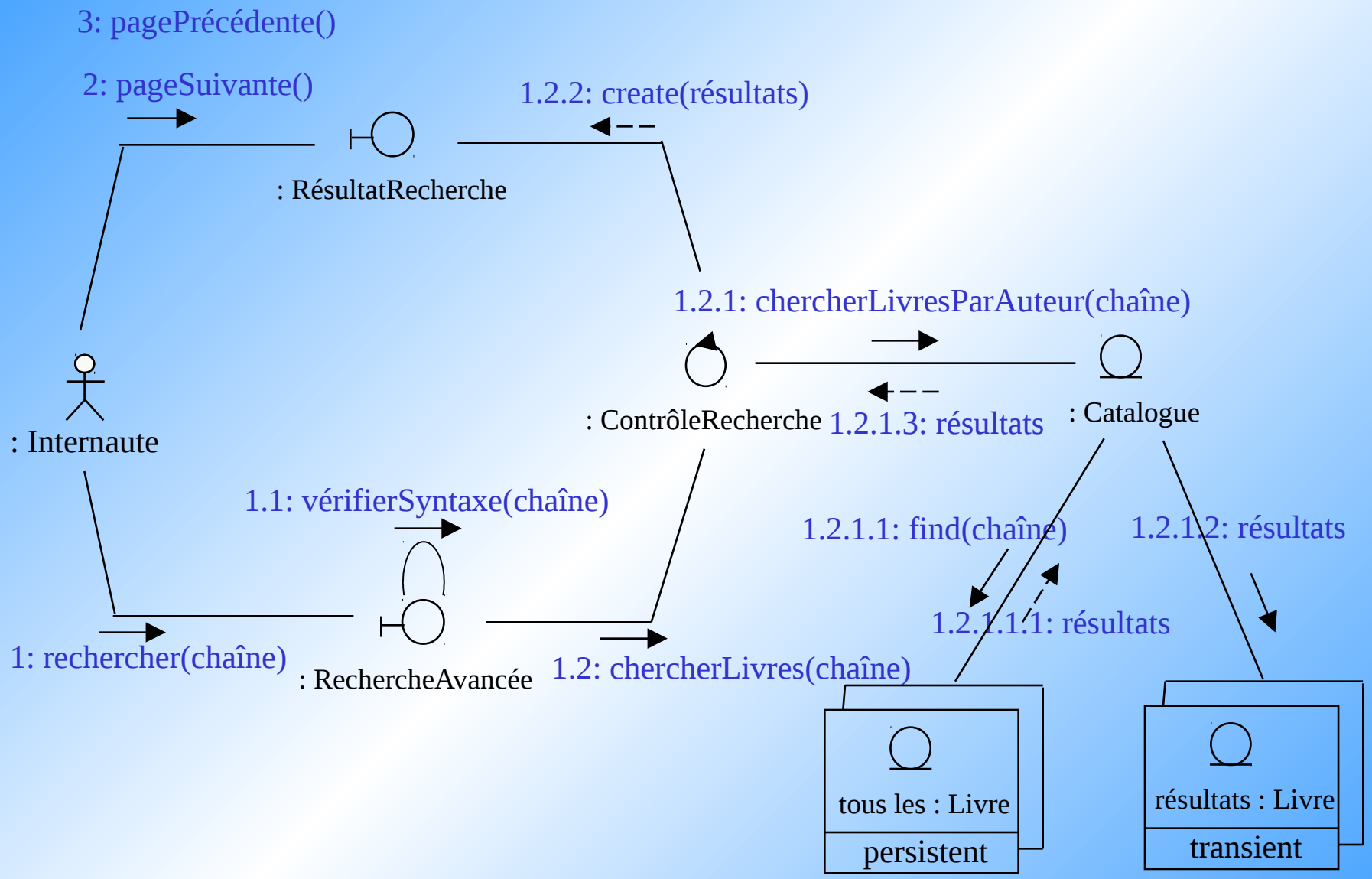
- Ce traitement interne se traduit généralement par une méthode privée (*private*) en Java, C++ ou C#, alors que la réception d'un message venant d'un autre objet correspond forcément à l'invocation d'une méthode publique (*public*).

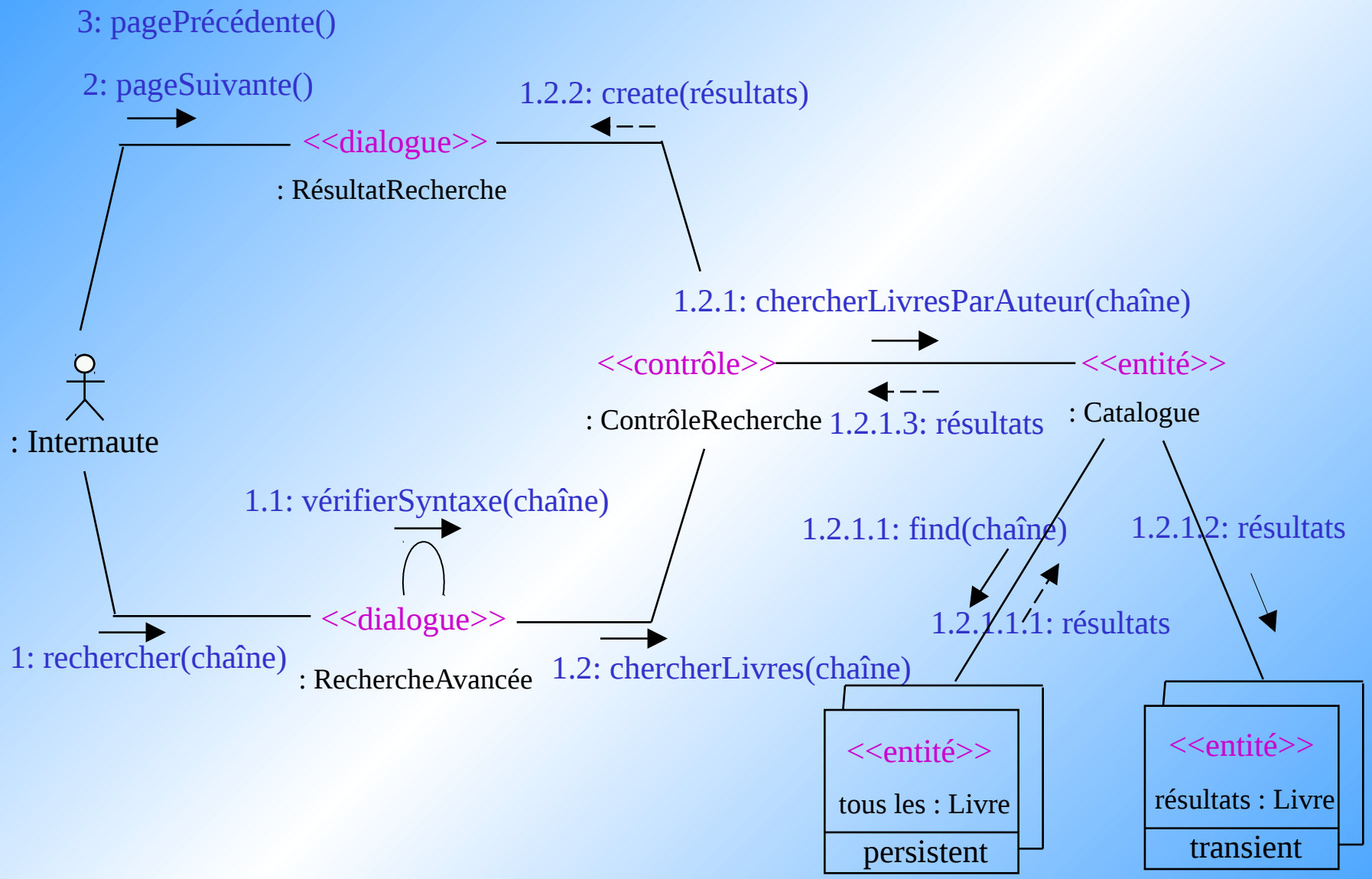
diagrammes d'interaction site web

Faire un diagramme de communication

Rechercher des ouvrages

Scénario nominal de recherche avancée par nom d'auteur.


3: pagePrécédente()
2: pageSuivante()
1.2.2: create(résultats)
: RésultatRecherche
1.2.1: chercherLivresParAuteur(chaîne)
: Internaute
: ContrôleRecherche
1.2.1.3: résultats
: Catalogue
1.1: vérifierSyntaxe(chaîne)
1.2.1.1: find(chaîne)
1.2.1.2: résultats
1.2.1.1.1: résultats
1: rechercher(chaîne)
: RechercheAvancée
1.2: chercherLivres(chaîne)
tous les : Livre
persistent
résultats : Livre
transient


3: pagePrécédente()
2: pageSuivante()
1.2.2: create(résultats)
<<dialogue>>
: RésultatRecherche
1.2.1: chercherLivresParAuteur(chaîne)
<<contrôle>>
<<entité>>
: ContrôleRecherche
1.2.1.3: résultats
: Catalogue
: Internaute
1.1: vérifierSyntaxe(chaîne)
1.2.1.1: find(chaîne)
1.2.1.2: résultats
1.2.1.1.1: résultats
<<dialogue>>
1: rechercher(chaîne)
: RechercheAvancée
1.2: chercherLivres(chaîne)
<<entité>>
tous les : Livre
persistent
<<entité>>
résultats : Livre
transient

Solution

- Pour indiquer que le catalogue contient une collection de livres, on utilise un multi-objet.

- Le multi-objet est une construction UML qui représente en un seul symbole plusieurs objets de la même classe. Cela permet de ne pas ajouter trop tôt de classes de conception détaillée liées à la technique d'implémentation *(comme la classe Vector de la STL C++ ou ArrayList en Java, etc.*).
Un multi-objet peut également représenter l'abstraction entière d'une connexion à une base de données.

- On utilisera des noms de messages génériques comme *find()* ou *add()* sur les multi-objets.

Solution

- On a aussi fait figurer sur chacun des deux multi-objets une indication de persistance.

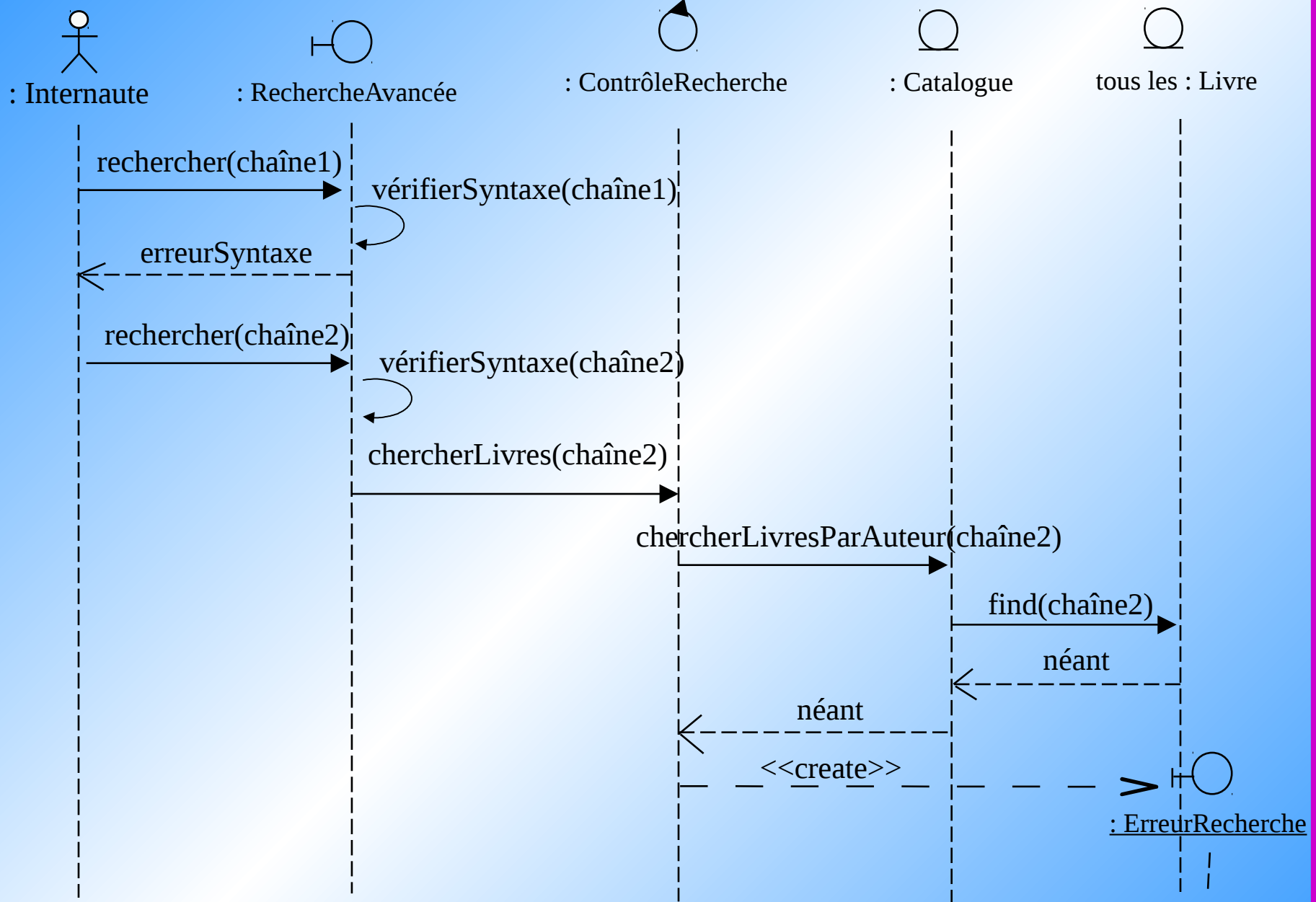
- Cela permet de bien distinguer l'ensemble des livres du catalogue (*persistent*) qui seront stockés grâce à un mécanisme de persistance (*en général*, *une base de données*), alors que le résultat de la recherche est une collection dynamique de livres (*transient*) affichée par la page de résultat de recherche mais non sauvegardée au-delà de la session de l'internaute.

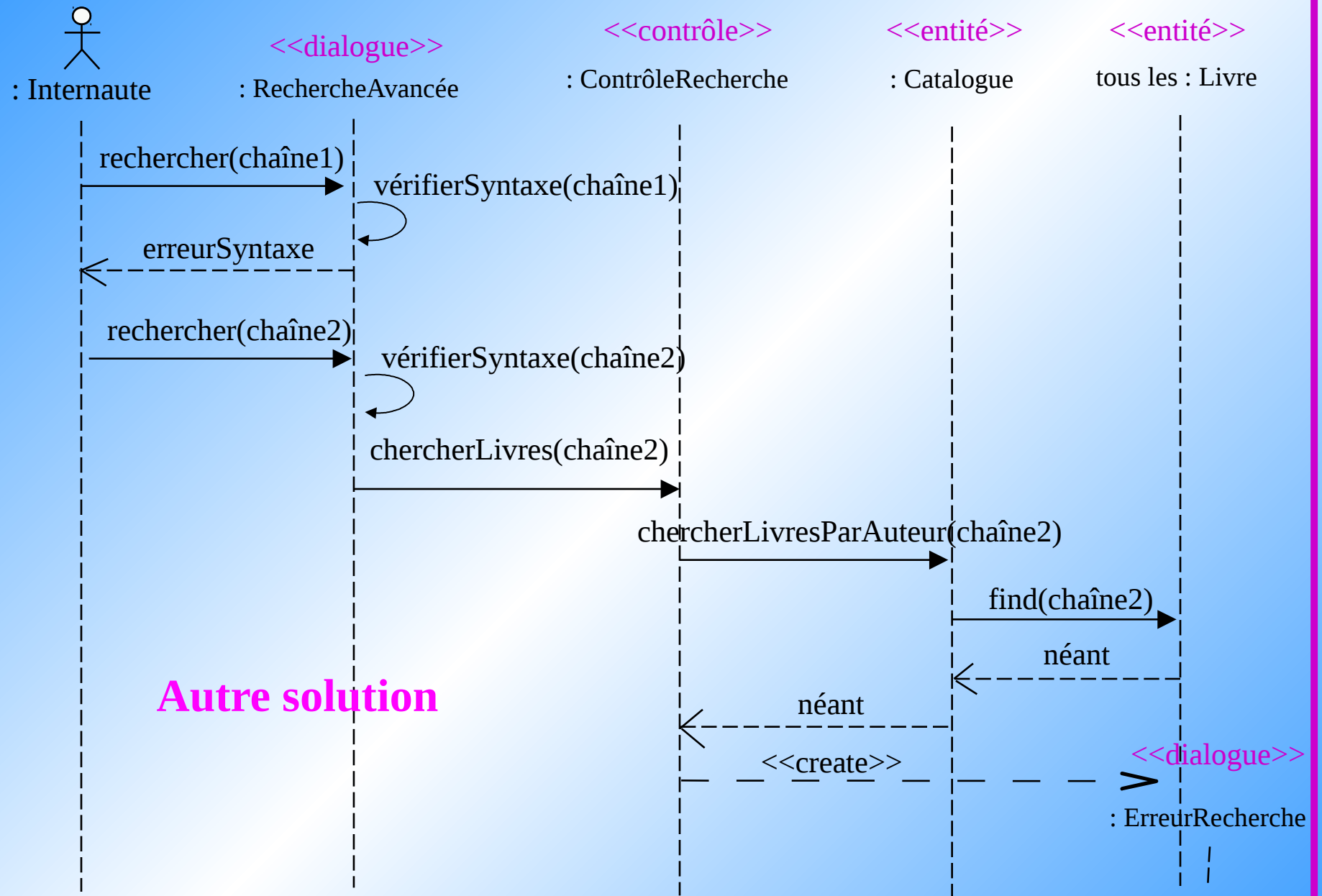
diagrammes d'interaction site web

Faire un diagramme de séquence
Rechercher des ouvrages

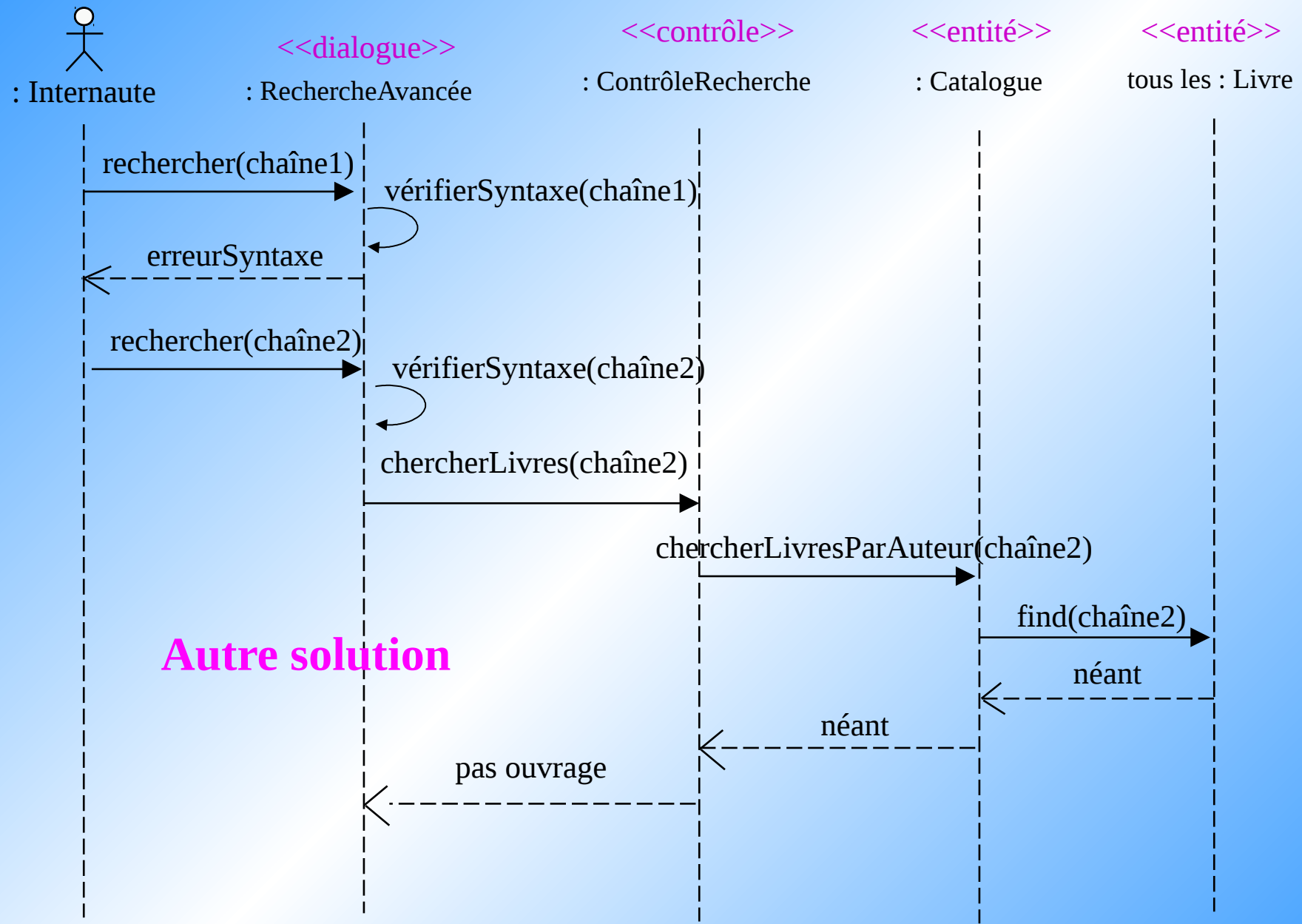
Échec de la recherche

1. Erreur de syntaxe
2. Pas d'ouvrage correspondant à la requête

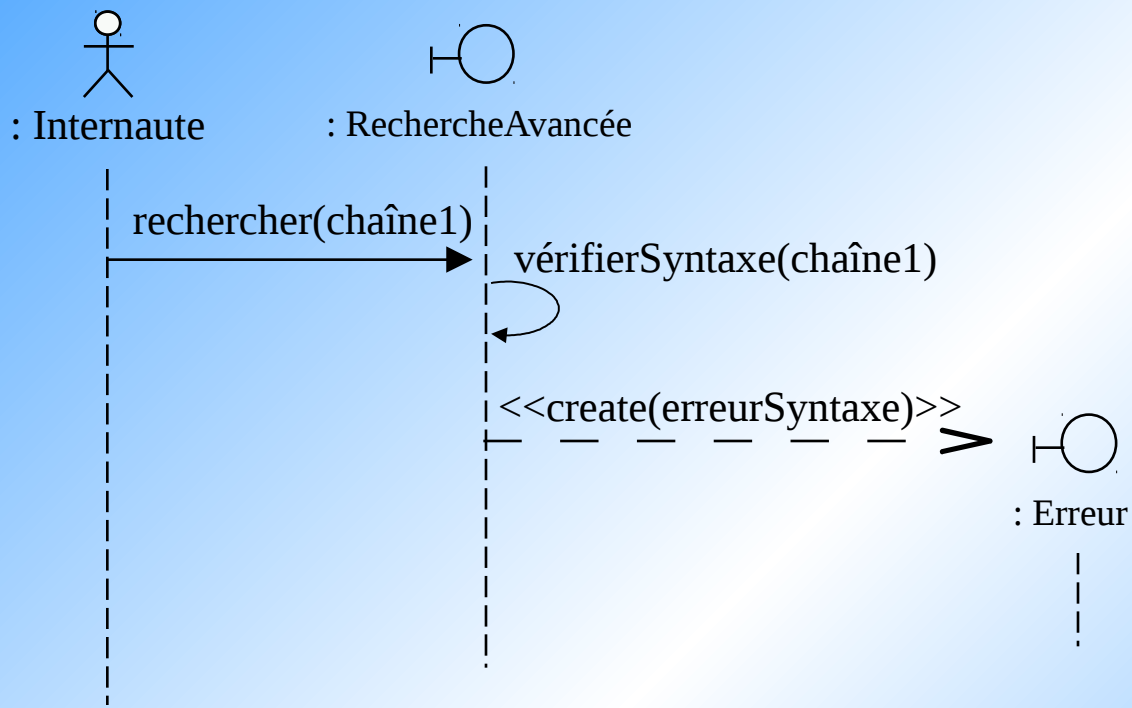

: Internaute
: RechercheAvancée
: ContrôleRecherche
: Catalogue
tous les : Livre
rechercher(chaîne1)
vérifierSyntaxe(chaîne1)
erreurSyntaxe
rechercher(chaîne2)
vérifierSyntaxe(chaîne2)
chercherLivres(chaîne2)
chercherLivresParAuteur(chaîne2)
find(chaîne2)
néant
néant
<<create>>
: ErreurRecherche



Autre solution



Autre solution



Autre solution

Solution

- Erreur de syntaxe dans la recherche (*phrase 1 erronée*) détectée directement par le dialogue sans intervention du contrôle.

- Aucun ouvrage trouvé suite à la recherche dans le catalogue (*la phrase 2 est syntaxiquement correcte*).

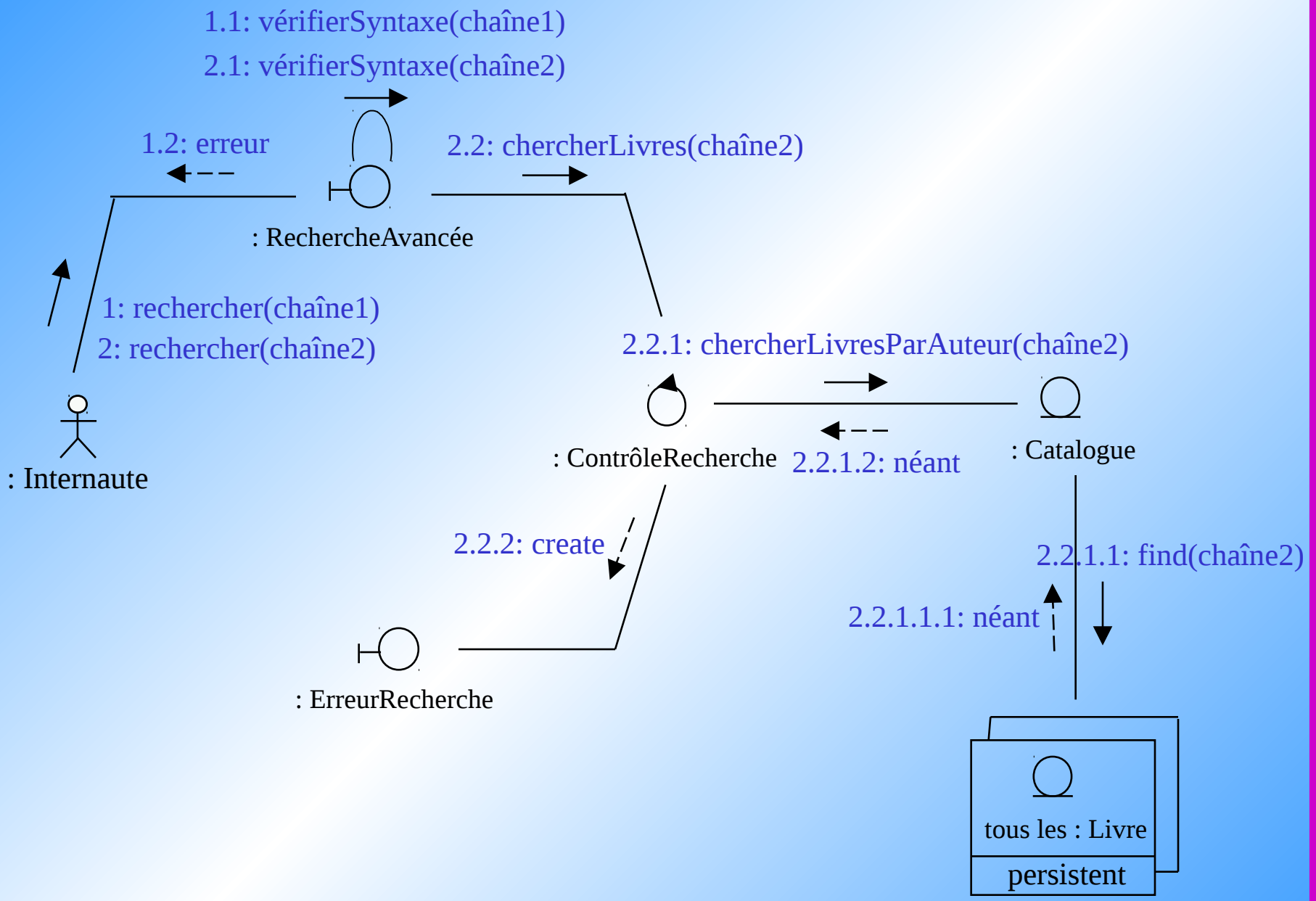
diagrammes d'interaction site web

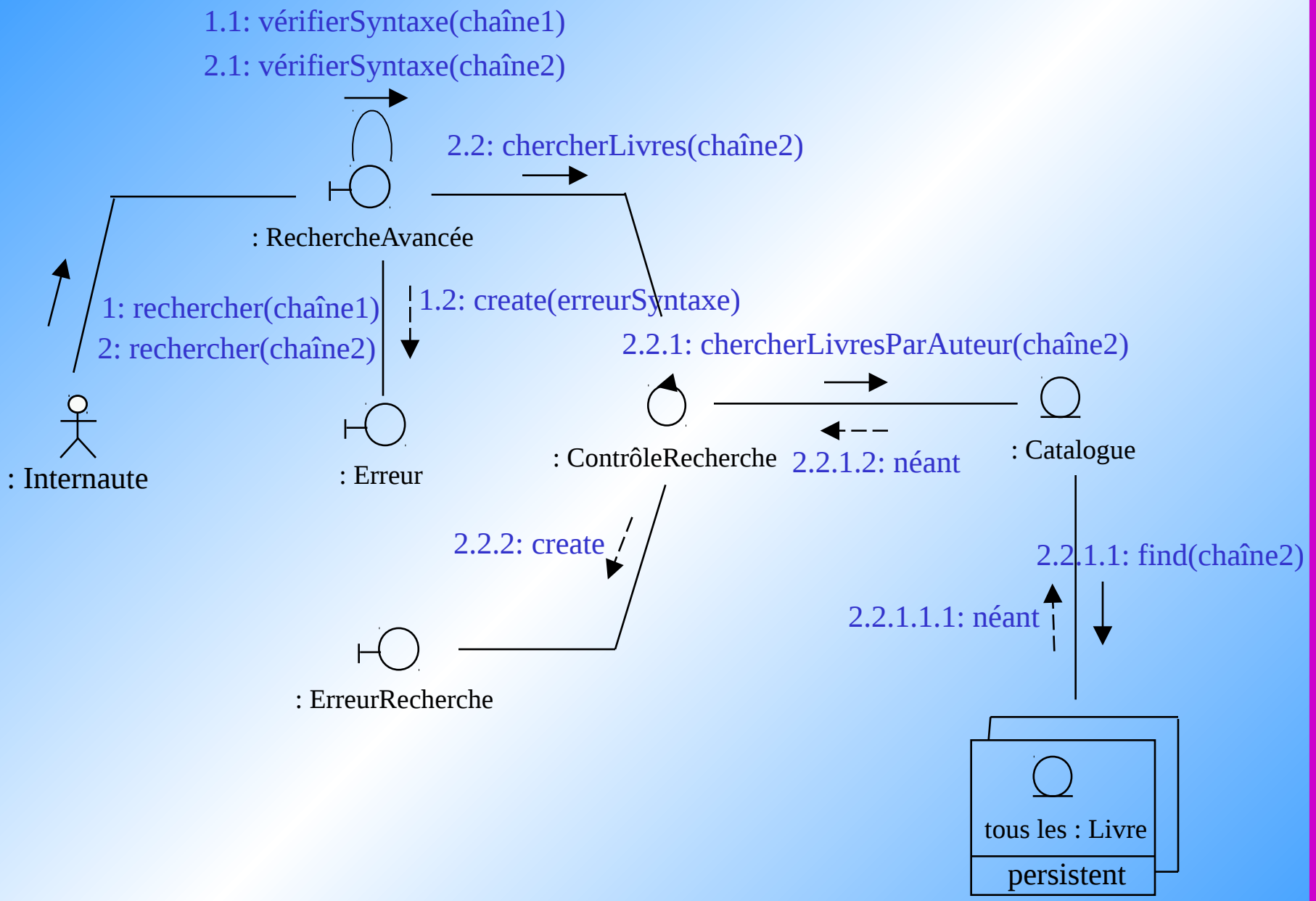
Faire un diagramme de communication

Rechercher des ouvrages

Échec de la recherche

1. Erreur de syntaxe
2. Pas d'ouvrage correspondant à la requête

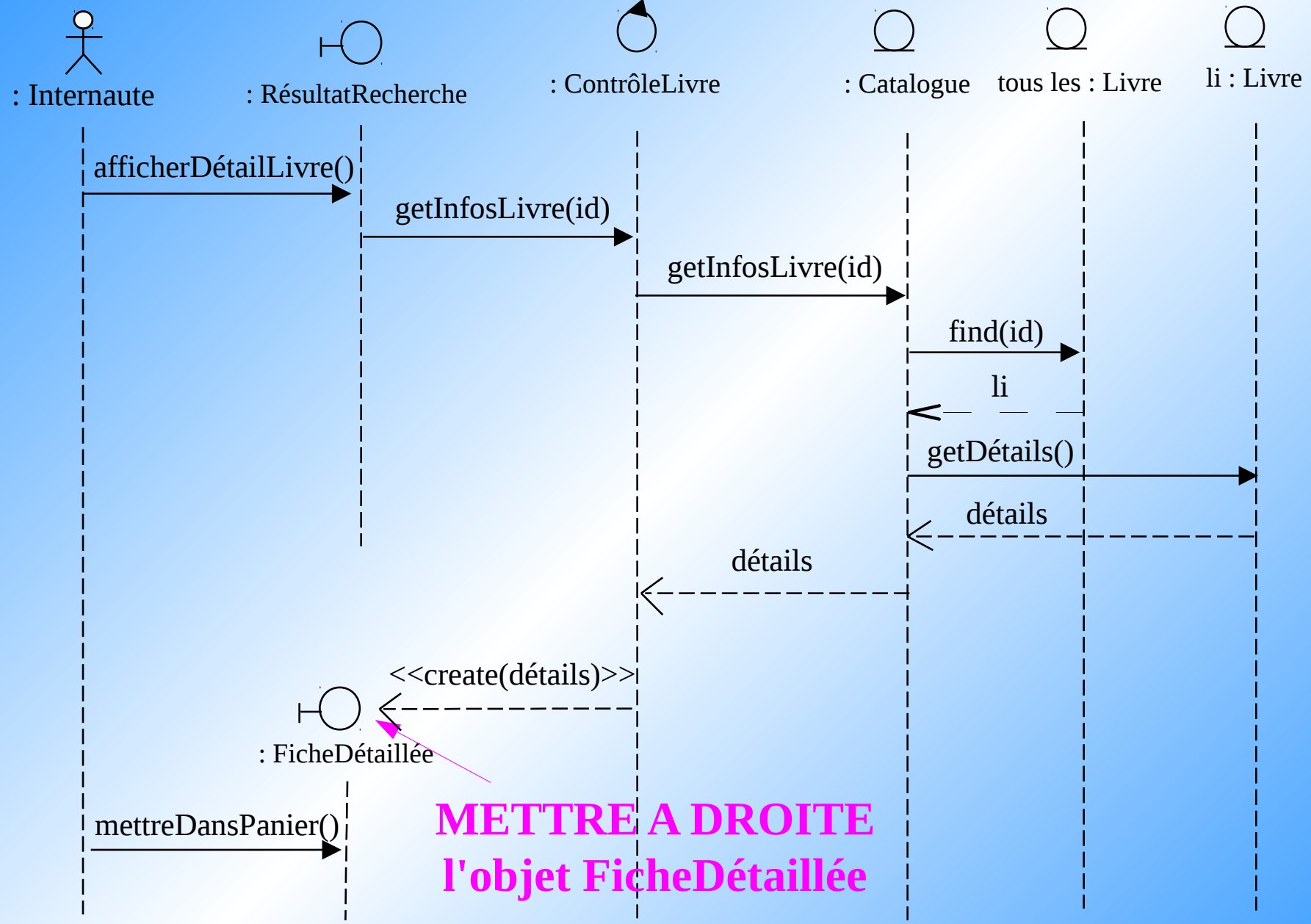

1.1: vérifierSyntaxe(chaîne1)
2.1: vérifierSyntaxe(chaîne2)
1.2: erreur
2.2: chercherLivres(chaîne2)
: RechercheAvancée
1: rechercher(chaîne1)
2: rechercher(chaîne2)
2.2.1: chercherLivresParAuteur(chaîne2)
: Internaute
: ContrôleRecherche
2.2.1.2: néant
: Catalogue
2.2.2: create
2.2.1.1: find(chaîne2)
2.2.1.1.1: néant
: ErreurRecherche
tous les : Livre
persistent


1.1: vérifierSyntaxe(chaîne1)
2.1: vérifierSyntaxe(chaîne2)
2.2: chercherLivres(chaîne2)
: RechercheAvancée
1: rechercher(chaîne1)
2: rechercher(chaîne2)
1.2: create(erreurSyntaxe)
2.2.1: chercherLivresParAuteur(chaîne2)
: Internaute
: Erreur
: ContrôleRecherche
2.2.1.2: néant
: Catalogue
2.2.2: create
2.2.1.1: find(chaîne2)
2.2.1.1.1: néant
: ErreurRecherche
tous les : Livre
persistent

diagrammes d'interaction site web

Faire un diagramme de séquence
Rechercher des ouvrages

Affichage du détail d'un livre


: Internaute
: RésultatRecherche
: ContrôleLivre
: Catalogue
tous les : Livre
li : Livre
afficherDétailLivre()
getInfosLivre(id)
getInfosLivre(id)
find(id)
li
getDétails()
détails
détails
<<create(détails)>>
: FicheDétaillée
mettreDansPanier()
METTRE A DROITE
l'objet FicheDétaillée

Solution

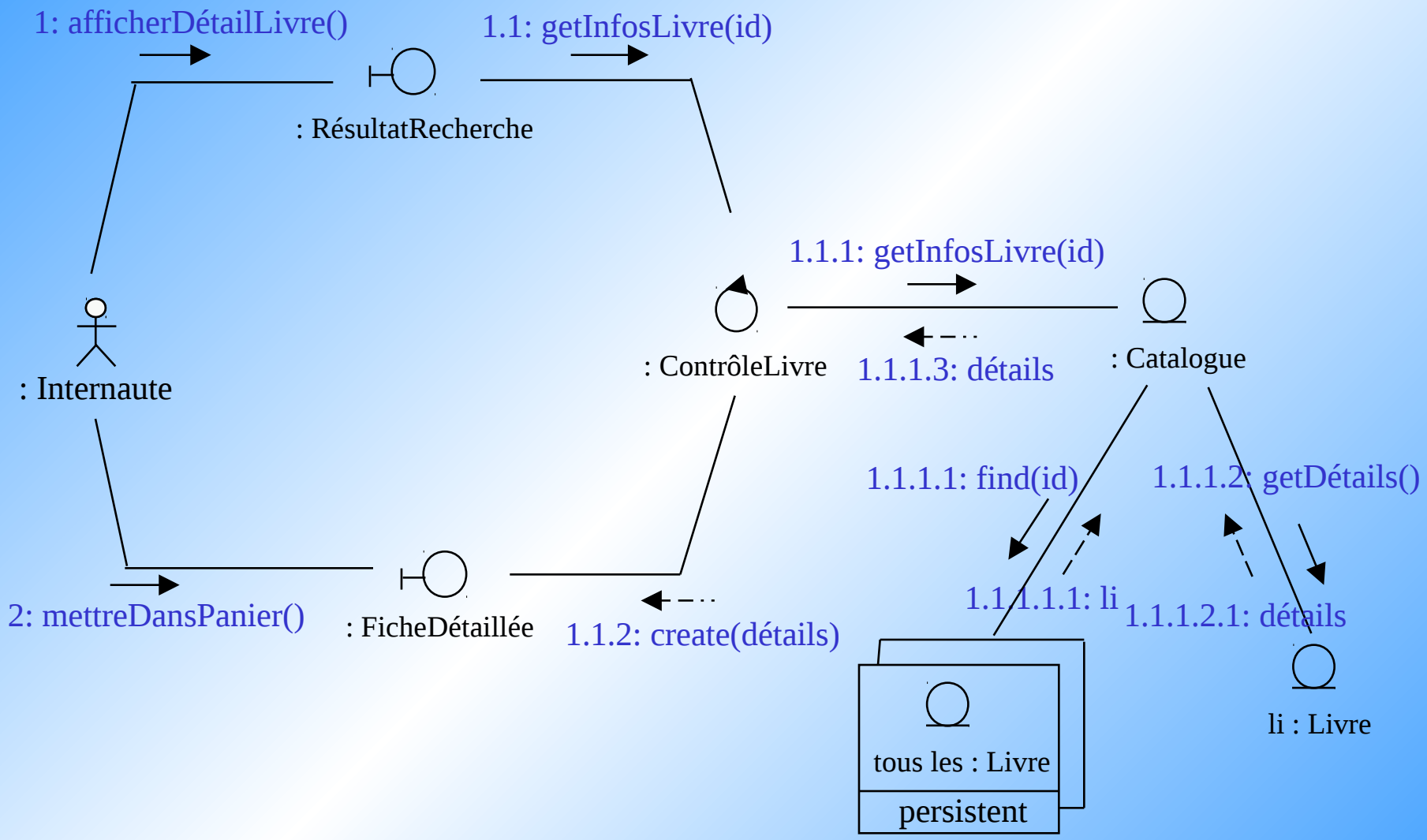
- L’internaute peut demander l’affichage du détail d’un livre sélectionné parmi ceux de la page de résultats.
- Le dialogue passe la main à un contrôle spécialisé qui sait récupérer les informations détaillées d’un livre à partir de son identifiant.
- Pour cela, il fait encore appel à l’expert des livres, à savoir l’entité catalogue.
- Ensuite, le contrôle crée un nouveau dialogue de fiche détaillée à partir de ces informations.
- C’est par exemple à ce moment-là que l’internaute choisit de mettre le livre sélectionné dans son panier virtuel.

diagrammes d'interaction site web

Faire un diagramme de communication

Rechercher des ouvrages

Affichage du détail d'un livre


1: afficherDétailLivre()
1.1: getInfosLivre(id)
: RésultatRecherche
1.1.1: getInfosLivre(id)
: ContrôleLivre
1.1.1.3: détails
: Catalogue
: Internaute
1.1.1.1: find(id)
1.1.1.2: getDétails()
1.1.1.1.1: li
1.1.1.2.1: détails
2: mettreDansPanier()
: FicheDétaillée
1.1.2: create(détails)
tous les : Livre
persistent
li : Livre

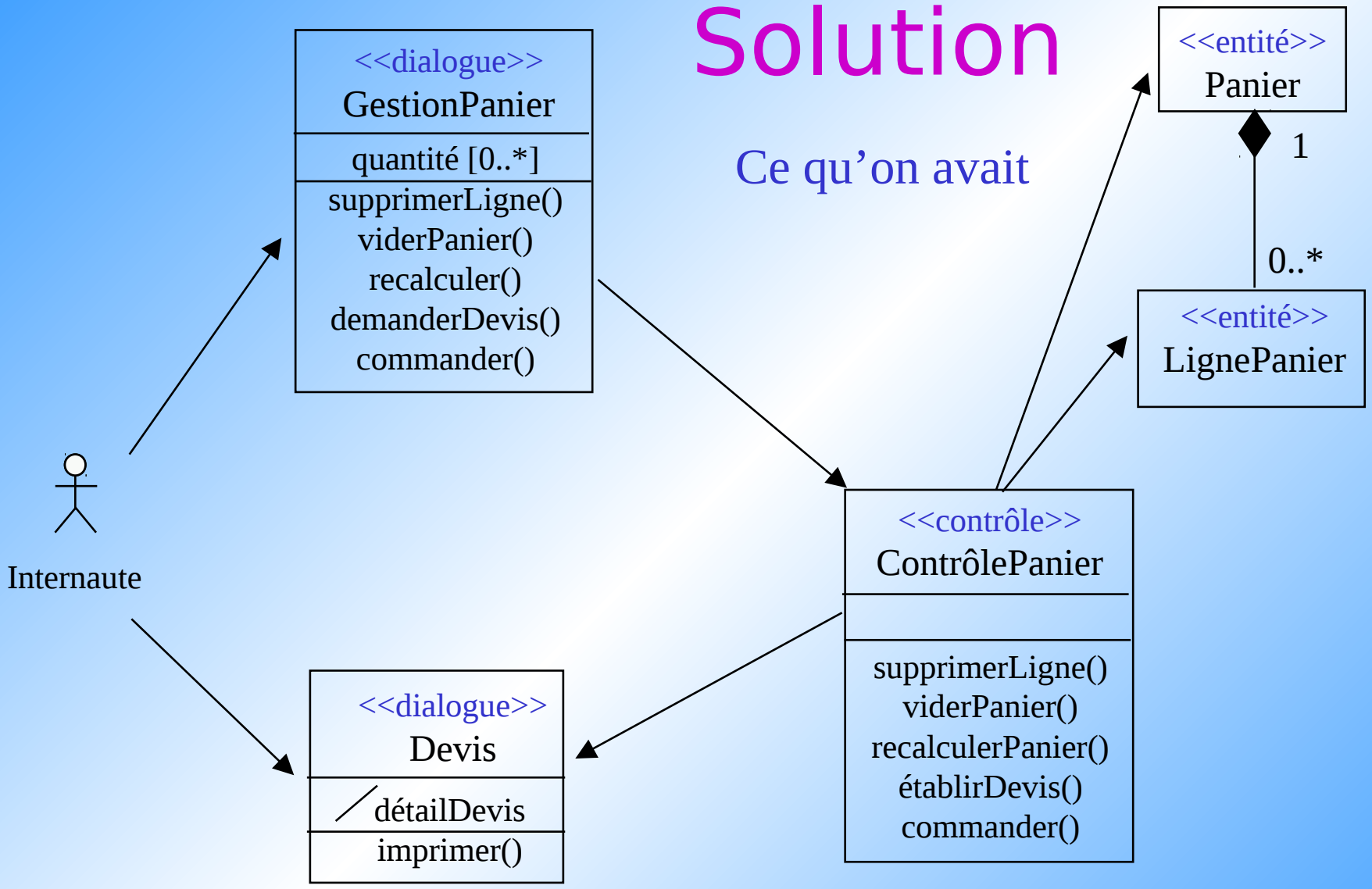
diagrammes d'interaction site web

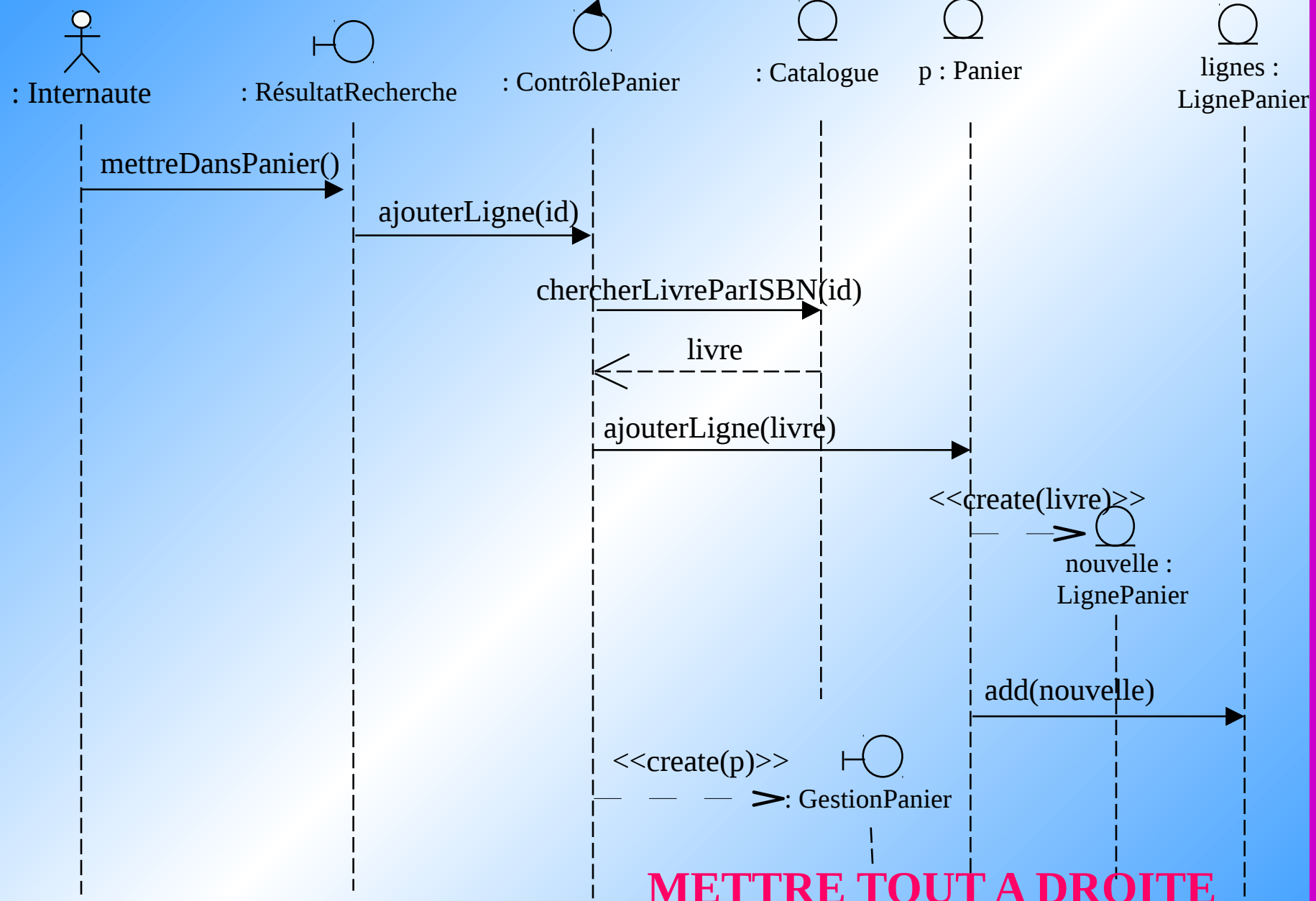
Faire un diagramme de séquence
Gérer son panier

L'internaute met un livre dans son panier
(*pas le premier livre*)

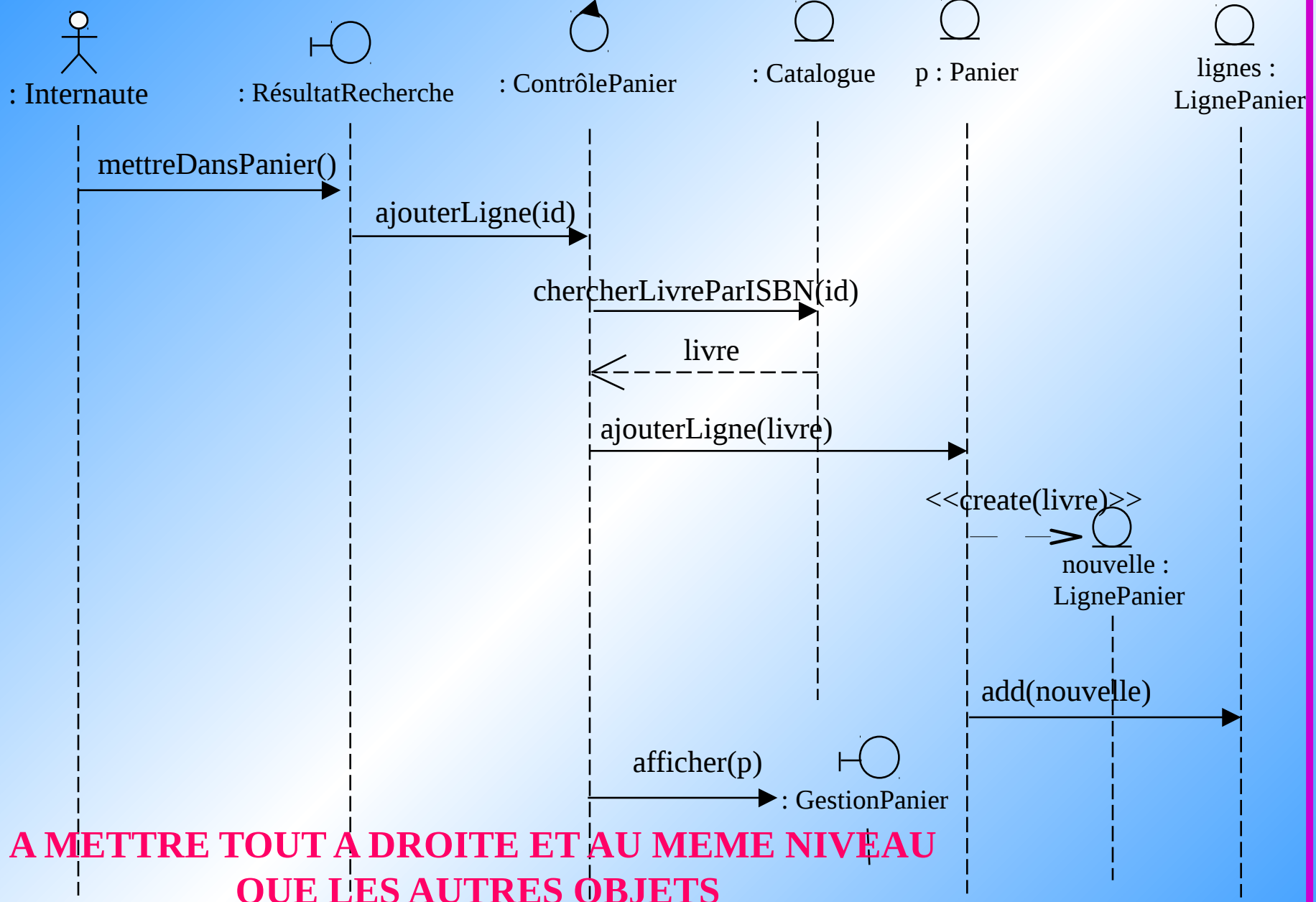
Solution

Ce qu'on avait





METTRE TOUT A DROITE l'objet GestionPanier



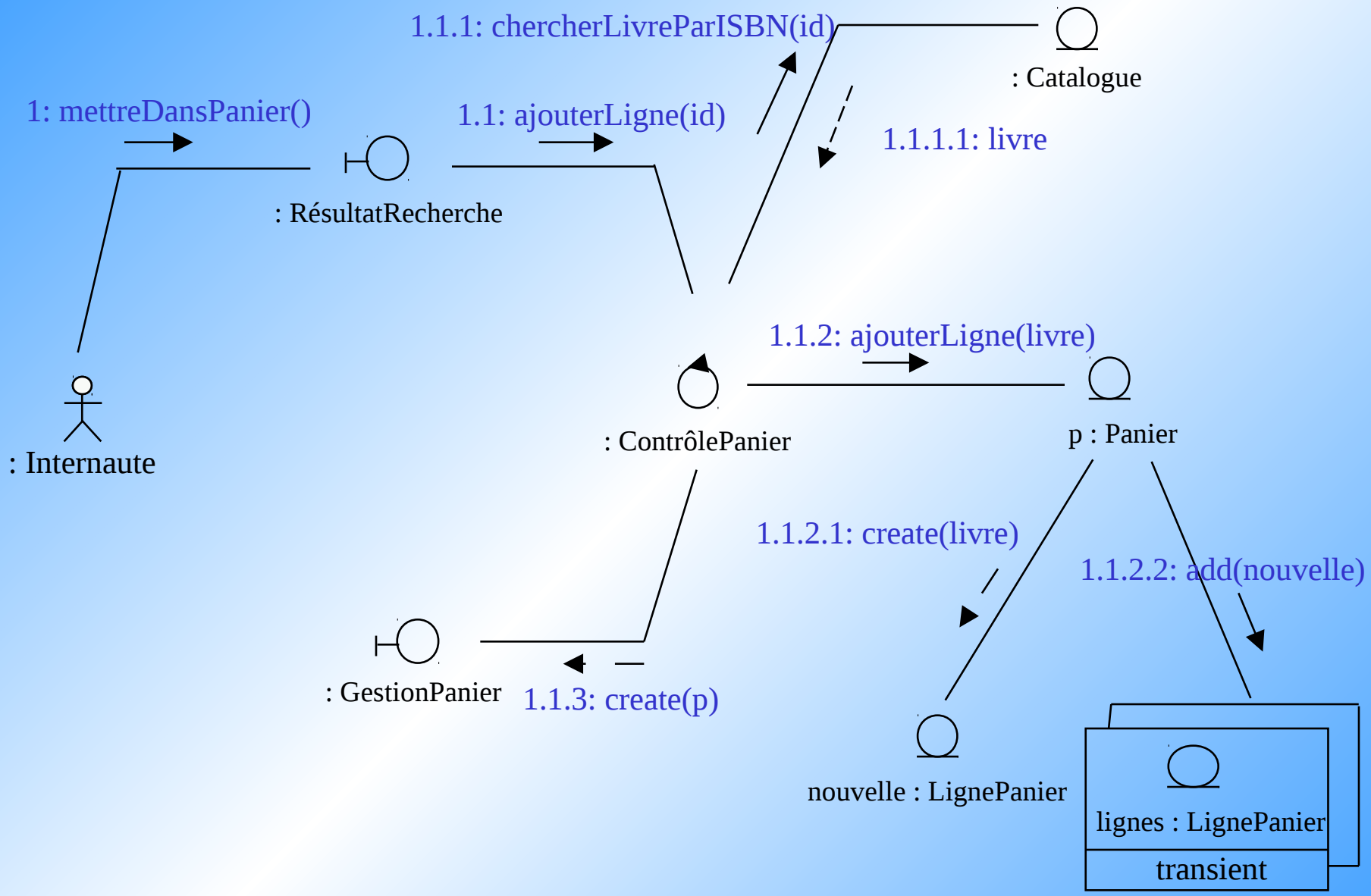
A METTRE TOUT A DROITE ET AU MEME NIVEAU QUE LES AUTRES OBJETS

diagrammes d’interaction site web

Faire un diagramme de communication

Gérer son panier

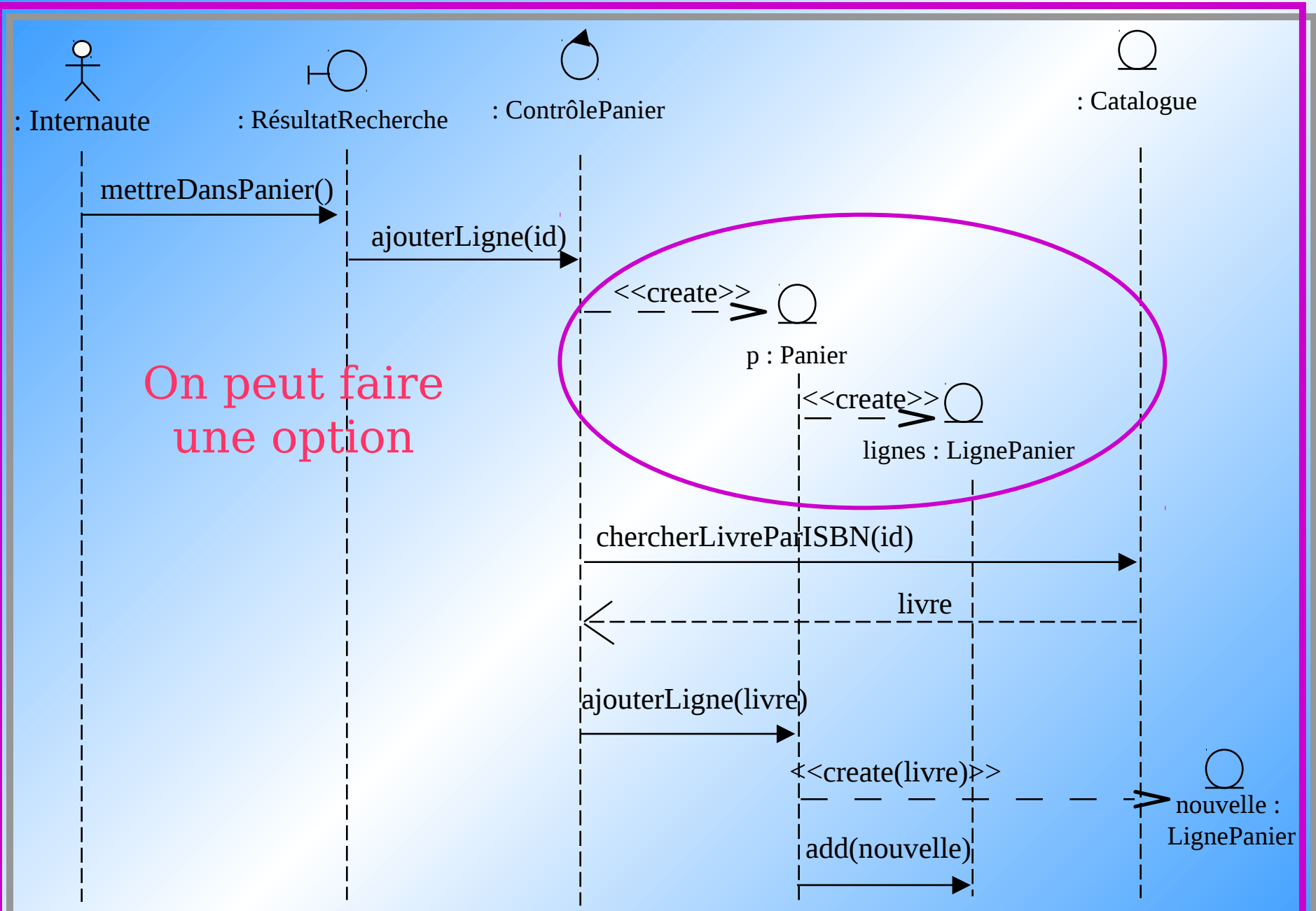
L'internaute met un livre dans son panier
(*pas le premier livre*)


1.1.1: chercherLivreParISBN(id)
: Catalogue
1: mettreDansPanier()
1.1: ajouterLigne(id)
1.1.1.1: livre
: RésultatRecherche
1.1.2: ajouterLigne(livre)
: Internaute
: ContrôlePanier
p : Panier
1.1.2.1: create(livre)
1.1.2.2: add(nouvelle)
: GestionPanier
1.1.3: create(p)
nouvelle : LignePanier
lignes : LignePanier
transient

diagrammes d’interaction site web

Faire un diagramme de séquence
Gérer son panier

L'internaute met un premier livre dans son panier


: Internaute
: RésultatRecherche
: ContrôlePanier
: Catalogue
mettreDansPanier()
ajouterLigne(id)
<<create>>
p : Panier
<<create>>
lignes : LignePanier
On peut faire une option
chercherLivreParISBN(id)
livre
ajouterLigne(livre)
<<create(livre)>>
nouvelle : LignePanier
add(nouvelle)

Solution

- L’internaute met de côté un premier livre dans son panier virtuel.

- Le dialogue passe la main à un contrôle spécialisé dans la gestion du panier, qui a la responsabilité de créer le panier lors de la première sélection mais aussi toutes les lignes du panier au fur et à mesure.

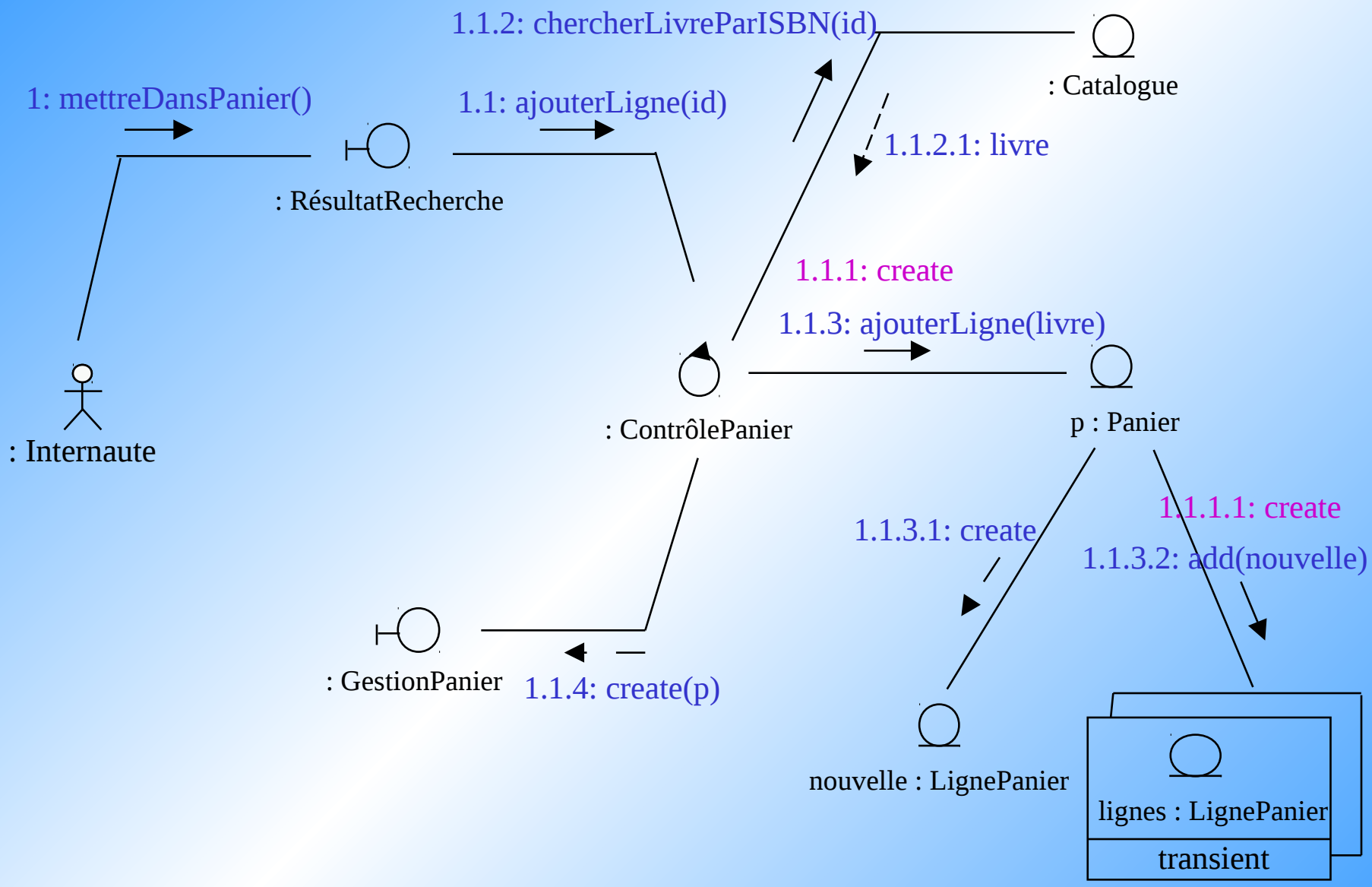
- Le contrôle est également responsable de l’affichage d’un dialogue particulier qui récapitule le panier en cours et permet ensuite à l’internaute de le modifier et de le recalculer.

diagrammes d’interaction site web

Faire un diagramme de communication

Gérer son panier

L'internaute met un premier livre dans son panier


1.1.2: chercherLivreParISBN(id)
: Catalogue
1: mettreDansPanier()
1.1: ajouterLigne(id)
1.1.2.1: livre
: RésultatRecherche
1.1.1: create
1.1.3: ajouterLigne(livre)
: Internaute
: ContrôlePanier
p : Panier
1.1.3.1: create
1.1.1.1: create
1.1.3.2: add(nouvelle)
: GestionPanier
1.1.4: create(p)
nouvelle : LignePanier
lignes : LignePanier
transient

Solution

- L’initialisation du panier provoque la création de la collection vide de lignes du panier.

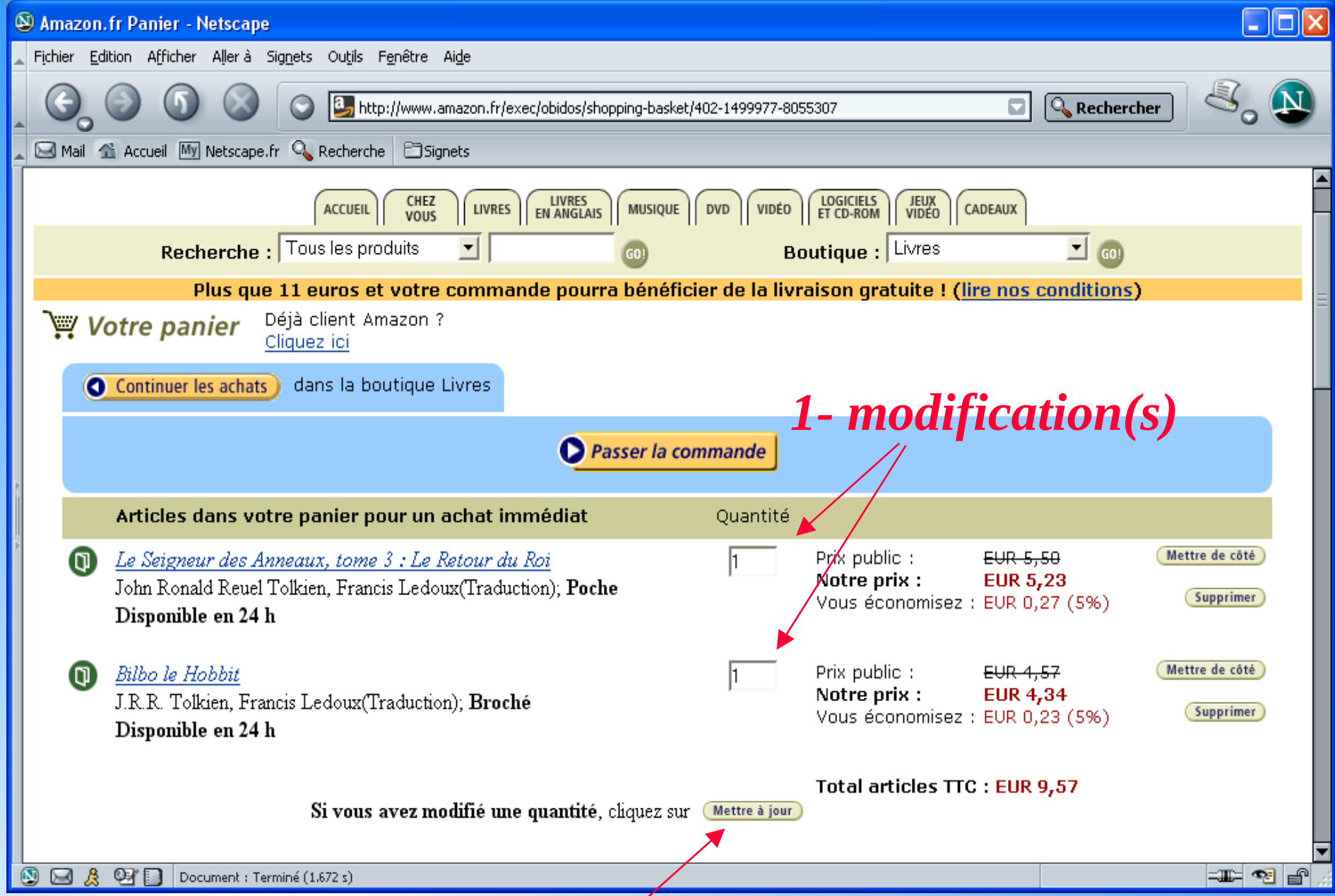
- Cette collection n’est pas persistante (*transient*) : elle ne survit pas à une session de l’internaute.

- Lors de la création des lignes suivantes du panier, il suffit de créer un objet *LignePanier* et de l’ajouter à la collection de lignes du panier.

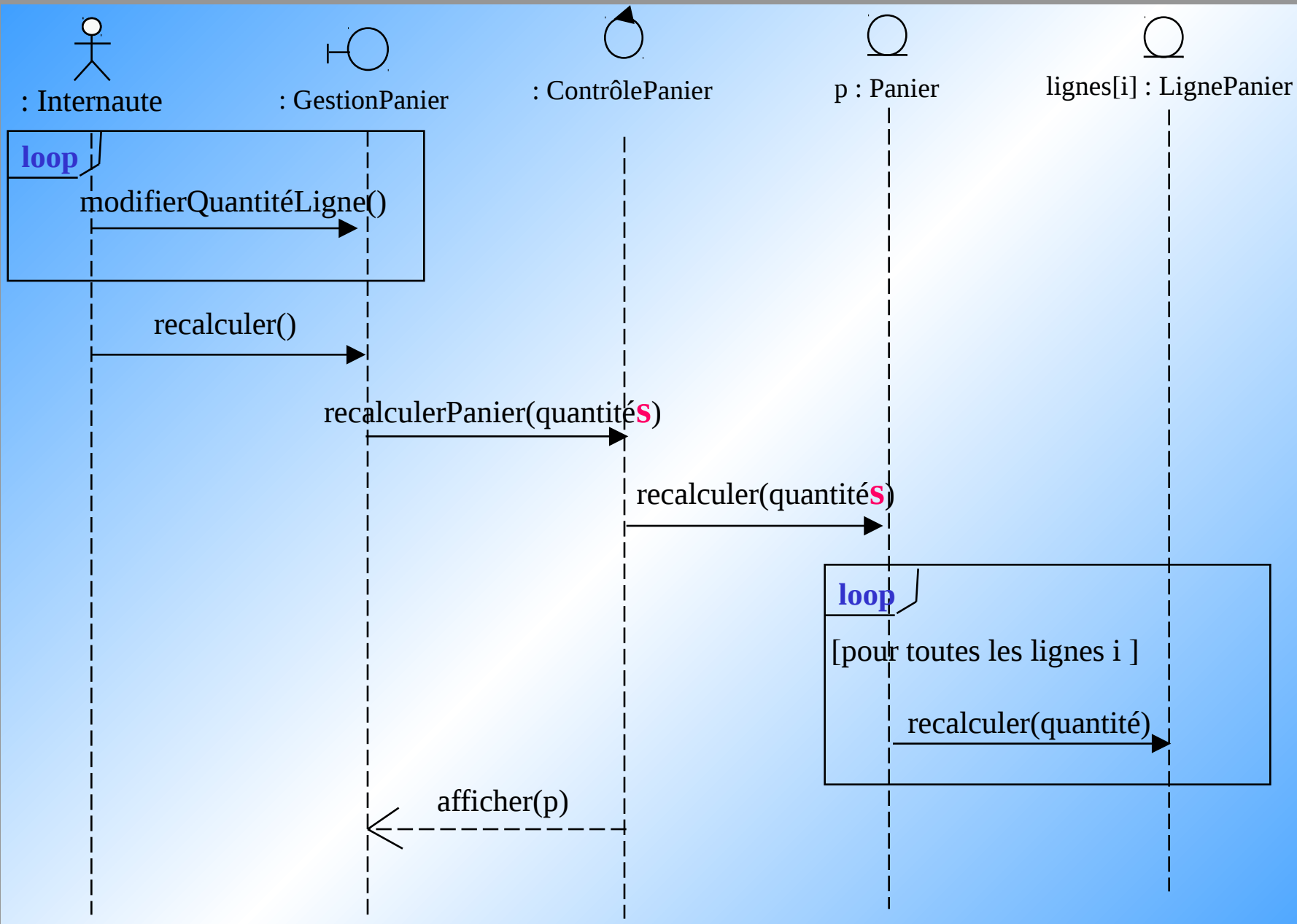
diagrammes d’interaction site web

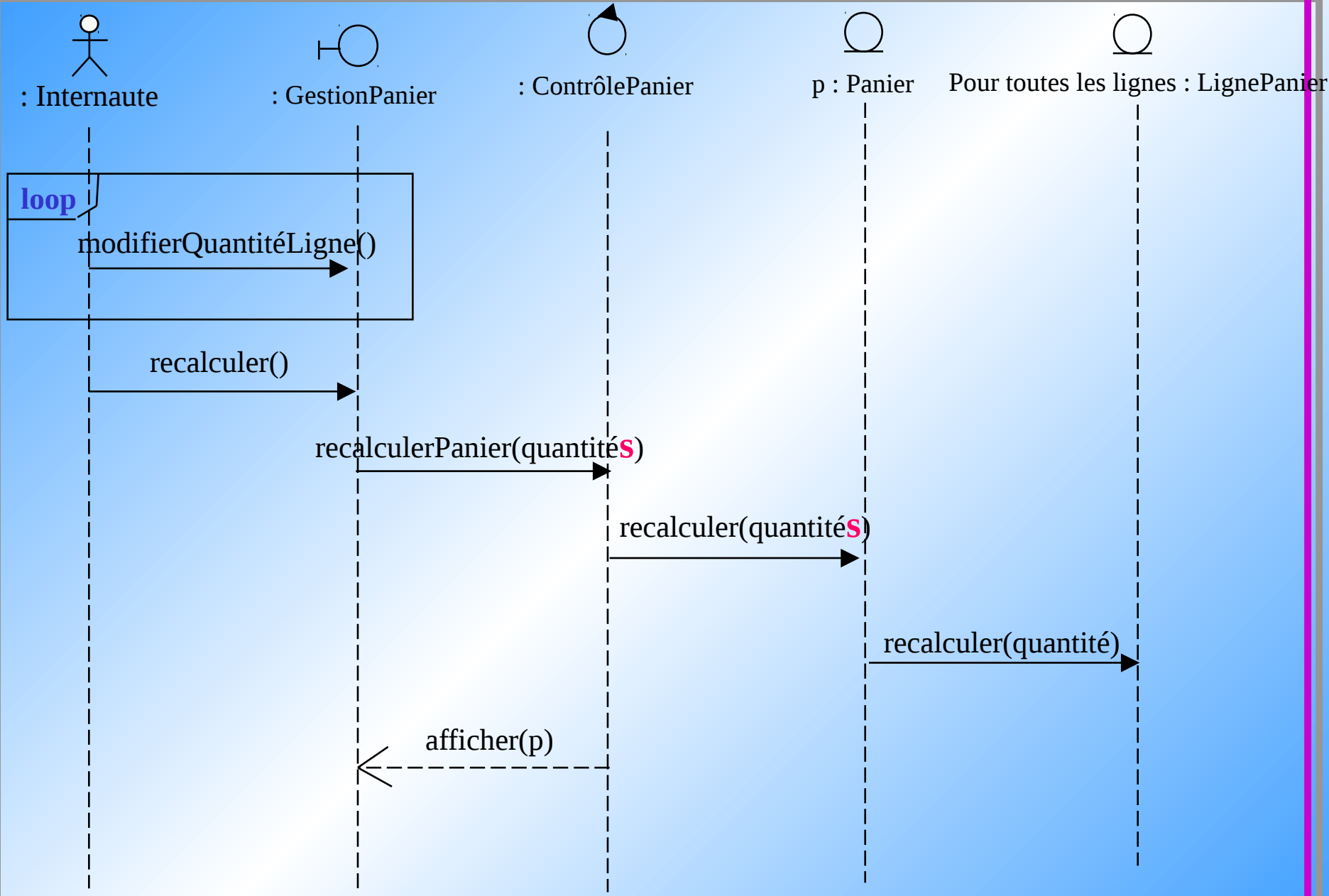
Faire un diagramme de séquence
Gérer son panier

L'internaute modifie la quantité d’un ouvrage ou
plusieurs ouvrages sélectionnés
puis demande un recalcul de son panier



2- puis mise à jour


: Internaute
: GestionPanier
: ContrôlePanier
p : Panier
lignes[i] : LignePanier
loop
modifierQuantitéLigne()
recalculer()
recalculerPanier(quantités)
recalculer(quantités)
loop
[pour toutes les lignes i]
recalculer(quantité)
afficher(p)


: Internaute
: GestionPanier
: ContrôlePanier
p : Panier
Pour toutes les lignes : LignePanier
loop
modifierQuantitéLigne()
recalculer()
recalculerPanier(quantités)
recalculer(quantités)
recalculer(quantité)
afficher(p)

Solution

- Le contrôle reçoit une collection de quantités et la passe à l'entité panier.

- Celui-ci est responsable de la gestion de ses lignes : il va donc demander à chaque ligne de se recalculer individuellement en lui passant en paramètre la quantité qui la concerne.

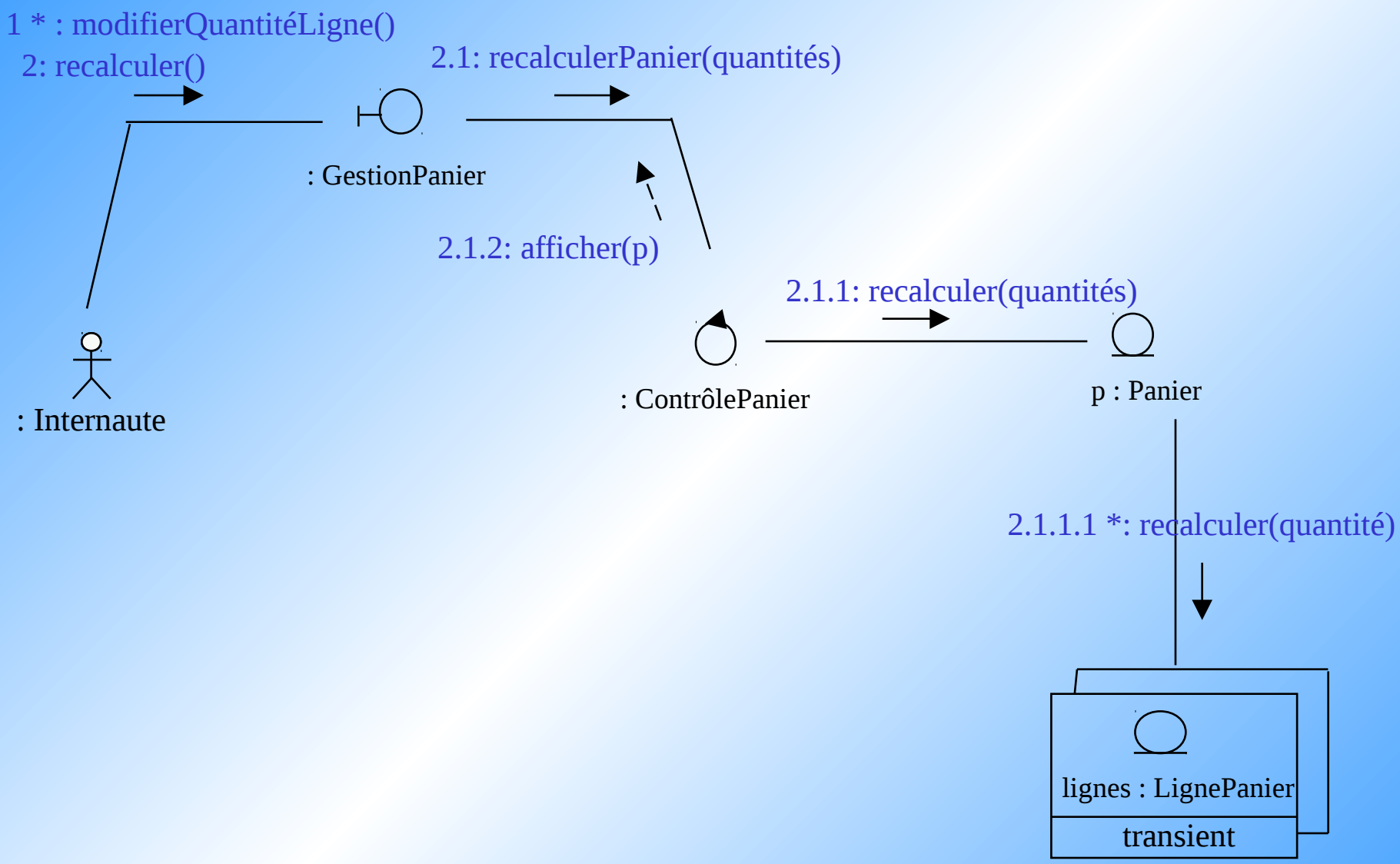
- Si cette quantité a été positionnée à zéro par l'internaute, la ligne est supprimée.

diagrammes d’interaction site web

Faire un diagramme de communication

Gérer son panier

L'internaute modifie la quantité d’un ouvrage sélectionné puis demande un recalcul de son panier


1 * : modifierQuantitéLigne()
2: recalculer()
2.1: recalculerPanier(quantités)
: GestionPanier
2.1.2: afficher(p)
2.1.1: recalculer(quantités)
: Internaute
: ContrôlePanier
p : Panier
2.1.1.1 *: recalculer(quantité)
lignes : LignePanier
transient

Solution

Itération sur une collection (*multi-objet*)

Un algorithme très courant consiste à opérer une itération (*ou boucle*) sur tous les éléments d'une collection en envoyant un message à chacun d'eux.

- On a déjà vu qu'une collection d'instances est modélisée sur le diagramme de collaboration par un multi-objet (*avec un double cadre*).

- Le marqueur de multiplicité * préfixant le message indique que celui-ci est adressé à chaque élément de la collection au lieu d'être envoyé de façon répétitive à la collection elle-même.

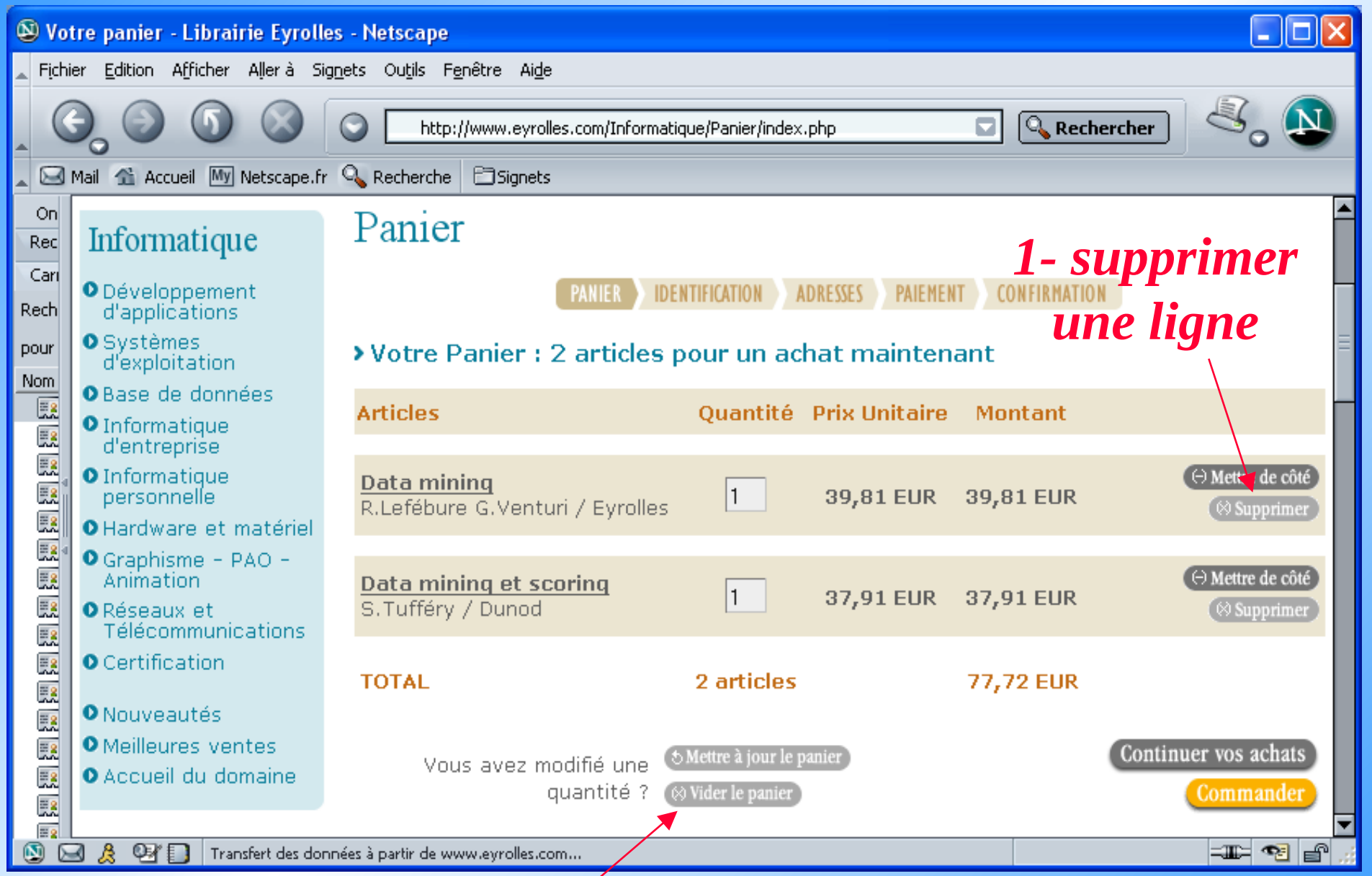
C'est le cas pour le message *2.1.1.1. *recalculer(q)* qui est envoyé par le panier à chacune de ses lignes.

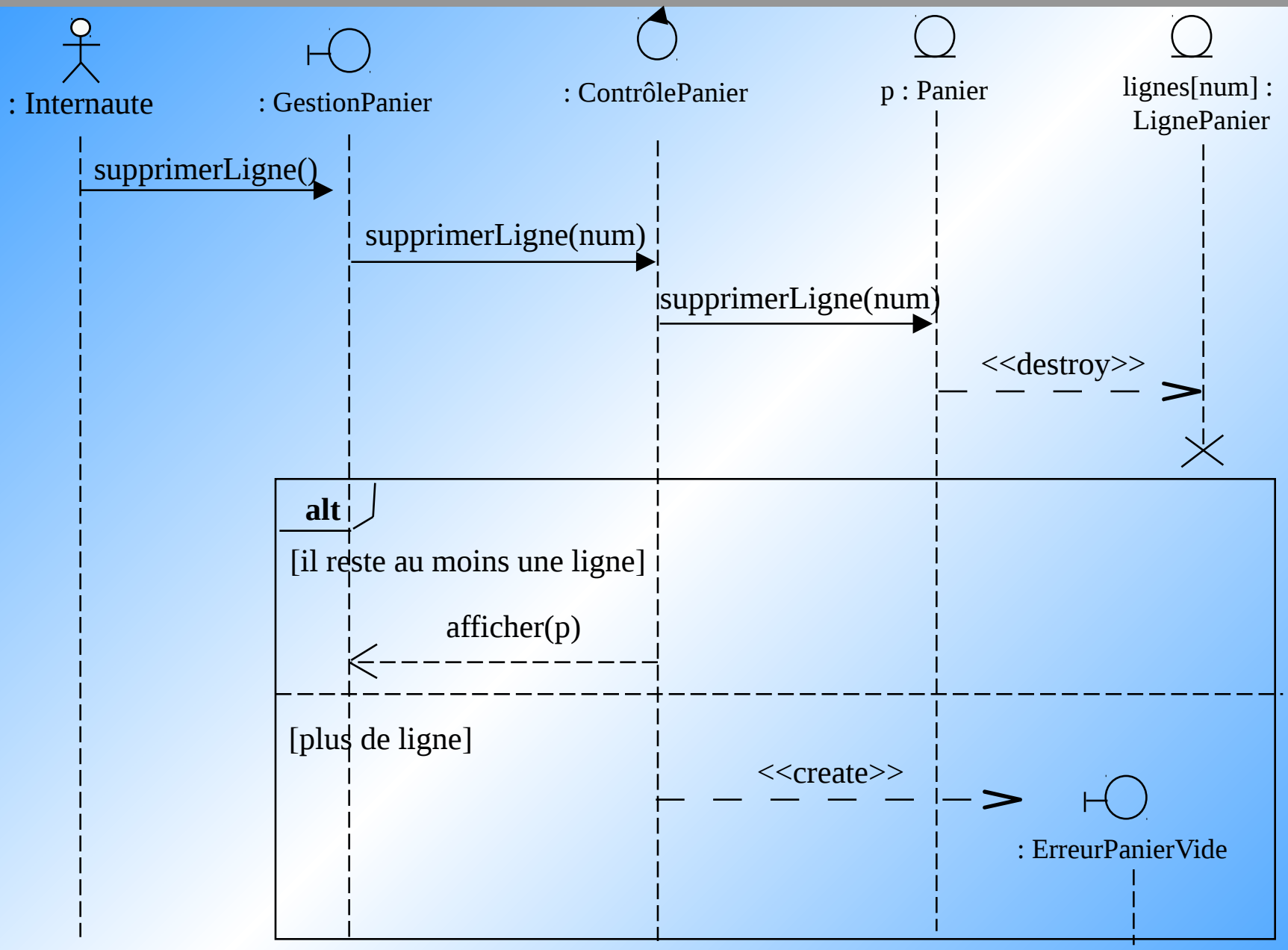
diagrammes d’interaction site web

Faire un diagramme de séquence

Gérer son panier

L'internaute supprime une ligne


Votre panier - Librairie Eyrolles - Netscape
Fichier Edition Afficher Aller à Signets Outils Fenêtre Aide
http://www.eyrolles.com/Informatique/Panier/index.php
Rechercher
Mail Accueil Netscape.fr Recherche Signets
Informatique
Développement d'applications
Systèmes d'exploitation
Base de données
Informatique d'entreprise
Informatique personnelle
Hardware et matériel
Graphisme - PAO - Animation
Réseaux et Télécommunications
Certification
Nouveautés
Meilleures ventes
Accueil du domaine
Panier
PANIER IDENTIFICATION ADRESSES PAIEMENT CONFIRMATION
Votre Panier : 2 articles pour un achat maintenant
Articles Quantité Prix Unitaire Montant
Data mining
R.Lefébure G.Venturi / Eyrolles
1 39,81 EUR 39,81 EUR
Mettre de côté
Supprimer
Data mining et scoring
S.Tufféry / Dunod
1 37,91 EUR 37,91 EUR
Mettre de côté
Supprimer
TOTAL 2 articles 77,72 EUR
Vous avez modifié une quantité ?
Mettre à jour le panier
Vider le panier
Continuer vos achats
Commander
Transfert des données à partir de www.eyrolles.com...
1- supprimer une ligne
2- vider

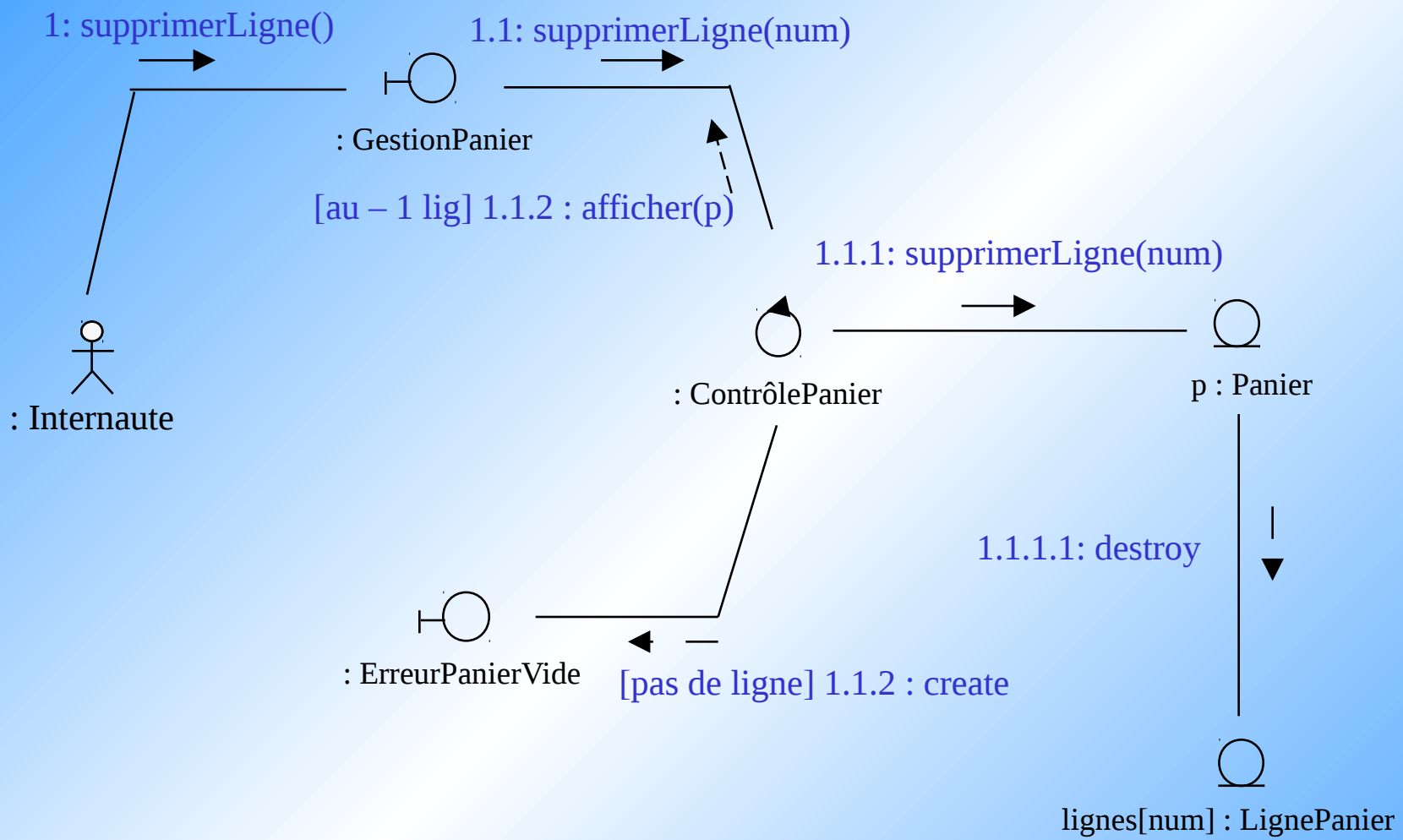

: Internaute
: GestionPanier
: ContrôlePanier
p : Panier
lignes[num] : LignePanier
supprimerLigne()
supprimerLigne(num)
supprimerLigne(num)
<<destroy>>
alt
[il reste au moins une ligne]
afficher(p)
[plus de ligne]
<<create>>
: ErreurPanierVide

diagrammes d’interaction site web

Faire un diagramme de communication

Gérer son panier

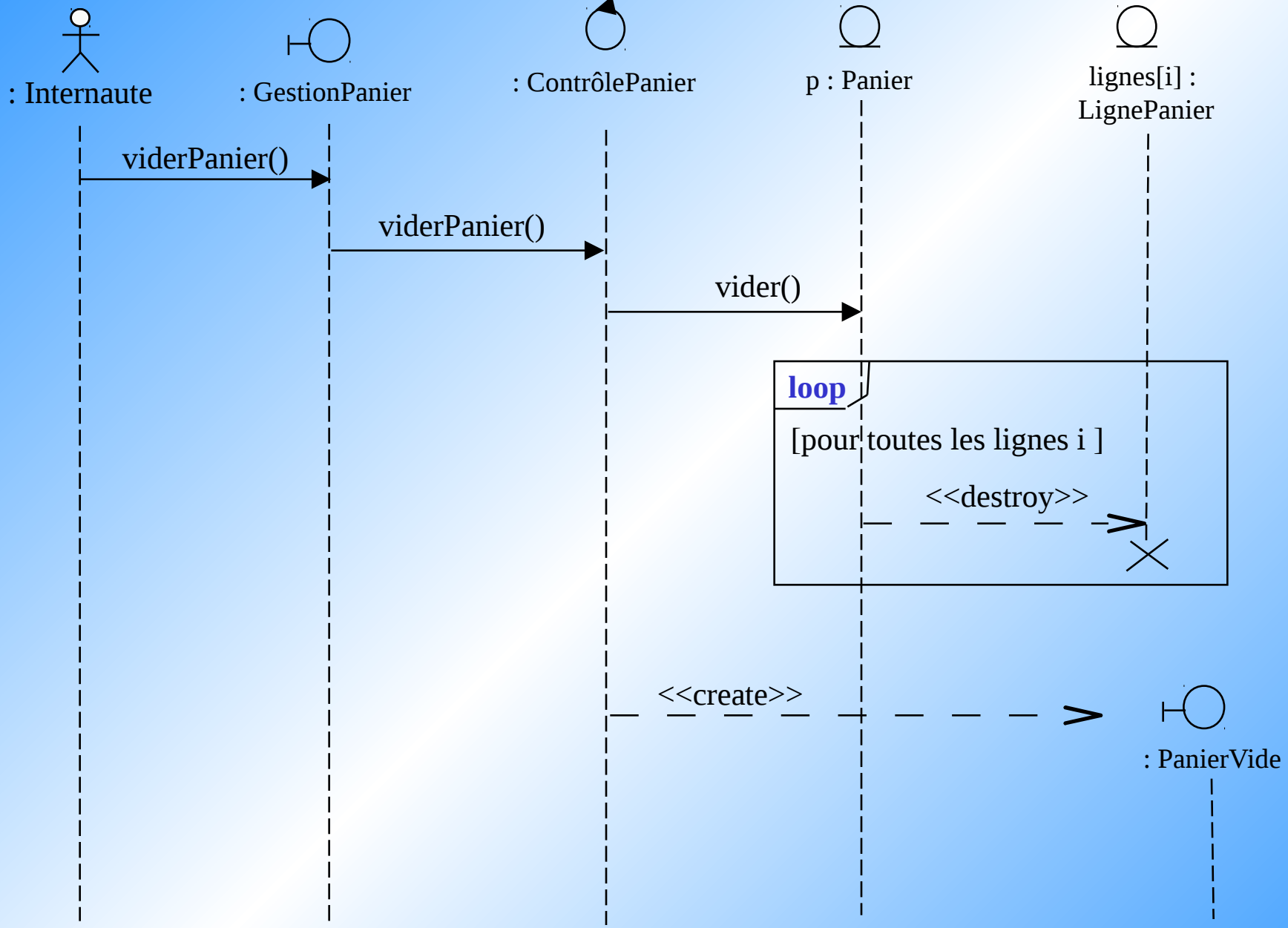
L'internaute supprime une ligne


1: supprimerLigne()
1.1: supprimerLigne(num)
: GestionPanier
[au – 1 lig] 1.1.2 : afficher(p)
1.1.1: supprimerLigne(num)
: Internaute
: ContrôlePanier
p : Panier
1.1.1.1: destroy
: ErreurPanierVide
[pas de ligne] 1.1.2 : create
lignes[num] : LignePanier

diagrammes d’interaction site web

Faire un diagramme de séquence
Gérer son panier

L'internaute supprime toutes les lignes de son panier

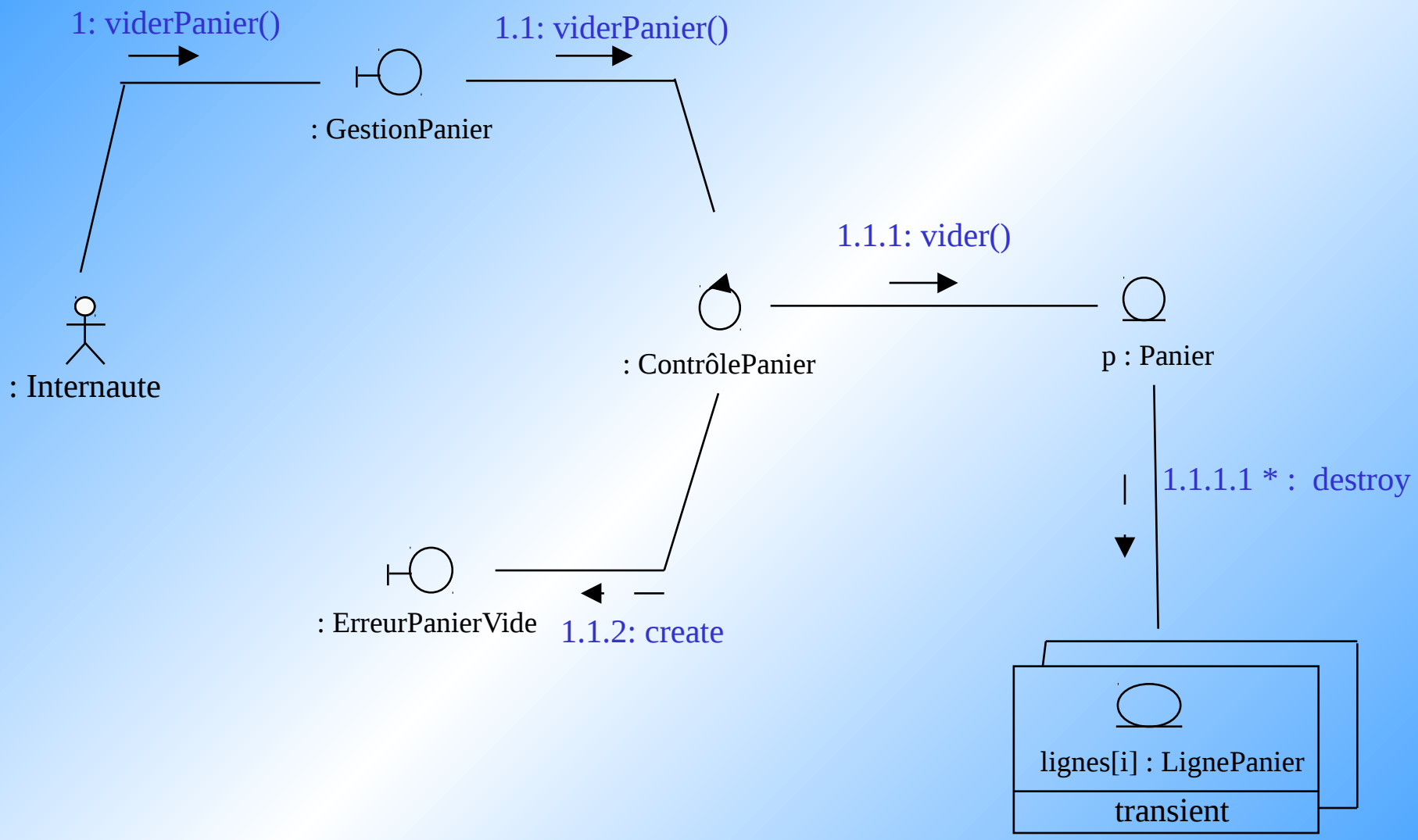

: Internaute
: GestionPanier
: ContrôlePanier
p : Panier
lignes[i] :
LignePanier
viderPanier()
viderPanier()
vider()
loop
[pour toutes les lignes i]
<<destroy>>
<<create>>
: PanierVide

diagrammes d’interaction site web

Faire un diagramme de communication

Gérer son panier

L'internaute supprime toutes les lignes de son panier


1: viderPanier()
1.1: viderPanier()
: GestionPanier
1.1.1: vider()
: Internaute
: ContrôlePanier
p : Panier
1.1.1.1 * : destroy
: ErreurPanierVide
1.1.2: create
lignes[i] : LignePanier
transient

Solution

Dans le cas où l'internaute demande à vider son panier,
celui-ci n'est pas supprimé,
mais sont supprimées uniquement les lignes qu'il contient
(*grâce à l'itération *destroy*).

Le panier n'est supprimé qu'avec la fin de la session de l'internaute.

Sommaire

- Introduction
- Diagrammes d’interaction
- Diagrammes pour le site Web
- **Classes de conception préliminaire**

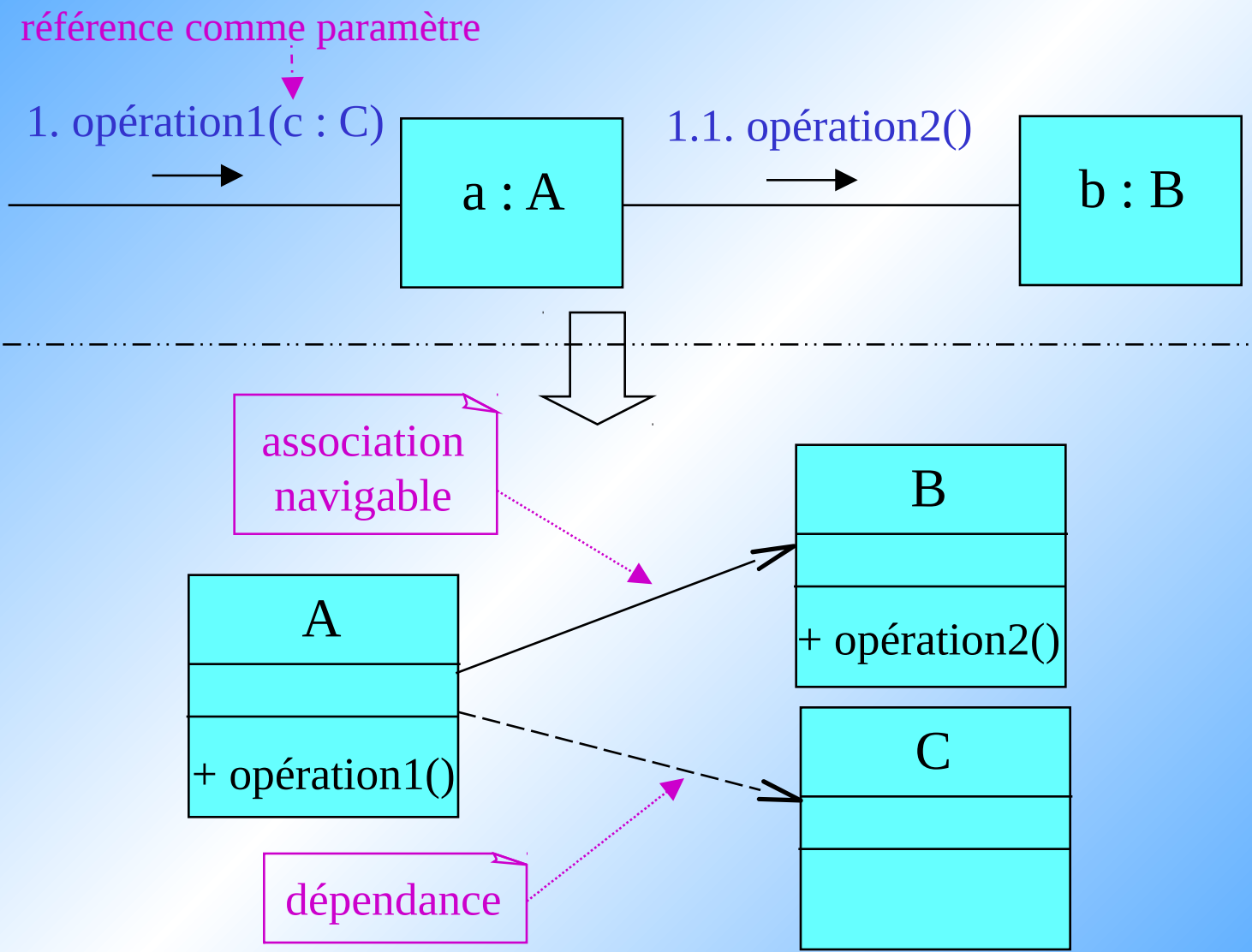
Classes de conception préliminaire

- On va affiner et compléter les diagrammes de classes déjà obtenus (*analyse du domaine + classes participantes*).

- Pour cela, on utilise les diagrammes d'interaction qu'on vient de réaliser pour :

- **ajouter** ou **préciser les opérations dans les classes**
  (*un message ne peut être reçu par un objet que si sa classe a déclaré l'opération publique correspondante*) ;
- **ajouter des types aux attributs**,
  aux paramètres et
  aux retours des opérations ;
- **affiner les relations entre classes**
  (*indication de navigabilité, dépendances*).

Classes de conception préliminaire



Classes de conception préliminaire

Liens durables ou temporaires

- Un **lien durable** entre objets va donner lieu à une association navigable entre les classes correspondantes.

Le lien entre l'objet a et l'objet b devient une association navigable entre les classes A et B.

- Un **lien temporaire** va donner lieu à une relation de dépendance.

Le fait que l'objet a reçoive en paramètre d'un message une référence sur un objet de la classe C induit une dépendance entre les classes A et C.

Classes de conception préliminaire

Les types ne sont pas encore ceux d'un langage de programmation, puisqu'on veut rester **indépendants des choix technologiques** à ce niveau.

On utilisera bien des noms comme *String* ou *Date* dans les diagrammes qui suivent ; ce ne sont pas les types Java, mais bien des types génériques.

Classes de conception préliminaire

Les multi-objets ne sont pas représentés en tant que classes collections de façon à rester le plus longtemps possible **indépendants du langage de programmation cible**.
De ce fait, les opérations génériques sur les collections (comme *find* ou *add*) n'apparaissent pas.

On ne fait pas figurer les opérations systématiques de création ou destruction d'instances ainsi que les accesseurs des attributs (*get* et *set*). Cela permet de garder des diagrammes lisibles et qui contiennent seulement les informations les plus importantes.

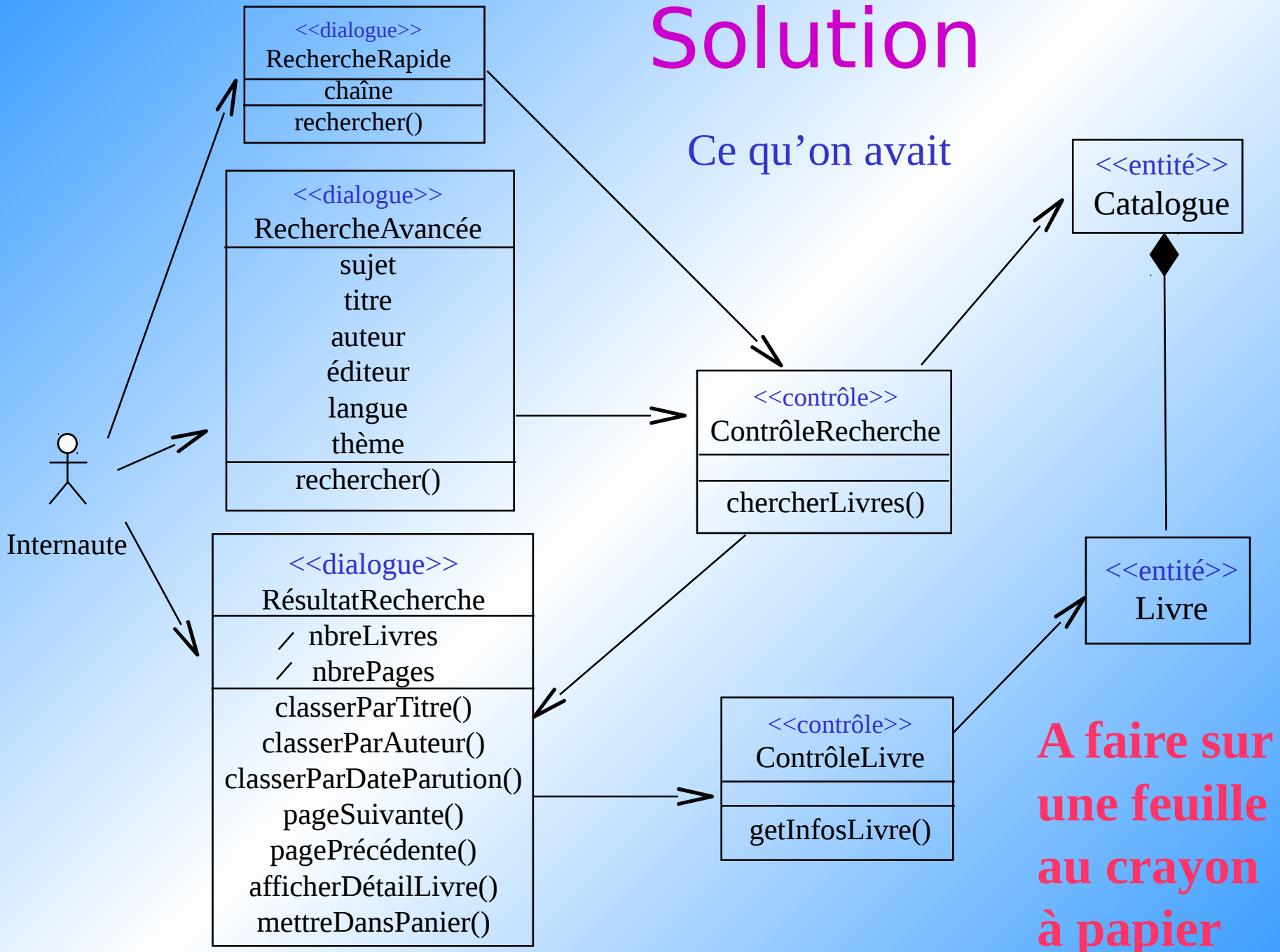
Classes de conception préliminaire

Faire le diagramme de classes de conception préliminaire

Rechercher des ouvrages

Solution

Ce qu'on avait



A faire sur une feuille au crayon à papier

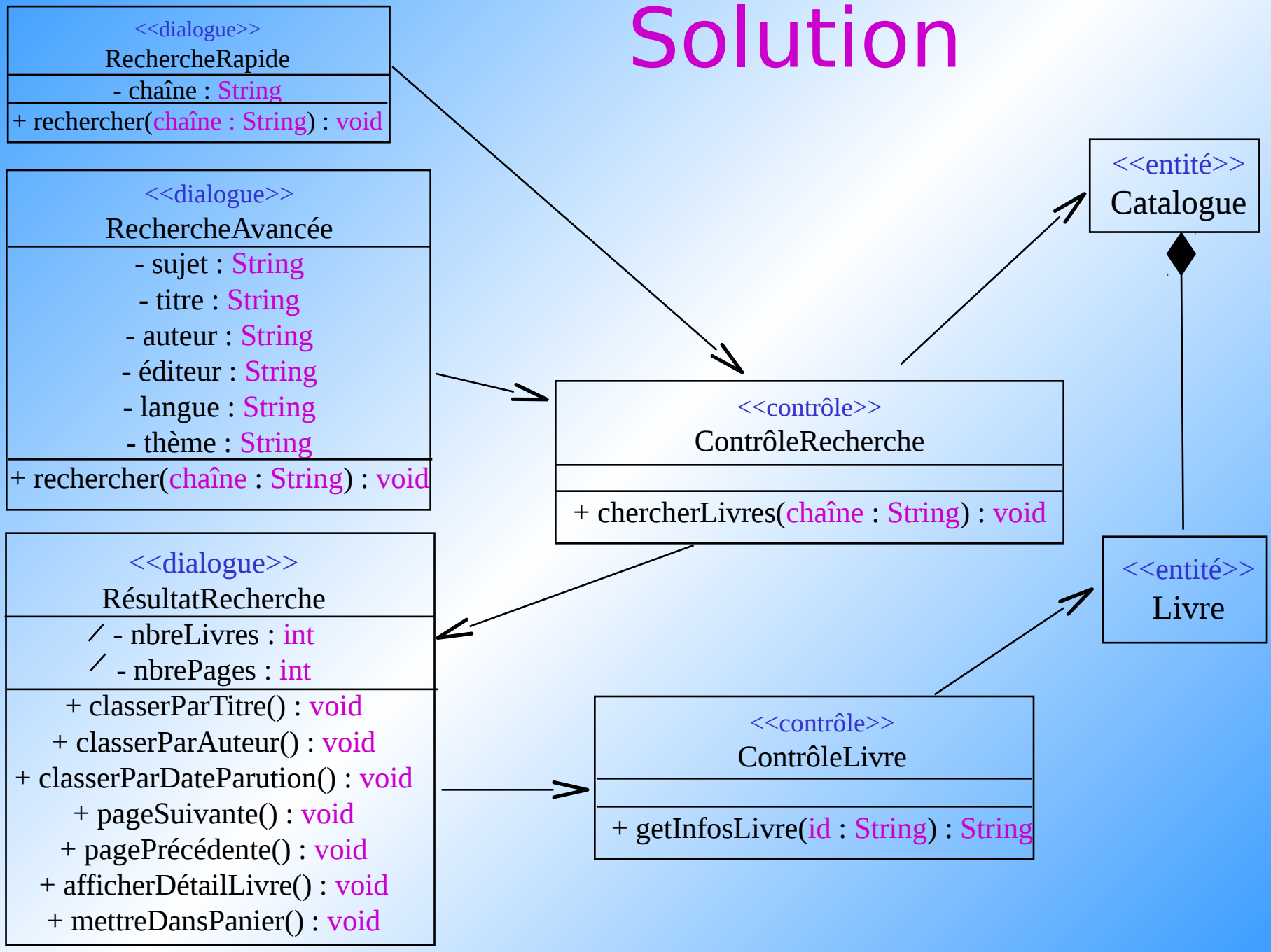
Solution

Faire le diagramme de classes de conception préliminaire

Rechercher des ouvrages

Ajouter les types et indiquer si les attributs et les méthodes sont publiques ou privés

Solution


<<dialogue>>
RechercheRapide
- chaîne : String
+ rechercher(chaîne : String) : void
<<dialogue>>
RechercheAvancée
- sujet : String
- titre : String
- auteur : String
- éditeur : String
- langue : String
- thème : String
+ rechercher(chaîne : String) : void
<<dialogue>>
RésultatRecherche
/ - nbreLivres : int
/ - nbrePages : int
+ classerParTitre() : void
+ classerParAuteur() : void
+ classerParDateParution() : void
+ pageSuivante() : void
+ pagePrécédente() : void
+ afficherDétailLivre() : void
+ mettreDansPanier() : void
<<contrôle>>
ContrôleRecherche
+ chercherLivres(chaîne : String) : void
<<contrôle>>
ContrôleLivre
+ getInfosLivre(id : String) : String
<<entité>>
Catalogue
<<entité>>
Livre

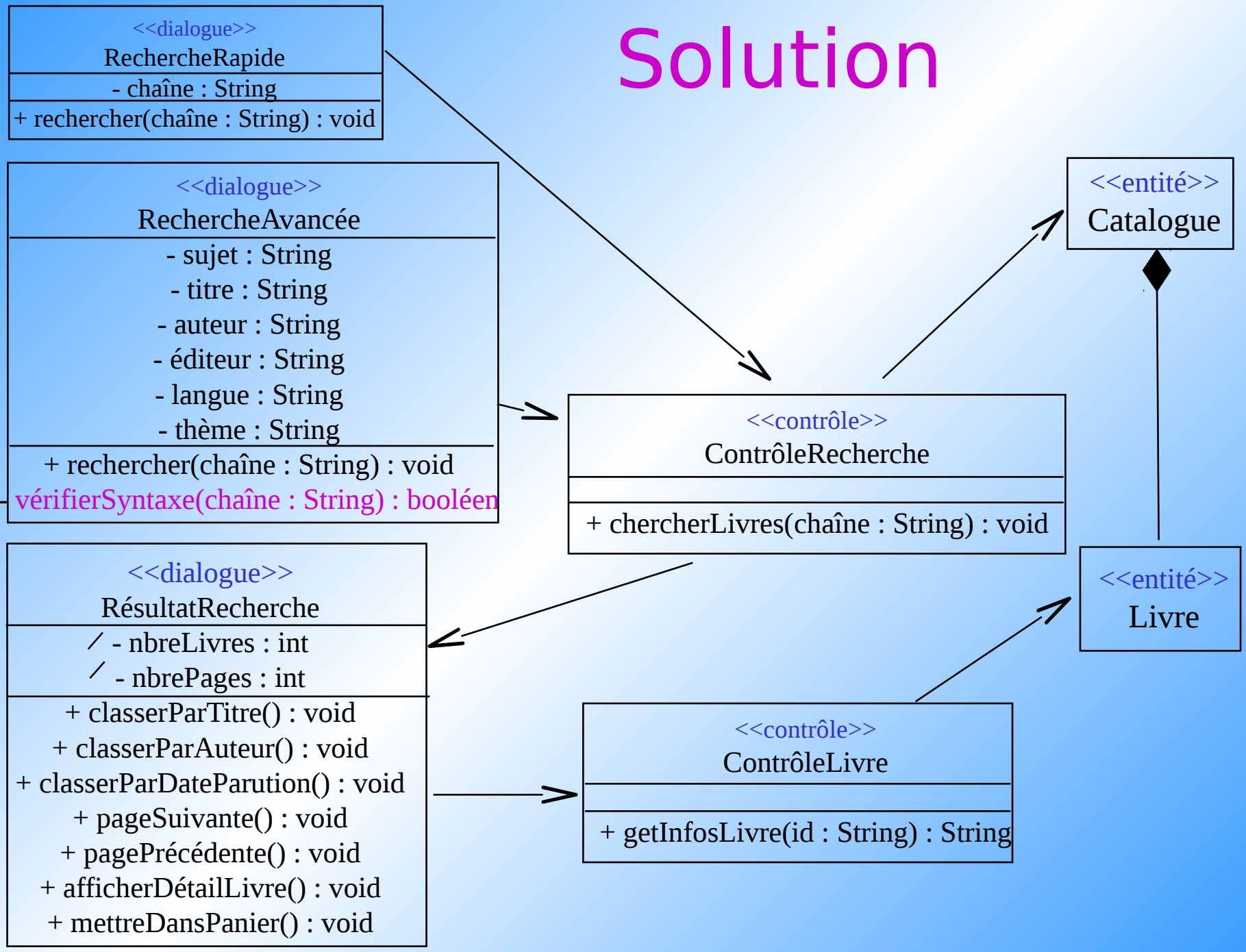
Solution

Faire le diagramme de classes de conception préliminaire

Rechercher des ouvrages

Ajouter les nouvelles méthodes

Solution


<<dialogue>>
RechercheRapide
- chaîne : String
+ rechercher(chaîne : String) : void
<<dialogue>>
RechercheAvancée
- sujet : String
- titre : String
- auteur : String
- éditeur : String
- langue : String
- thème : String
+ rechercher(chaîne : String) : void
- vérifierSyntaxe(chaîne : String) : booléen
<<contrôle>>
ContrôleRecherche
+ chercherLivres(chaîne : String) : void
<<entité>>
Catalogue
<<entité>>
Livre
<<dialogue>>
RésultatRecherche
/ - nbreLivres : int
/ - nbrePages : int
+ classerParTitre() : void
+ classerParAuteur() : void
+ classerParDateParution() : void
+ pageSuivante() : void
+ pagePrécédente() : void
+ afficherDétailLivre() : void
+ mettreDansPanier() : void
<<contrôle>>
ContrôleLivre
+ getInfosLivre(id : String) : String

Solution

Qu'avons-nous appris de nouveau sur ces classes ?

- Un certain nombre d'opérations sont à ajouter aux classes existantes :

  - *verifierSyntaxeRecherche(phrase)* à la classe dialogue *RechercheAvancee*.

  - les opérations importantes dans les classes entités.

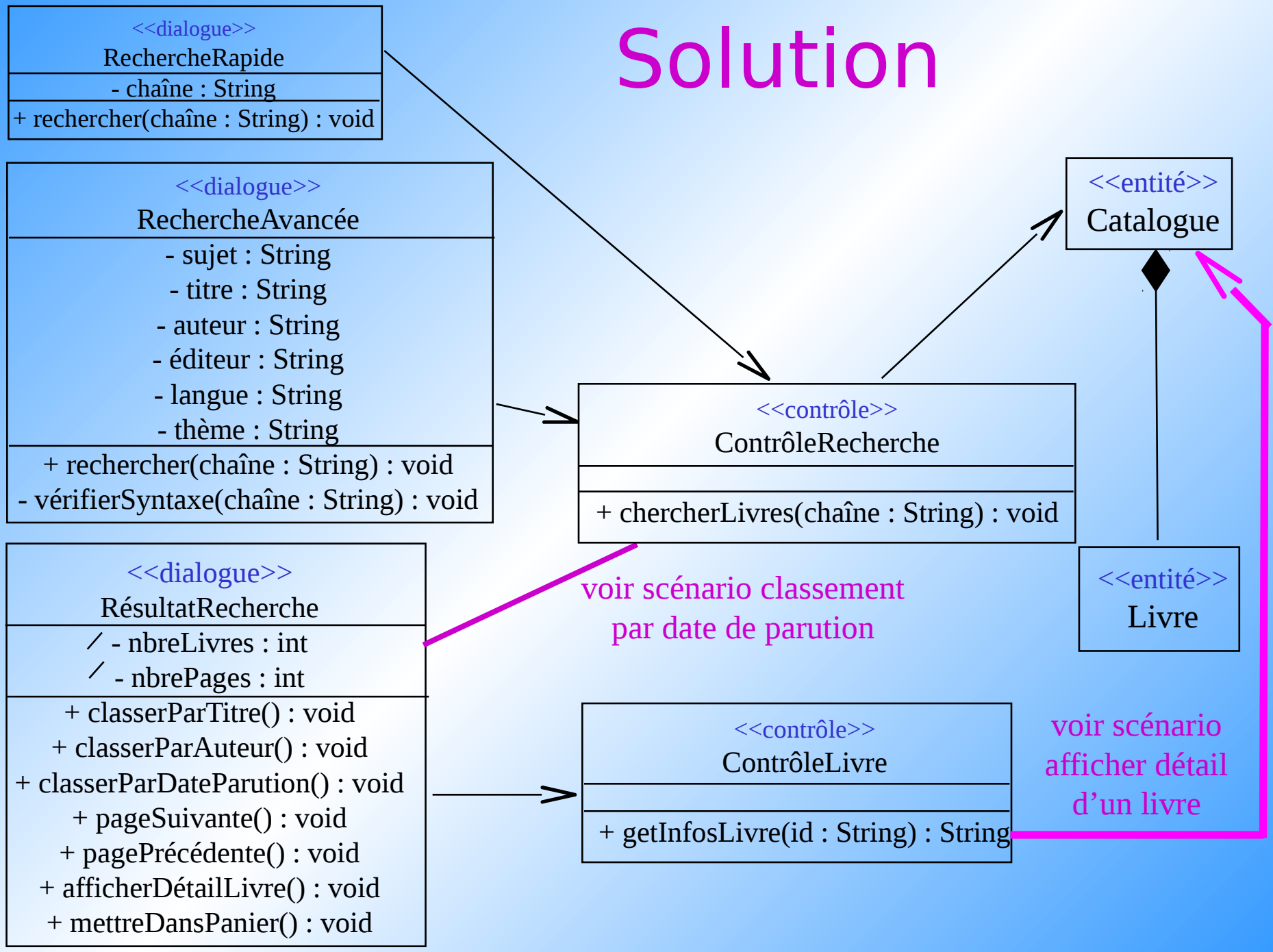
Solution

Faire le diagramme de classes de conception préliminaire

Rechercher des ouvrages

Vérifier la navigabilité

Solution


<<dialogue>>
RechercheRapide
- chaîne : String
+ rechercher(chaîne : String) : void
<<dialogue>>
RechercheAvancée
- sujet : String
- titre : String
- auteur : String
- éditeur : String
- langue : String
- thème : String
+ rechercher(chaîne : String) : void
- vérifierSyntaxe(chaîne : String) : void
<<dialogue>>
RésultatRecherche
- nbreLivres : int
- nbrePages : int
+ classerParTitre() : void
+ classerParAuteur() : void
+ classerParDateParution() : void
+ pageSuivante() : void
+ pagePrécédente() : void
+ afficherDétailLivre() : void
+ mettreDansPanier() : void
<<contrôle>>
ContrôleRecherche
+ chercherLivres(chaîne : String) : void
<<contrôle>>
ContrôleLivre
+ getInfosLivre(id : String) : String
<<entité>>
Catalogue
<<entité>>
Livre
voir scénario classement par date de parution
voir scénario afficher détail d'un livre

Solution

Qu'avons-nous appris de nouveau sur ces classes ?

- Les diagrammes d'interaction précisent que le contrôle *ControleFicheDétaillée* doit passer par l'entité *Catalogue* pour pouvoir accéder à un livre.
L'association existante doit donc être modifiée.

- Les sens de circulation des messages nous permettent de limiter la navigabilité de certaines associations entre entités.

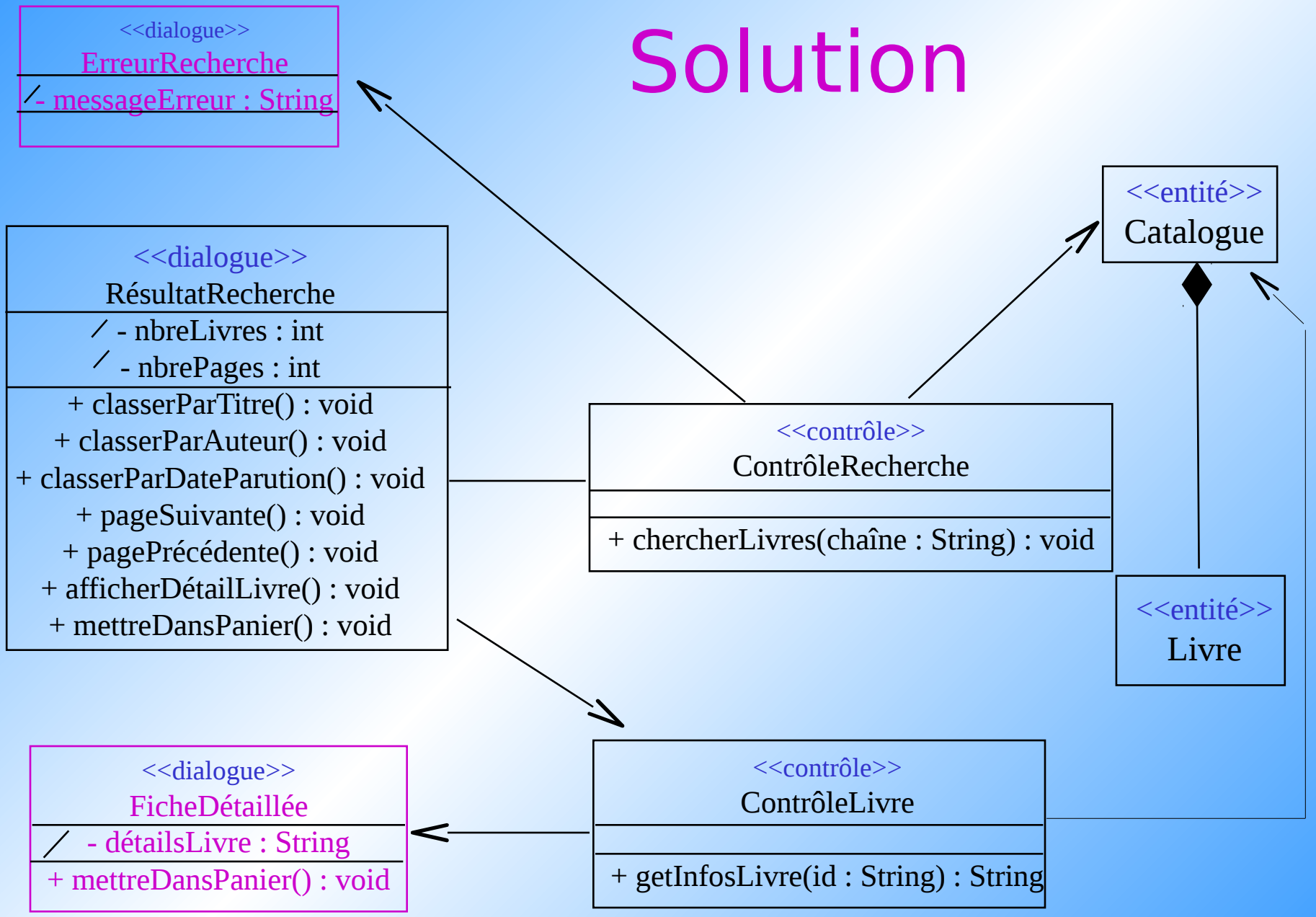
Solution

Faire le diagramme de classes de conception préliminaire

Rechercher des ouvrages

Rajouter des dialogues

Solution


<<dialogue>>
ErreurRecherche
- messageErreur : String
<<dialogue>>
RésultatRecherche
- nbreLivres : int
- nbrePages : int
+ classerParTitre() : void
+ classerParAuteur() : void
+ classerParDateParution() : void
+ pageSuivante() : void
+ pagePrécédente() : void
+ afficherDétailLivre() : void
+ mettreDansPanier() : void
<<contrôle>>
ContrôleRecherche
+ chercherLivres(chaîne : String) : void
<<entité>>
Catalogue
<<entité>>
Livre
<<dialogue>>
FicheDétaillée
- détailsLivre : String
+ mettreDansPanier() : void
<<contrôle>>
ContrôleLivre
+ getInfosLivre(id : String) : String

Solution

Qu'avons-nous appris de nouveau sur ces classes ?

- Il existe deux dialogues supplémentaires :

  - l'un correspondant à l'erreur *"aucun ouvrage trouvé"* : dialogue *ErreurRecherche*,

  - l'autre permettant d'afficher la page détaillée d'un ouvrage : dialogue *FicheDetaillee*.

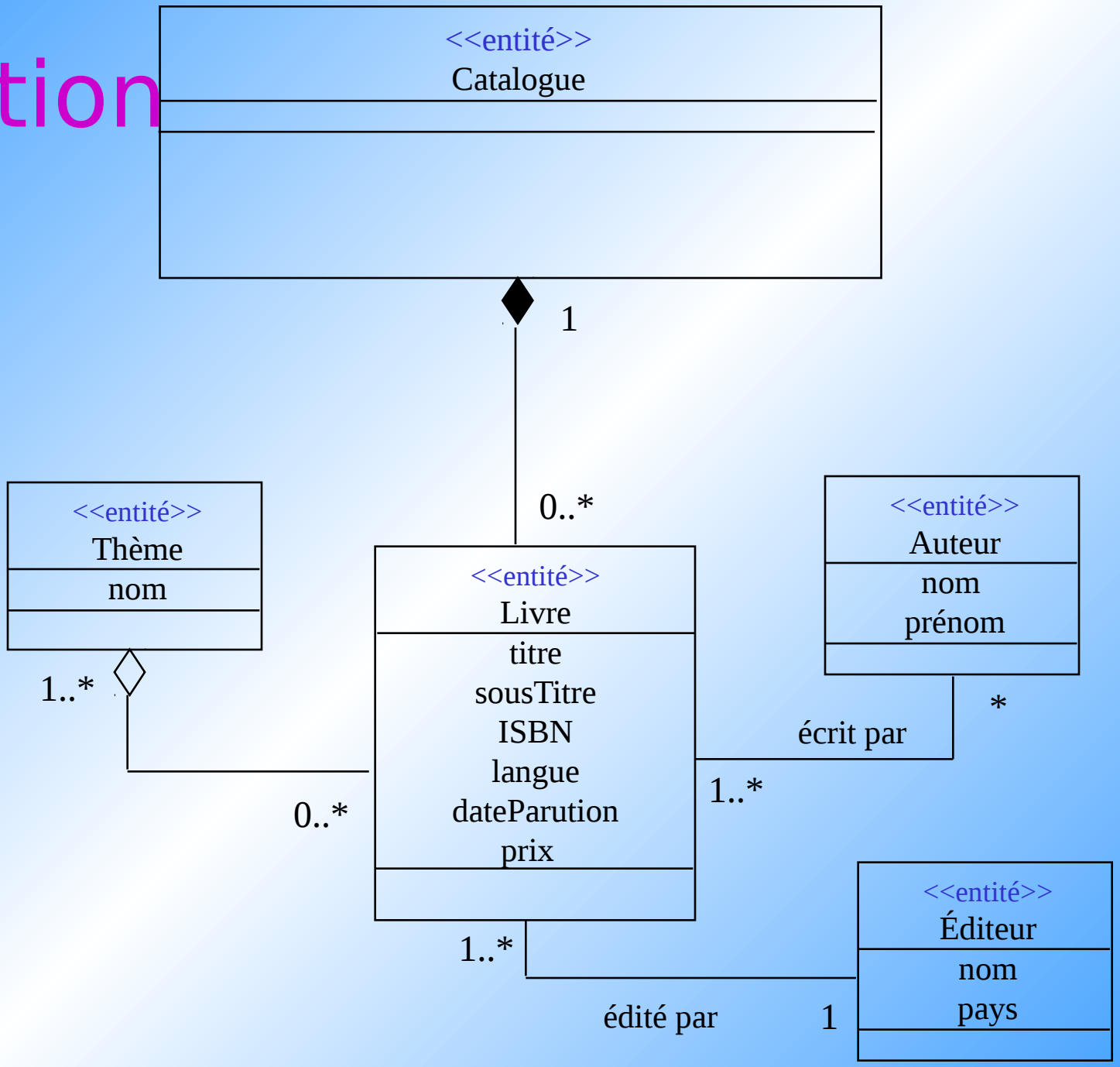
Solution

Faire le diagramme de classes de conception préliminaire

Rechercher des ouvrages

S’occuper des entités

Solution


<<entité>>
Catalogue
1
0..*
<<entité>>
Thème
nom
1..*
0..*
<<entité>>
Livre
titre
sousTitre
ISBN
langue
dateParution
prix
<<entité>>
Auteur
nom
prénom
*
écrit par
1..*
1..*
édité par
1
<<entité>>
Éditeur
nom
pays

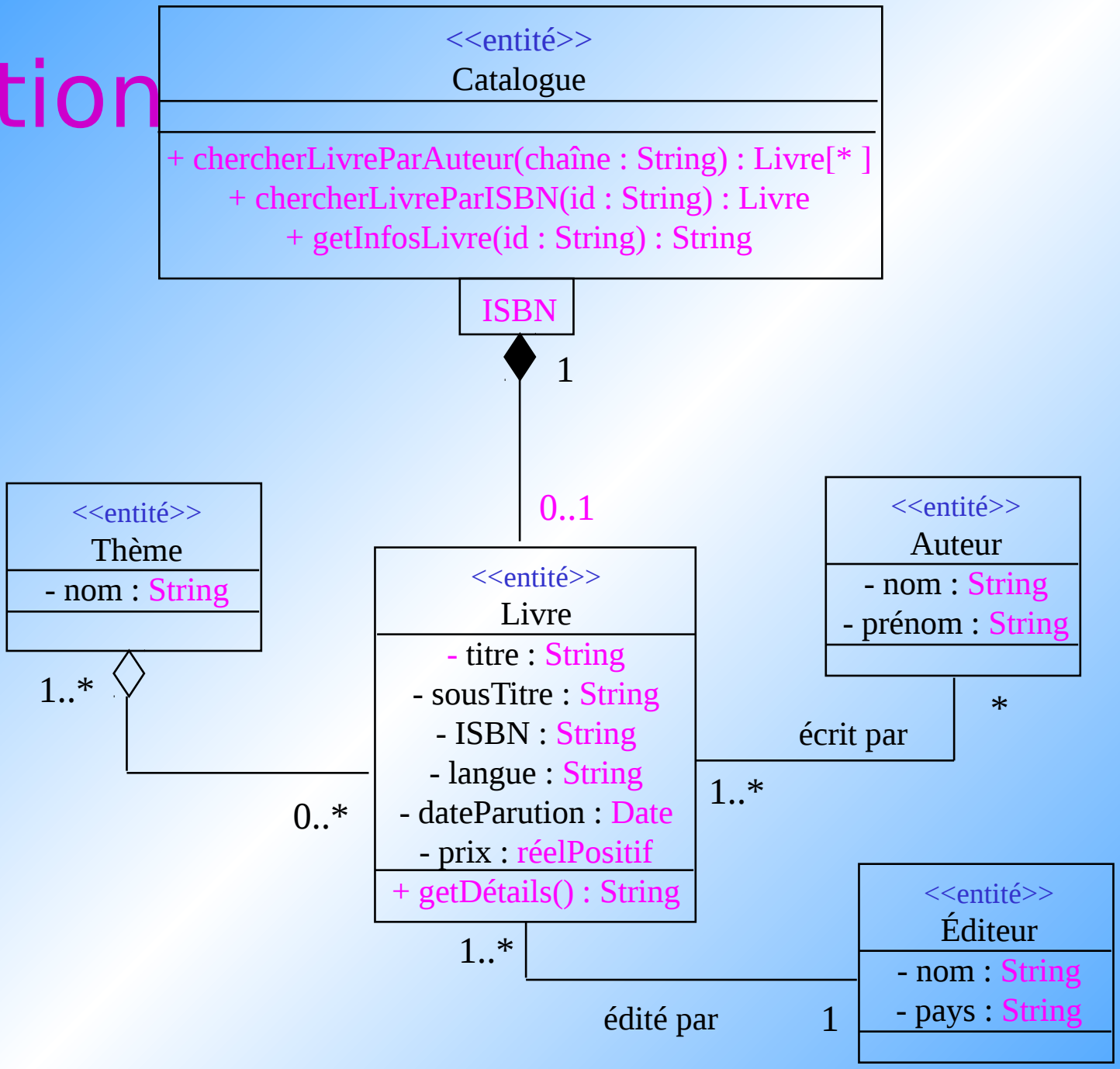
Solution

Faire le diagramme de classes de conception préliminaire

Rechercher des ouvrages

Ajouter les types et indiquer
si les attributs et les méthodes
sont publiques ou privés
et
Ajouter les nouvelles méthodes

Solution


<<entité>>
Catalogue
+ chercherLivreParAuteur(chaîne : String) : Livre[*]
+ chercherLivreParISBN(id : String) : Livre
+ getInfosLivre(id : String) : String
ISBN
1
0..1
<<entité>>
Thème
- nom : String
1..*
0..*
<<entité>>
Livre
- titre : String
- sousTitre : String
- ISBN : String
- langue : String
- dateParution : Date
- prix : réelPositif
+ getDétails() : String
<<entité>>
Auteur
- nom : String
- prénom : String
*
écrit par
1..*
1..*
édité par
1
<<entité>>
Éditeur
- nom : String
- pays : String

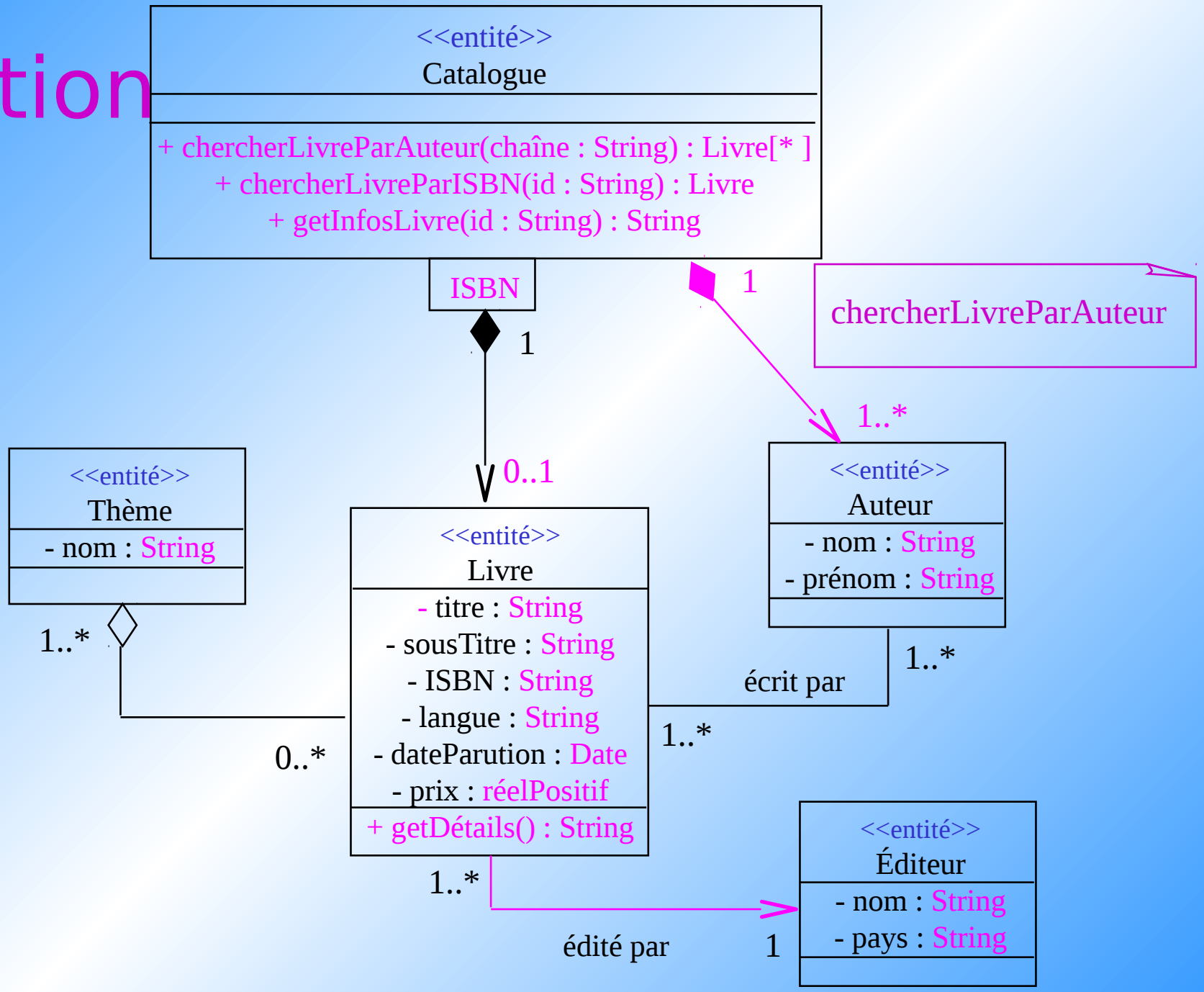
Solution

Faire le diagramme de classes de conception préliminaire

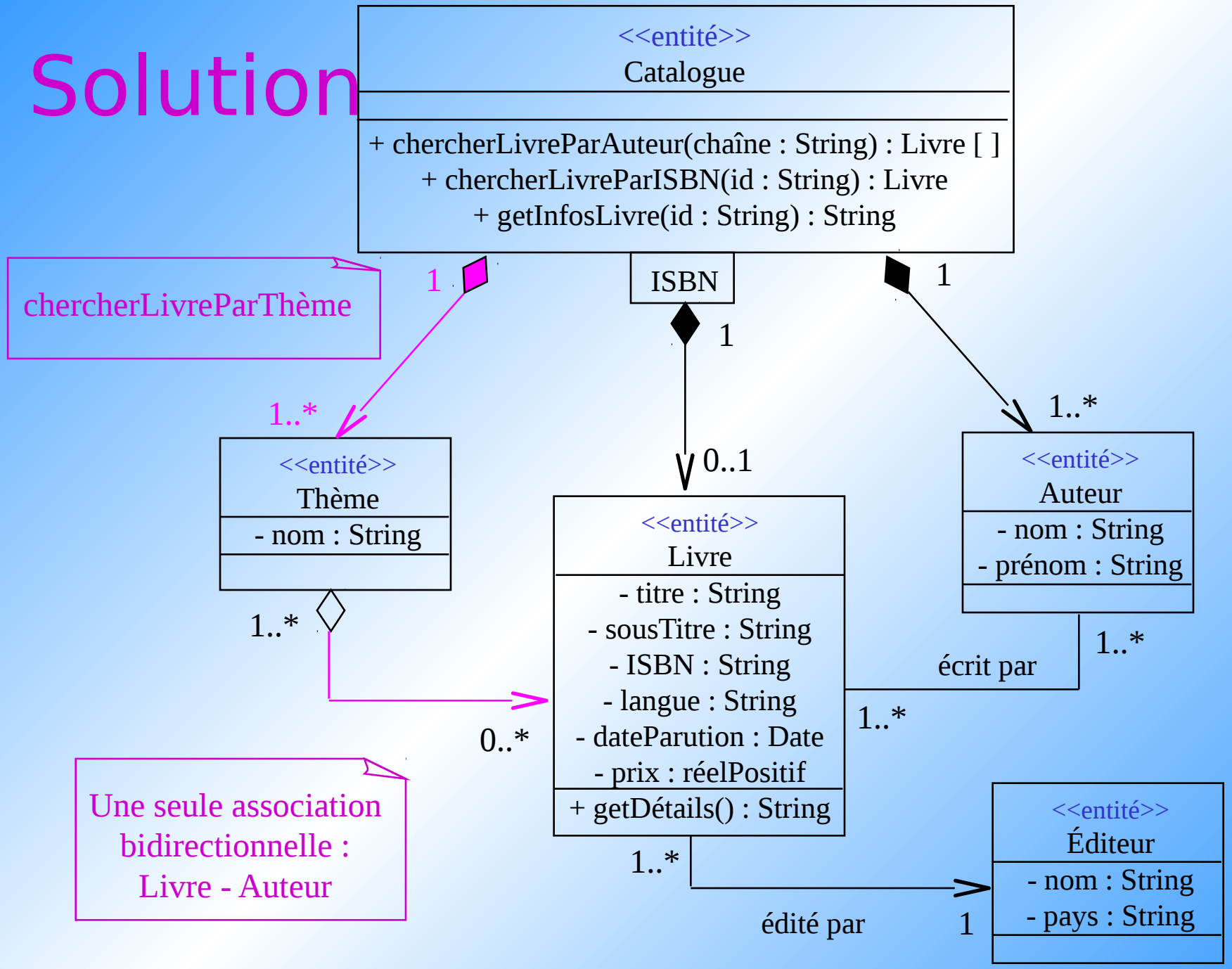
Rechercher des ouvrages

Vérifier la navigabilité

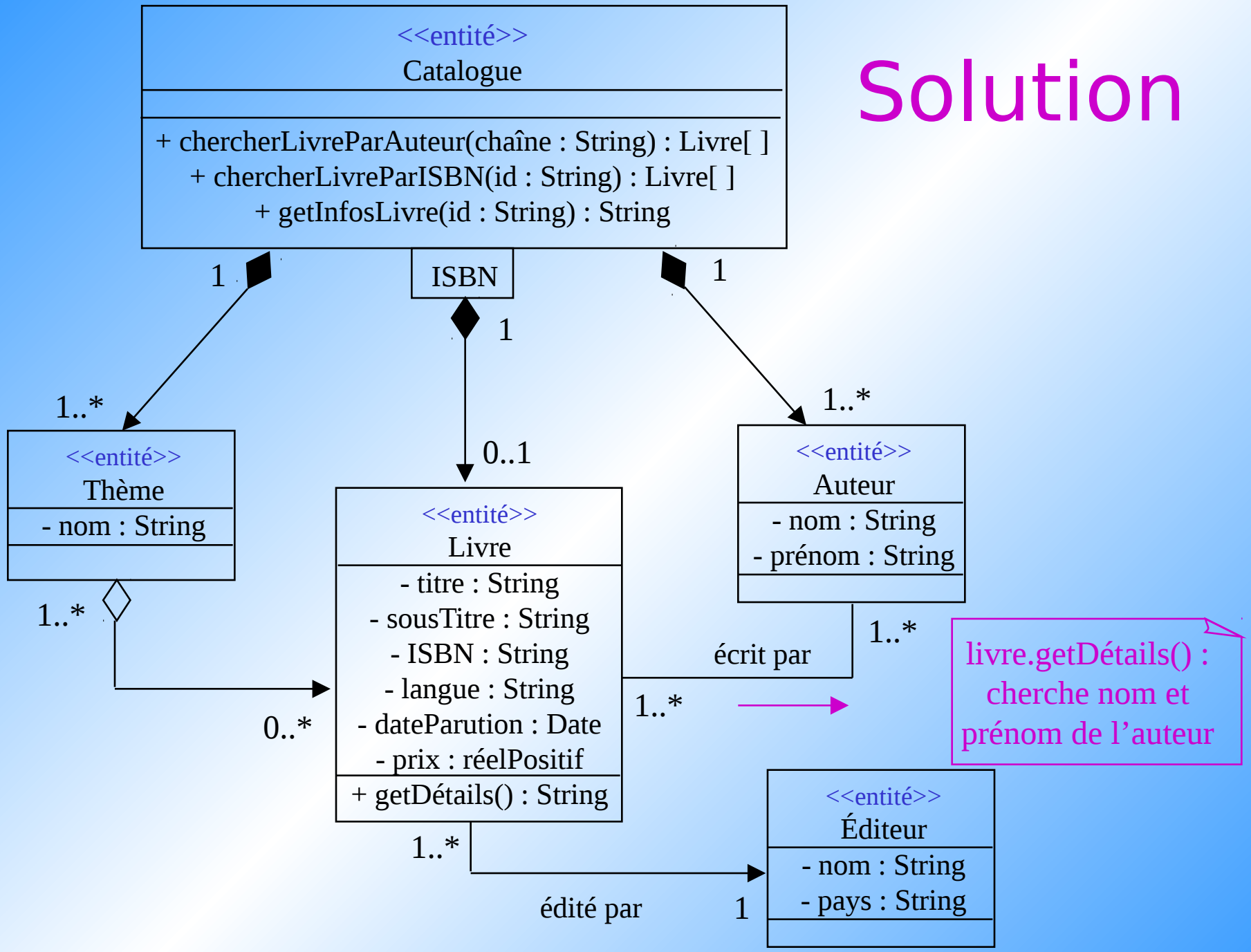
Solution


<<entité>>
Catalogue
+ chercherLivreParAuteur(chaîne : String) : Livre[*]
+ chercherLivreParISBN(id : String) : Livre
+ getInfosLivre(id : String) : String
ISBN
1
1
chercherLivreParAuteur
1..*
0..1
<<entité>>
Thème
- nom : String
1..*
<<entité>>
Livre
- titre : String
- sousTitre : String
- ISBN : String
- langue : String
- dateParution : Date
- prix : réelPositif
+ getDétails() : String
0..*
<<entité>>
Auteur
- nom : String
- prénom : String
1..*
écrit par
1..*
1..*
édité par
1
<<entité>>
Éditeur
- nom : String
- pays : String

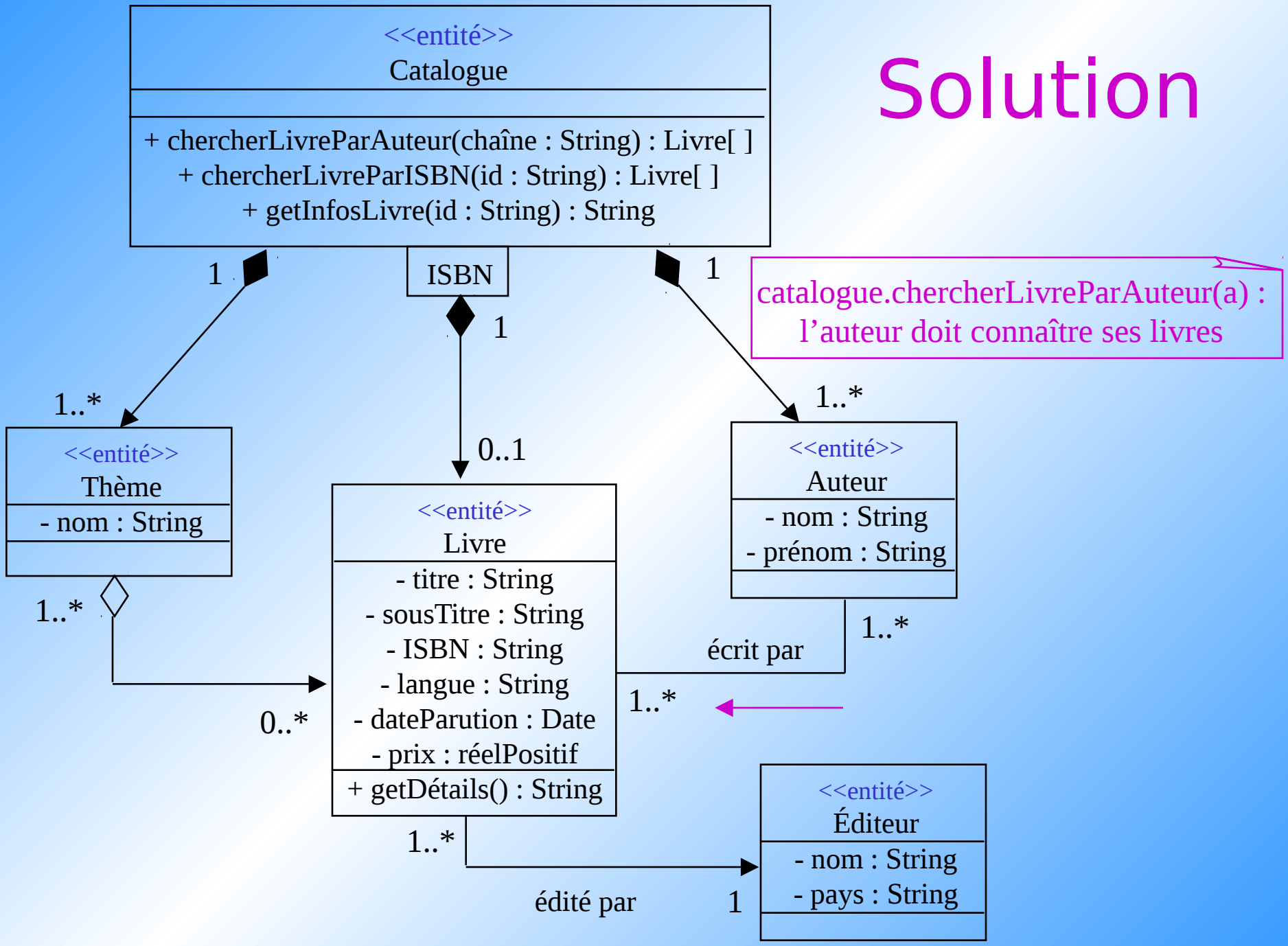
Solution


<<entité>>
Catalogue
+ chercherLivreParAuteur(chaîne : String) : Livre [ ]
+ chercherLivreParISBN(id : String) : Livre
+ getInfosLivre(id : String) : String
chercherLivreParThème
1
ISBN
1
1
1..*
0..1
1..*
<<entité>>
Thème
- nom : String
1..*
<<entité>>
Livre
- titre : String
- sousTitre : String
- ISBN : String
- langue : String
- dateParution : Date
- prix : réelPositif
+ getDétails() : String
<<entité>>
Auteur
- nom : String
- prénom : String
1..*
écrit par
1..*
0..*
1..*
édité par
1
<<entité>>
Éditeur
- nom : String
- pays : String
Une seule association bidirectionnelle : Livre - Auteur

Solution


<<entité>>
Catalogue
+ chercherLivreParAuteur(chaîne : String) : Livre[ ]
+ chercherLivreParISBN(id : String) : Livre[ ]
+ getInfosLivre(id : String) : String
ISBN
1
1
1
1..*
1..*
0..1
<<entité>>
Thème
- nom : String
<<entité>>
Livre
- titre : String
- sousTitre : String
- ISBN : String
- langue : String
- dateParution : Date
- prix : réelPositif
+ getDétails() : String
<<entité>>
Auteur
- nom : String
- prénom : String
1..*
0..*
écrit par
1..*
1..*
livre.getDétails() : cherche nom et prénom de l'auteur
1..*
édité par
1
<<entité>>
Éditeur
- nom : String
- pays : String

Solution


<<entité>>
Catalogue
+ chercherLivreParAuteur(chaîne : String) : Livre[ ]
+ chercherLivreParISBN(id : String) : Livre[ ]
+ getInfosLivre(id : String) : String
1
ISBN
1
1
catalogue.chercherLivreParAuteur(a) :
l'auteur doit connaître ses livres
1..*
0..1
1..*
<<entité>>
Thème
- nom : String
<<entité>>
Livre
- titre : String
- sousTitre : String
- ISBN : String
- langue : String
- dateParution : Date
- prix : réelPositif
+ getDétails() : String
<<entité>>
Auteur
- nom : String
- prénom : String
1..*
écrit par
1..*
1..*
0..*
1..*
édité par
1
<<entité>>
Éditeur
- nom : String
- pays : String

Classes de conception préliminaire

- Une seule association est restée bidirectionnelle : celle entre les entités *Livre* et *Auteur*.

- En effet l'opération *getDetails()* sur un livre va chercher le nom et le prénom de l'auteur. La classe *Livre* doit donc pouvoir naviguer vers la classe *Auteur*.

- Mais pour l'opération *chercherLivresParAuteur(a)* du *Catalogue*, on a gardé la possibilité de passer par l'objet auteur qui doit alors connaître ses livres.

- L'association est donc pour l'instant bidirectionnelle, mais un choix plus restrictif pourra être effectué en conception détaillée.

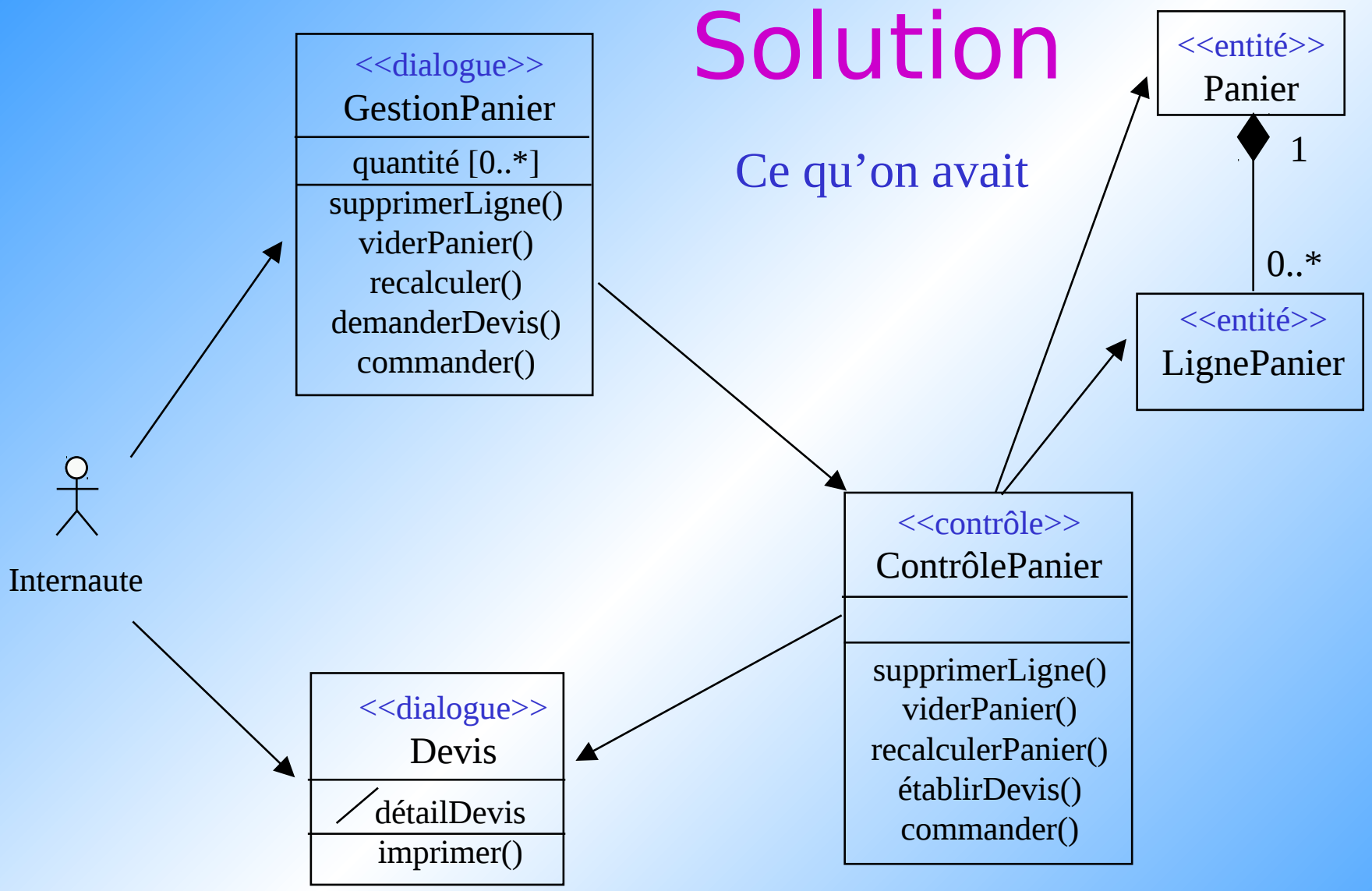
Classes de conception préliminaire

Faire le diagramme de classes de conception préliminaire

Gérer son panier

Solution

Ce qu'on avait



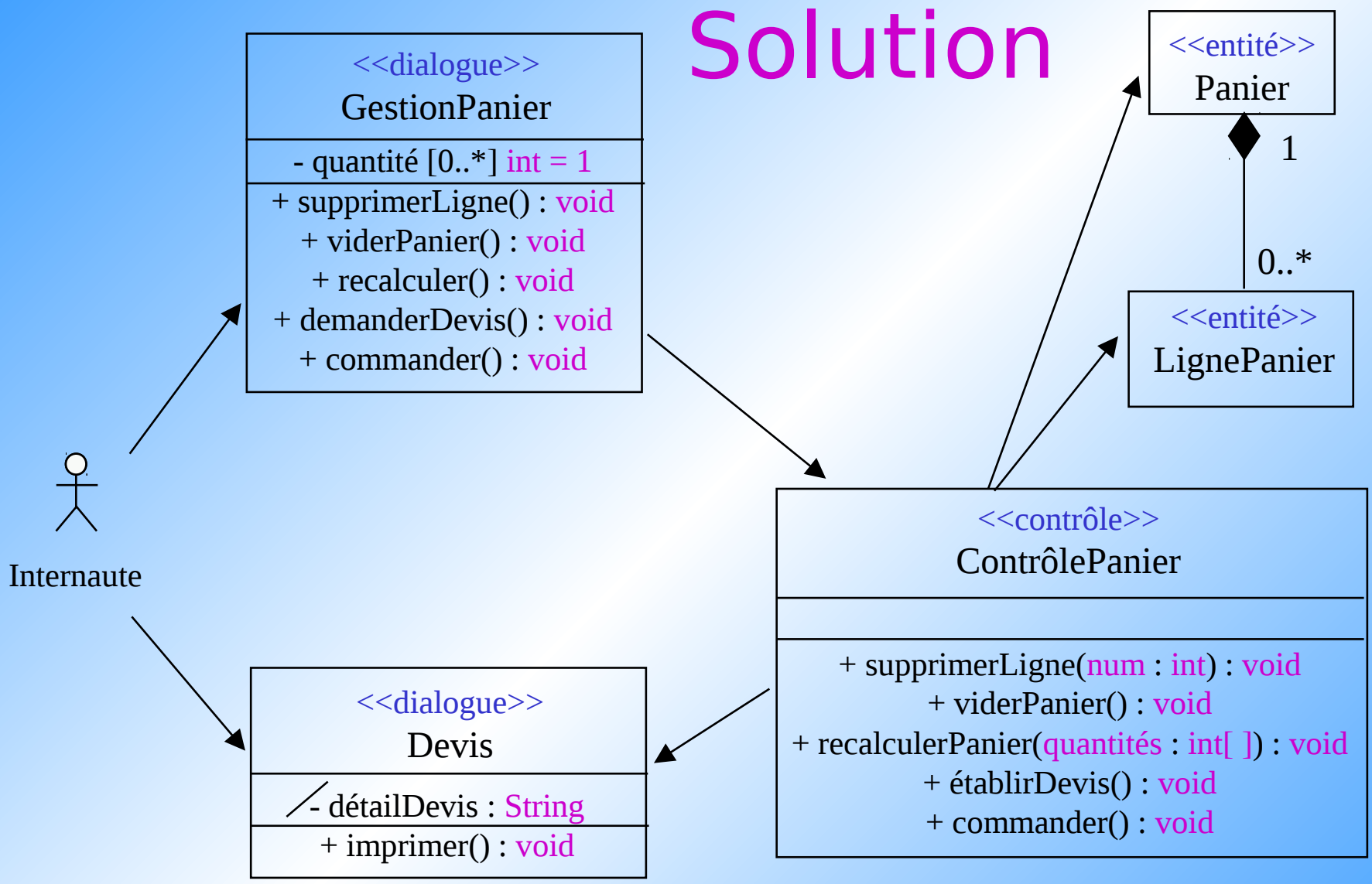
Solution

Faire le diagramme de classes de conception préliminaire

Gérer son panier

Ajouter les types et indiquer si les attributs et les méthodes sont publiques ou privés

Solution


<<dialogue>>
GestionPanier
- quantité [0..*] int = 1
+ supprimerLigne() : void
+ viderPanier() : void
+ recalculer() : void
+ demanderDevis() : void
+ commander() : void
<<entité>>
Panier
1
0..*
<<entité>>
LignePanier
Internaute
<<contrôle>>
ContrôlePanier
+ supprimerLigne(num : int) : void
+ viderPanier() : void
+ recalculerPanier(quantités : int[ ]) : void
+ établirDevis() : void
+ commander() : void
<<dialogue>>
Devis
/- détailDevis : String
+ imprimer() : void

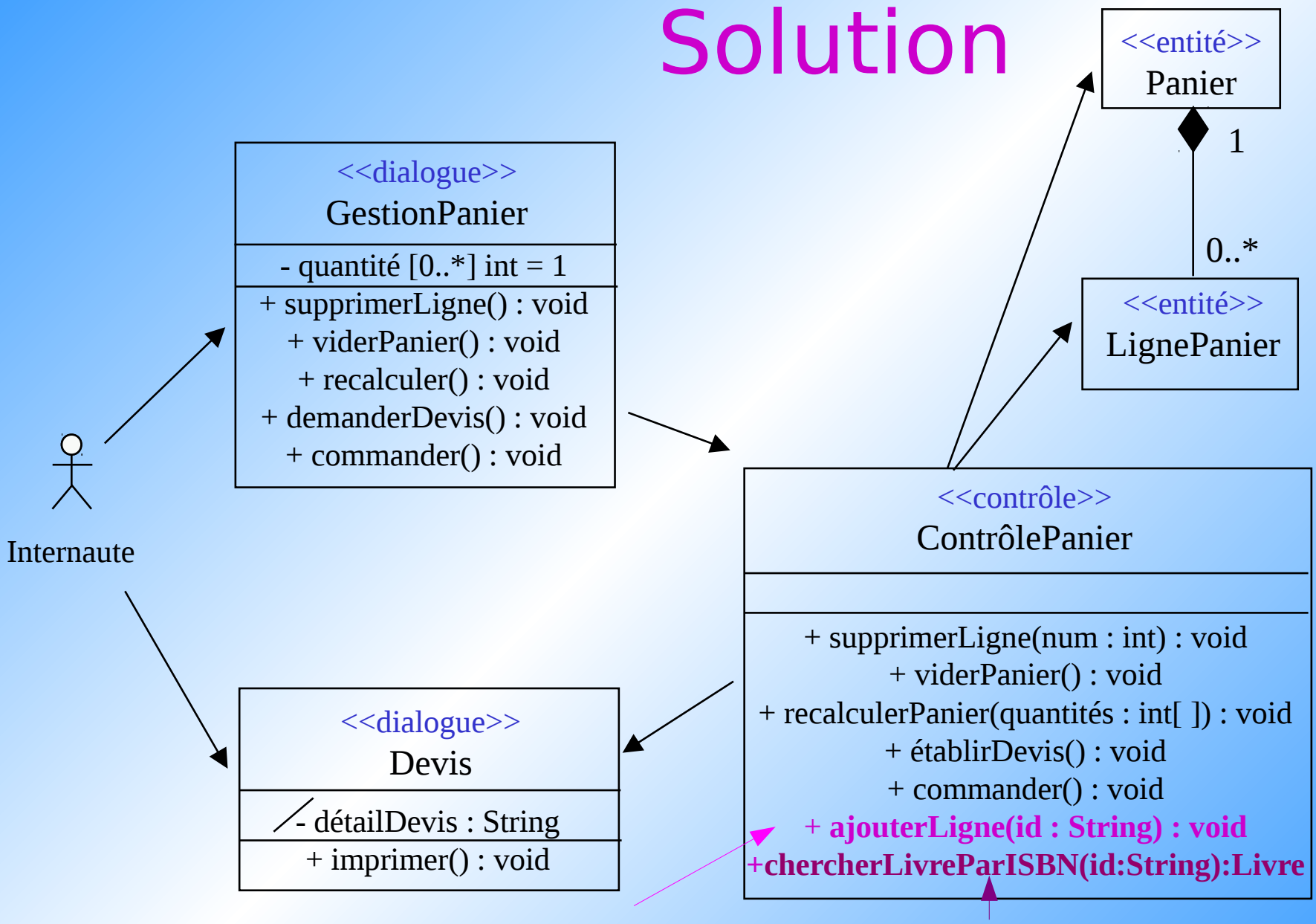
Solution

Faire le diagramme de classes de conception préliminaire

Gérer son panier

Ajouter les nouvelles méthodes

Solution



scénario ″l'internaute met 1 livre dans son panier″

scénario ″l'internaute met 1 premier livre dans son panier″

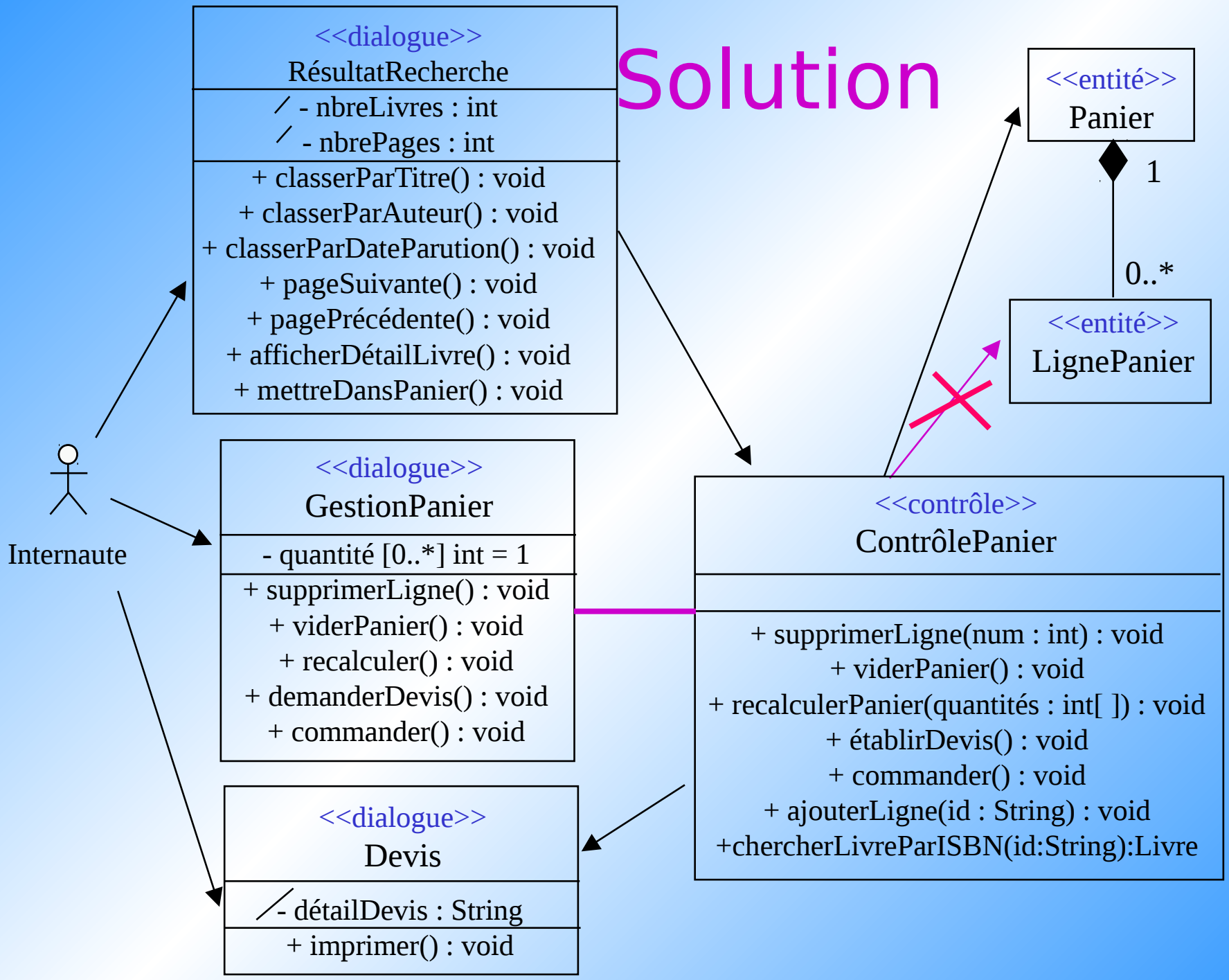
Solution

Faire le diagramme de classes de conception préliminaire

Gérer son panier

Vérifier la navigabilité

Solution


<<dialogue>>
RésultatRecherche
- nbreLivres : int
- nbrePages : int
+ classerParTitre() : void
+ classerParAuteur() : void
+ classerParDateParution() : void
+ pageSuivante() : void
+ pagePrécédente() : void
+ afficherDétailLivre() : void
+ mettreDansPanier() : void
<<entité>>
Panier
1
0..*
<<entité>>
LignePanier
Internaute
<<dialogue>>
GestionPanier
- quantité [0..*] int = 1
+ supprimerLigne() : void
+ viderPanier() : void
+ recalculer() : void
+ demanderDevis() : void
+ commander() : void
<<contrôle>>
ContrôlePanier
+ supprimerLigne(num : int) : void
+ viderPanier() : void
+ recalculerPanier(quantités : int[ ]) : void
+ établirDevis() : void
+ commander() : void
+ ajouterLigne(id : String) : void
+chercherLivreParISBN(id:String):Livre
<<dialogue>>
Devis
- détailDevis : String
+ imprimer() : void

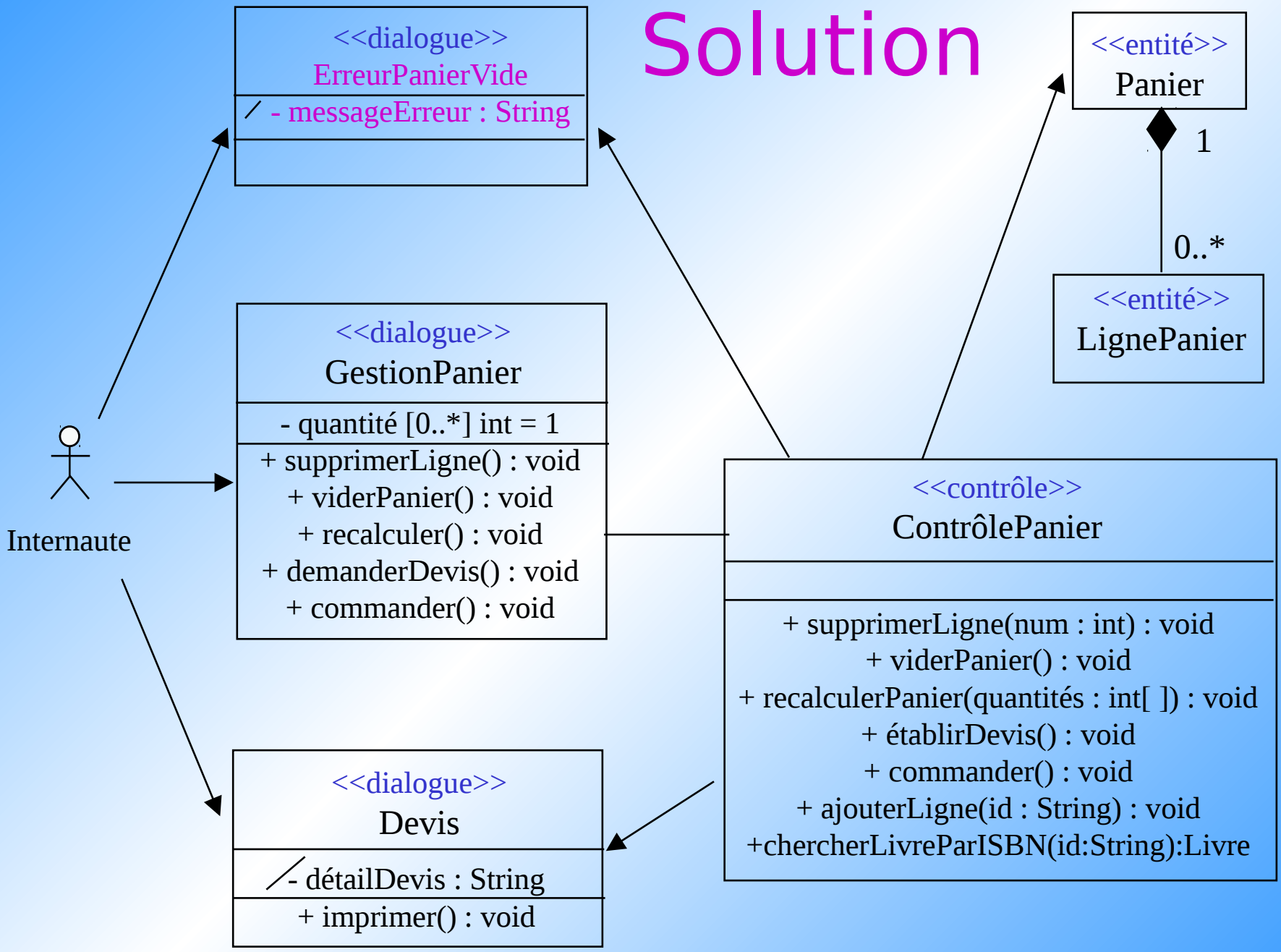
Solution

Faire le diagramme de classes de conception préliminaire

Gérer son panier

Rajouter des dialogues

Solution


<<dialogue>>
ErreurPanierVide
- messageErreur : String
<<entité>>
Panier
1
0..*
<<entité>>
LignePanier
<<dialogue>>
GestionPanier
- quantité [0..*] int = 1
+ supprimerLigne() : void
+ viderPanier() : void
+ recalculer() : void
+ demanderDevis() : void
+ commander() : void
Internaute
<<contrôle>>
ContrôlePanier
+ supprimerLigne(num : int) : void
+ viderPanier() : void
+ recalculerPanier(quantités : int[ ]) : void
+ établirDevis() : void
+ commander() : void
+ ajouterLigne(id : String) : void
+chercherLivreParISBN(id:String):Livre
<<dialogue>>
Devis
- détailDevis : String
+ imprimer() : void

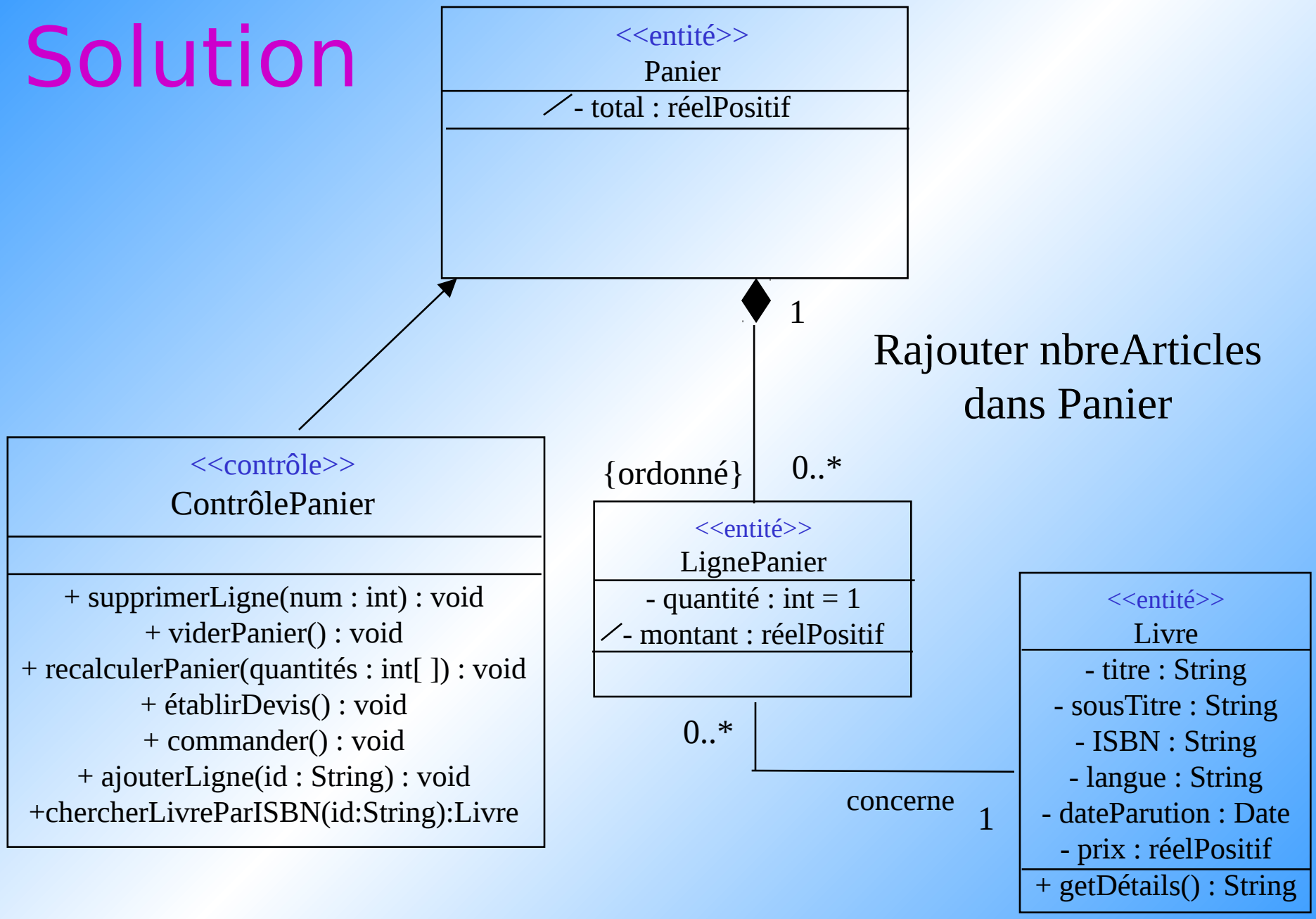
Solution

Faire le diagramme de classes de conception préliminaire

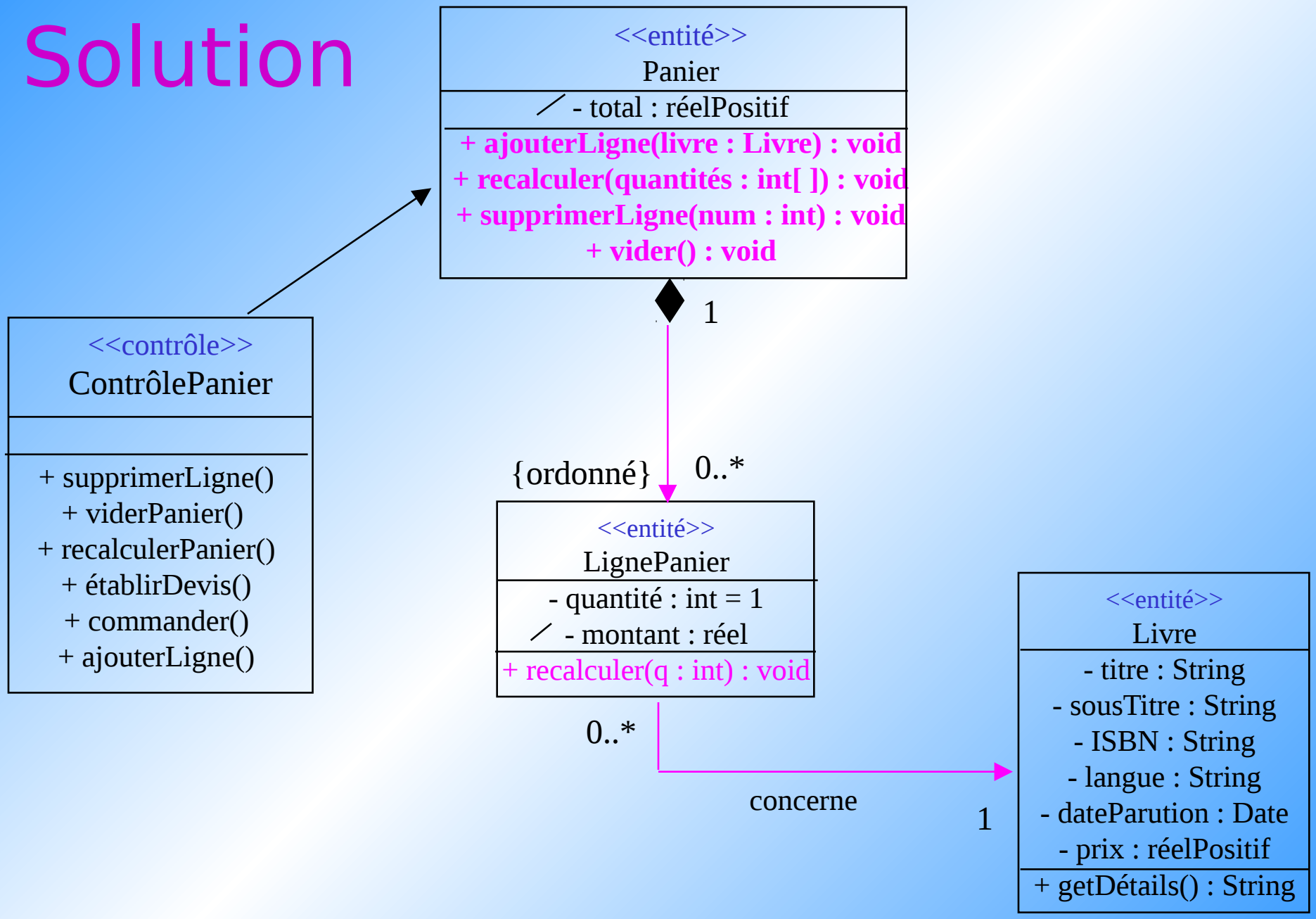
Gérer son panier

S’occuper des entités

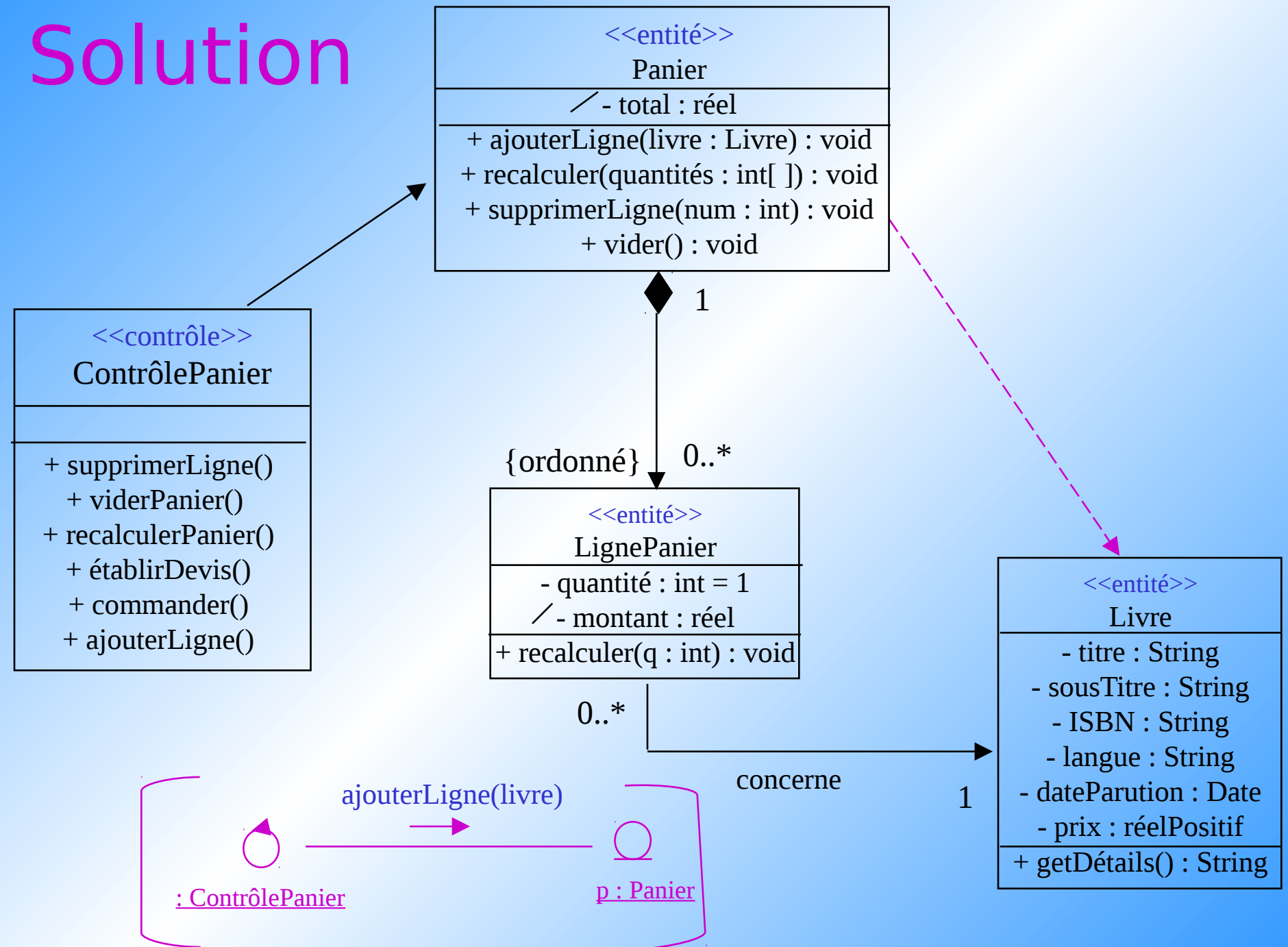
Solution


<<entité>>
Panier
/- total : réelPositif
1
Rajouter nbreArticles dans Panier
<<contrôle>>
ContrôlePanier
+ supprimerLigne(num : int) : void
+ viderPanier() : void
+ recalculerPanier(quantités : int[ ]) : void
+ établirDevis() : void
+ commander() : void
+ ajouterLigne(id : String) : void
+chercherLivreParISBN(id:String):Livre
{ordonné}
0..*
<<entité>>
LignePanier
- quantité : int = 1
/- montant : réelPositif
0..*
concerne
1
<<entité>>
Livre
- titre : String
- sousTitre : String
- ISBN : String
- langue : String
- dateParution : Date
- prix : réelPositif
+ getDétails() : String

Solution


<<entité>>
Panier
- total : réelPositif
+ ajouterLigne(livre : Livre) : void
+ recalculer(quantités : int[ ]) : void
+ supprimerLigne(num : int) : void
+ vider() : void
1
<<contrôle>>
ContrôlePanier
+ supprimerLigne()
+ viderPanier()
+ recalculerPanier()
+ établirDevis()
+ commander()
+ ajouterLigne()
{ordonné}
0..*
<<entité>>
LignePanier
- quantité : int = 1
- montant : réel
+ recalculer(q : int) : void
0..*
concerne
1
<<entité>>
Livre
- titre : String
- sousTitre : String
- ISBN : String
- langue : String
- dateParution : Date
- prix : réelPositif
+ getDétails() : String

Solution


<<entité>>
Panier
/ - total : réel
+ ajouterLigne(livre : Livre) : void
+ recalculer(quantités : int[ ]) : void
+ supprimerLigne(num : int) : void
+ vider() : void
1
<<contrôle>>
ContrôlePanier
+ supprimerLigne()
+ viderPanier()
+ recalculerPanier()
+ établirDevis()
+ commander()
+ ajouterLigne()
{ordonné}
0..*
<<entité>>
LignePanier
- quantité : int = 1
/ - montant : réel
+ recalculer(q : int) : void
0..*
concerne
1
<<entité>>
Livre
- titre : String
- sousTitre : String
- ISBN : String
- langue : String
- dateParution : Date
- prix : réelPositif
+ getDétails() : String
ajouterLigne(livre)
: ContrôlePanier
p : Panier

Classes de conception préliminaire

Qu'avons-nous appris de nouveau sur ces classes ?

- La classe *GestionPanier* possède une méthode *afficher(p:Panier)* qui induit une dépendance avec la classe *Panier*.

ute pour de nouvelles aventu