Cahier des charges pour le projet AL2000

Réalisé par : DEL-GROSSO Rémy, DERGUINI Mohamed, LOMBARD Damien, SIMON Dorian, VANDERSTEENE Alexis
Groupe 1-A

I- Descriptif du projet

Notre projet est de réaliser un logiciel embarqué pour un automate de distribution de films. Ce logiciel sera articulé autour de trois axes : l'IHM, le cœur métier et la base de données. La partie IHM permettra au client d'emprunter un film (qu'il a déjà choisi) en moins de 5 minutes).

Ce projet est composé de plusieurs rendus. Le premier livrable se compose du cahier des charges et des diagrammes UML des classes qui composeront notre cœur métier. Ce rendu est attendu le premier novembre. Le second livrable sera rendu le 15 novembre. Celui-ci se compose du programme dans son intégralité. Enfin le 19 novembre une soutenance aura lieu. Cette soutenance s'accompagne d'une démonstration.

II.Contexte du projet

Exposé de la situation

AL2000 est un distributeur autonome de films de CyberVideo. L'entreprise cherche à étendre la portée de son service de location et à agrandir le cercle de ses clients, en devenant accessible dans différents points stratégiques grâce à cet automate. Le distributeur devra être autonome afin de réduire les coûts et d'offrir aux clients la possibilité de gérer leur emprunts de la manière qui leur convient.

Objectifs

Tout d'abord, notre interface doit être facile à utiliser, c'est-à-dire que toute catégorie de personne doit pouvoir louer un film en un minimum de clics et en moins de 5 minutes en considérant que le client connaît le film qu'il souhaite louer.

Il est important que le système soit en marche 24h/24h, ça offre la possibilité au client de louer un film quand l'envie lui vient.

Nous offrons la possibilité de louer un film sous support non physique (QR code), pour que le client n'ait plus à penser quand rendre sa location et proposer un plus large choix de films.

Ces démarches encourageront nos clients à utiliser l'AL2000 parce qu'ils ressentiront de la facilité à le faire quand et où ils voudront. Mettre le client à l'aise avec AL2000 nous permettra aussi de le laisser gérer ses locations graĉe à un programme de fidélisation et un compte abonné, qui lui fera profiter de tarifs spéciaux, et d'une gestion de locations plus confortable. Avec son compte d'abonné il pourra créer et ajouter d'autres comptes pour le reste de sa famille, et pourra les configurer en mettant des restrictions protégeant les enfants.

Le système doit être maintenable à distance, ça limitera le déplacement du technicien uniquement aux cas graves et lors de la recharge des blu-ray sur le distributeur.

III. Description fonctionnelle des besoins

Page d’accueil

Par défaut lorsque la machine est en fonctionnement, un écran d’attente doit être affiché contenant différentes publicités et propositions de réductions. Dès que l’on détecte une action sur l'écran, nous débutons les étapes pour traiter les demandes du client.

Fonctionnalités

1. Consulter une liste de films
   Un client, qu'il soit abonné ou non, doit pouvoir consulter la liste des films disponibles à la fois dans la machine sous forme de blu-ray mais aussi présent en dématérialisé sur la base de données de l’entreprise. La liste sera filtrée par les restrictions stockées sur la carte de l’abonné.

   1.1. Consulter un film en particulier
   Parmi la liste de film précédente, ce client doit pouvoir consulter un film en particulier afin d’obtenir certaines informations sur le film. Ces informations sont : une description du film, les acteurs principaux, le réalisateur, les thèmes associés au film et l'âge recommandé.

   1.2. Filtrer la liste
   Ce client doit pouvoir entrer des critères permettant de filtrer les films affichés dans la liste. Ces critères doivent porter au minimum sur le thème du film, son réalisateur, son âge recommandé, ses acteurs principaux. Tous les critères peuvent être cumulés.

2. Louer un film
   Le solde sur la carte de l’adhérent qui souhaite emprunter un film doit être au minimum suffisant pour payer le premier jour de la location.

   2.1. Louer un film en blu-ray
   Si le film qu’un utilisateur consulte est présent sous forme de blu-ray il peut l’emprunter pour une journée minimum et une durée indéterminée.
   Si l’utilisateur est un client (non abonné), il ne peut en emprunter qu’un seul, sinon il est limité à 3.

   2.2. Louer un film sur QR-Code
   Tous les films affichés dans la liste peuvent être loués pour 12h sous la forme d’un QR-Code.

3. Rendre un film
   Les films empruntés sous le format d’un QR-Code sont automatiquement invalidés au bout de 12h.

3.1. Rendre un blu-ray

Au moment du rendu du DVD, la machine vérifie d'abord que le DVD correspond bien à un emprunt effectué par le client.
Le client peut prévenir au moment du rendu si le DVD est endommagé.

4. Sauvegarder les transactions

Tous les emprunts doivent être sauvegardés indépendamment du fait qu'ils aient été réalisés par des clients ou des adhérents. Chaque transaction doit être conservée pendant 1 an minimum.

4.1. Consulter l'historique

Les adhérents doivent pouvoir accéder à leur historique.

5. Créditer une carte abonnement

Cette fonctionnalité n'est disponible que pour les adhérents au service.

5.1. S'il s'agit de la carte principale d'un adhérent

Le montant est ajouté directement à la carte en question

5.2. S'il s'agit d'une autre carte liée à un adhérent

Si l'adhérent souhaite créditer une autre carte d'abonnement qui lui est liée, il doit d'abord la sélectionner puis effectuer la même opération que la 5.1.

6. S'abonner

Un client doit pouvoir s'il le souhaite, s'abonner au service et devenir adhérent à condition qu'il ait plus de 13 ans. Cela se traduit par la création d'une carte d'abonnement offrant des avantages. Dans ces avantages nous retrouvons, l'augmentation du nombre limite d'emprunts de DVD, une baisse du prix des locations de 5 à 4 € et une location offerte si 20 films ont été empruntés en un mois, qu'ils soient en blu-ray ou en QR-code.

6.1. Pour d'autres personnes

La première fois qu'un client s'abonne, nous considérons que sa carte sera la principale. Etant donné qu'un adhérent peut également s'abonner au service pour d'autres personnes et que chaque carte créée est liée au compte principal, cette carte principale lui permettra de modifier les restrictions et le soldes des tous les abonnements liés.

7. Modifier un abonnement

Un adhérent à une carte principale et éventuellement un nombre n de carte d'abonnement qui lui sont liées. Chaque carte doit pouvoir permettre d'ajouter ou de supprimer des restrictions (enfant par exemple), des filtres et de gérer le solde.

8. Payer

Dans le cas d'une location d'un film blu-ray, le paiement s'effectue au moment du rendu du DVD sauf si le DVD a été emprunté pendant une durée de plus d'un mois. A chaque mois écoulé pendant la durée de l'emprunt du DVD, le client sera

automatiquement débité pour le mois et paiera le reste au moment du rendu. Pour un QR-code le paiement s'effectue directement à l'emprunt.

8.1. Pour un client
Un client paie la facture par carte bancaire.

8.2. Pour un adhérent
Après avoir calculé les éventuelles réductions, il est possible de payer le montant de la location à partir du solde présent sur la carte d'adhérent.
Cette réduction consiste en une location gratuite lorsque celle-ci est la 20ème réalisée en un mois.

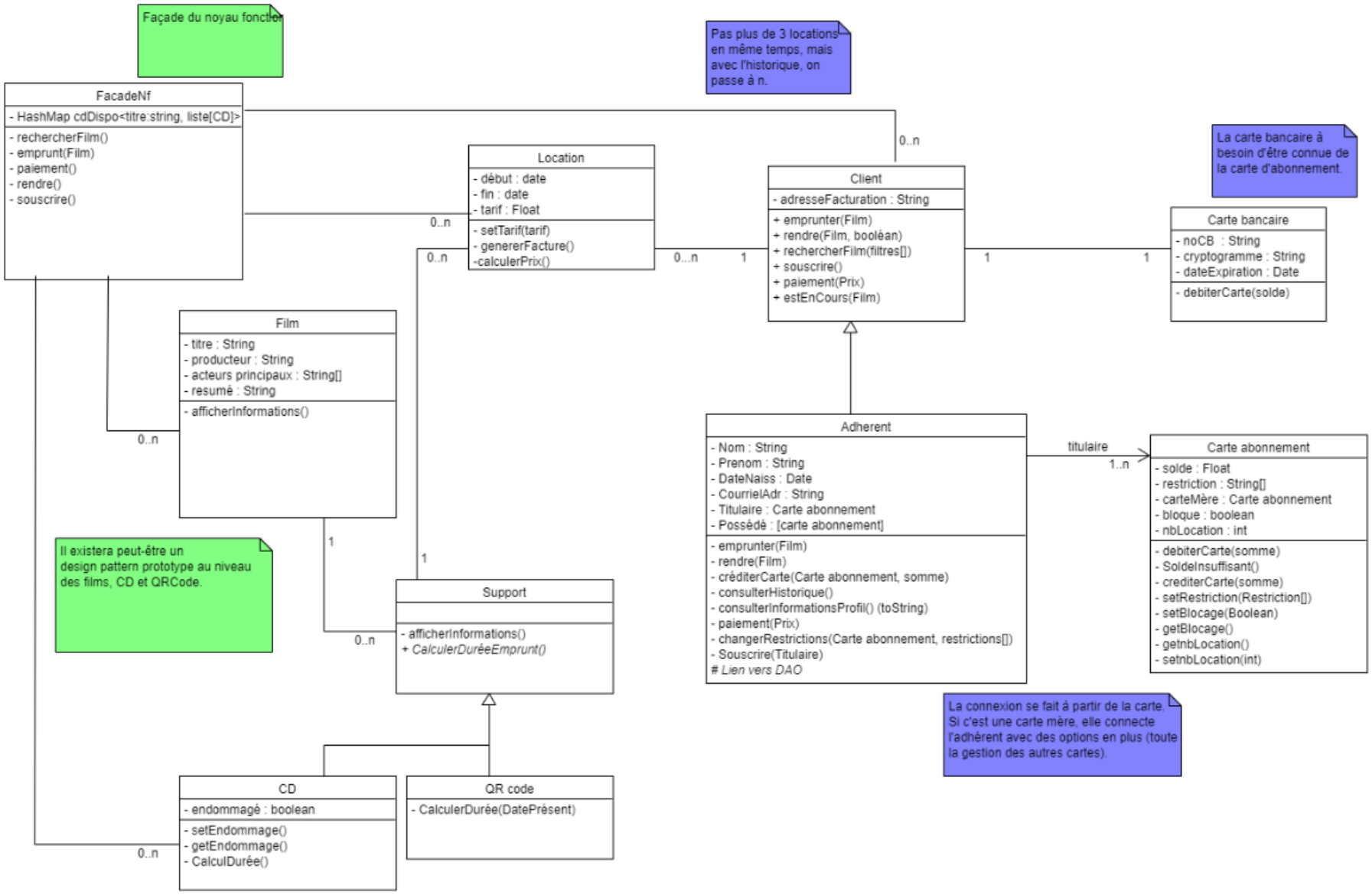
9. Gérer les blu-ray endommagés
Au moment du retour d'un DVD, le client a la possibilité de prévenir si le DVD est endommagé. Dans ce cas, le client est dédommagé et le DVD ne doit plus être proposé à la location jusqu'à ce qu'il soit récupéré par les techniciens.

IV Diagrammes UML

Description globale

La classe principale se compose d'une façade qui fait l'interface entre les fonctionnalités du cœur métier, l'IHM et la base de données. De plus, cette classe possède une hashMap qui permet de connaître les films disponibles. Enfin, depuis cette façade nous pouvons appeler les fonctions principales de notre programme.

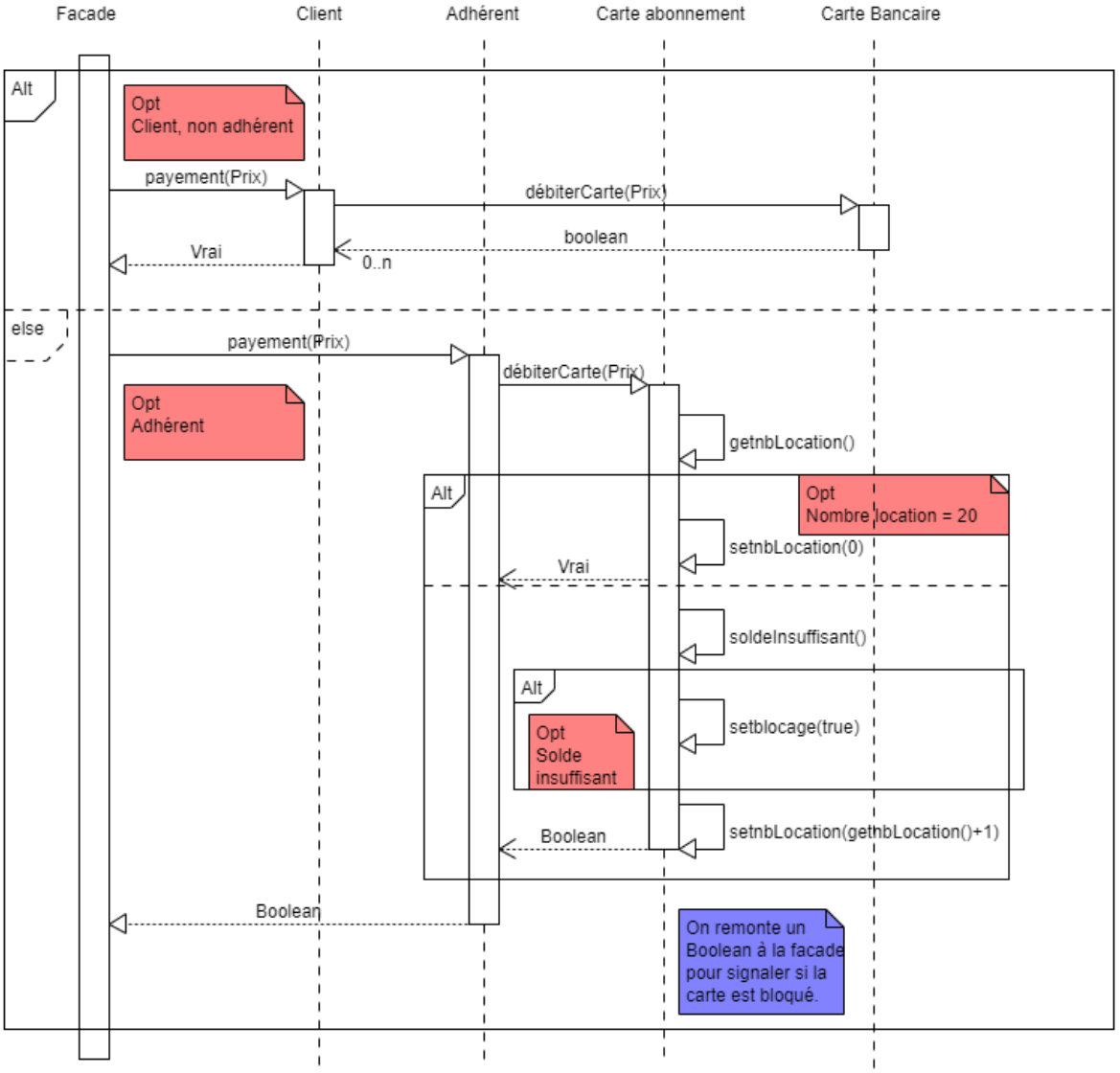
Diagramme de classes du noyau fonctionnel ou coeur métier


Façade du noyau fonctionnel
FacadeNf
- HashMap cdDispo<titre:string, liste[CD]>
- rechercherFilm()
- emprunt(Film)
- paiement()
- rendre()
- souscrire()
Pas plus de 3 locations en même temps, mais avec l'historique, on passe à n.
Location
- début : date
- fin : date
- tarif : Float
- setTarif(tarif)
- genererFacture()
-calculerPrix()
Client
- adresseFacturation : String
+ emprunter(Film)
+ rendre(Film, booléan)
+ rechercherFilm(filtres[])
+ souscrire()
+ paiement(Prix)
+ estEnCours(Film)
La carte bancaire à besoin d'être connue de la carte d'abonnement.
Carte bancaire
- noCB : String
- cryptogramme : String
- dateExpiration : Date
- debiterCarte(solde)
Film
- titre : String
- producteur : String
- acteurs principaux : String[]
- resumé : String
- afficherInformations()
Il existera peut-être un design pattern prototype au niveau des films, CD et QRCode.
Support
- afficherInformations()
+ CalculerDuréeEmprunt()
Adherent
- Nom : String
- Prenom : String
- DateNaiss : Date
- CourrielAdr : String
- Titulaire : Carte abonnement
- Possédé : [carte abonnement]
- emprunter(Film)
- rendre(Film)
- créditerCarte(Carte abonnement, somme)
- consulterHistorique()
- consulterInformationsProfil() (toString)
- paiement(Prix)
- changerRestrictions(Carte abonnement, restrictions[])
- Souscrire(Titulaire)
Lien vers DAO
titulaire
Carte abonnement
- solde : Float
- restriction : String[]
- carteMère : Carte abonnement
- bloque : boolean
- nbLocation : int
- debiterCarte(somme)
- SoldeInsuffisant()
- crediterCarte(somme)
- setRestriction(Restriction[])
- setBlocage(Boolean)
- getBlocage()
- getnbLocation()
- setnbLocation(int)
La connexion se fait à partir de la carte. Si c'est une carte mère, elle connecte l'adhérent avec des options en plus (toute la gestion des autres cartes).
CD
- endommagé : boolean
- setEndommage()
- getEndommage()
- CalculDurée()
QR code
- CalculerDurée(DatePrésent)
0..n
1
1..n

Diagrammes de séquence

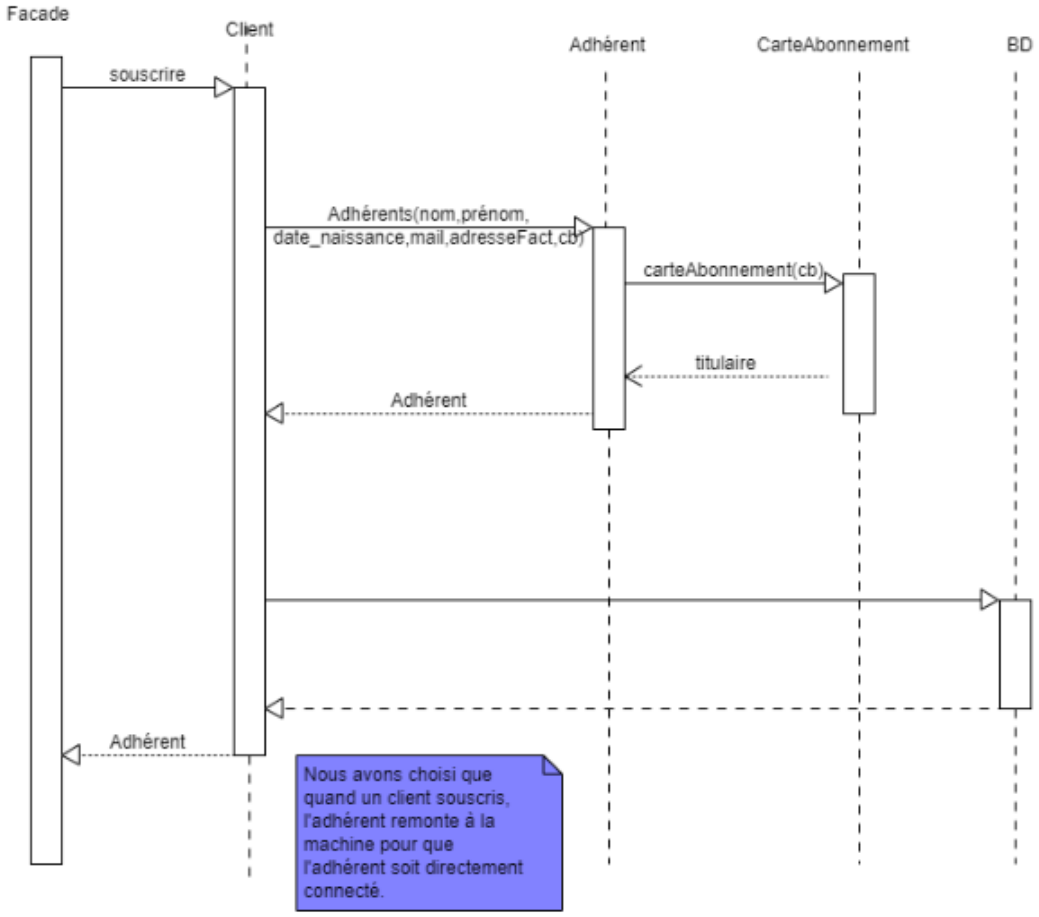
Fonction paiement :

Nous avons fait le choix qu'une carte bancaire pouvait toujours être débitée. La gestion du nombre de locations est faite dans le paiement et de façon automatique pour la réduction.



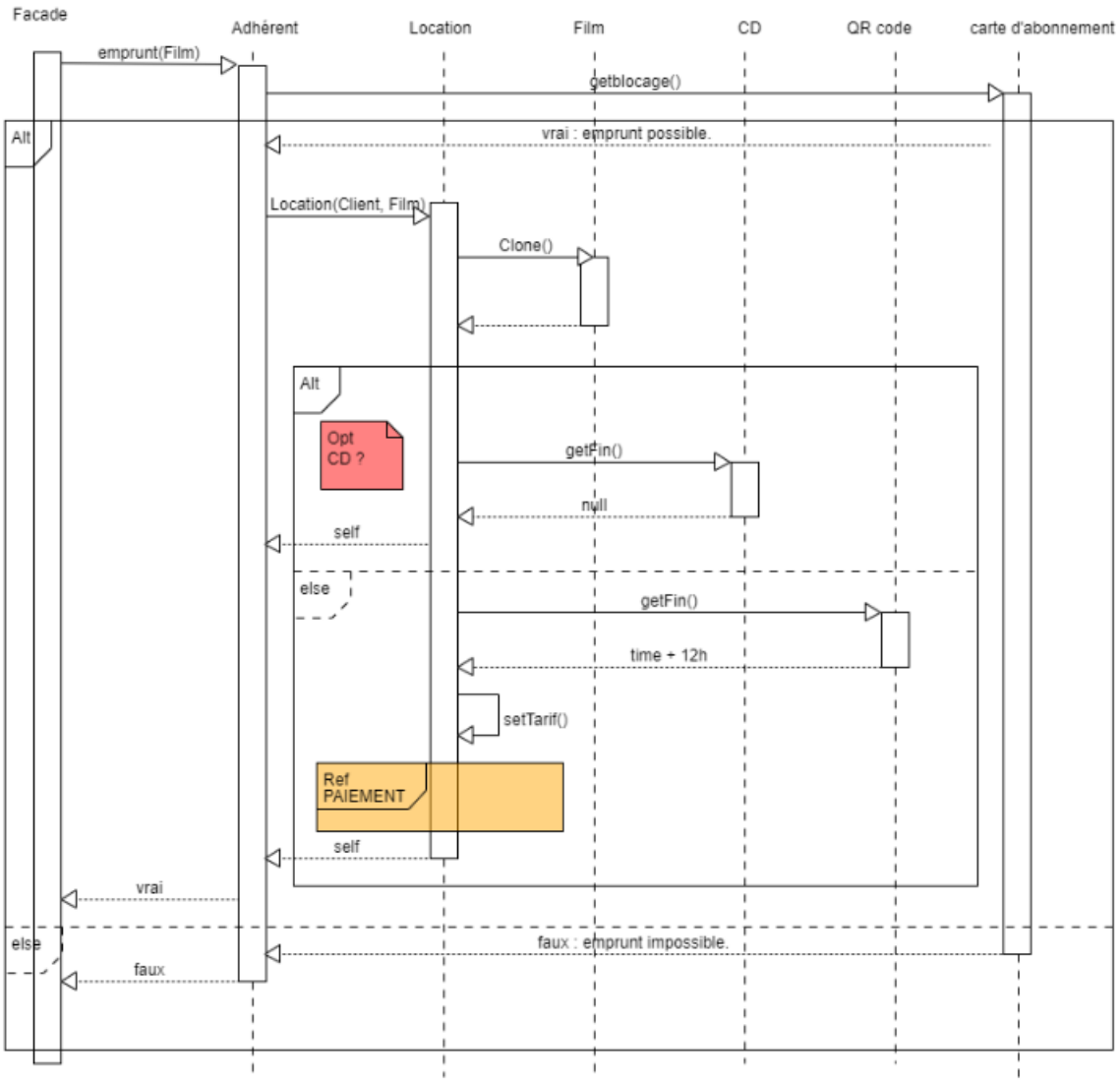
Fonction souscrire :

Le client remplit un formulaire contenant : nom, prénom, date de naissance, adresse postale et numéro de carte bancaire. Il lui est ensuite attribué une nouvelle carte d'abonnement, et un nouvel adhérent est ajouté à notre BD.



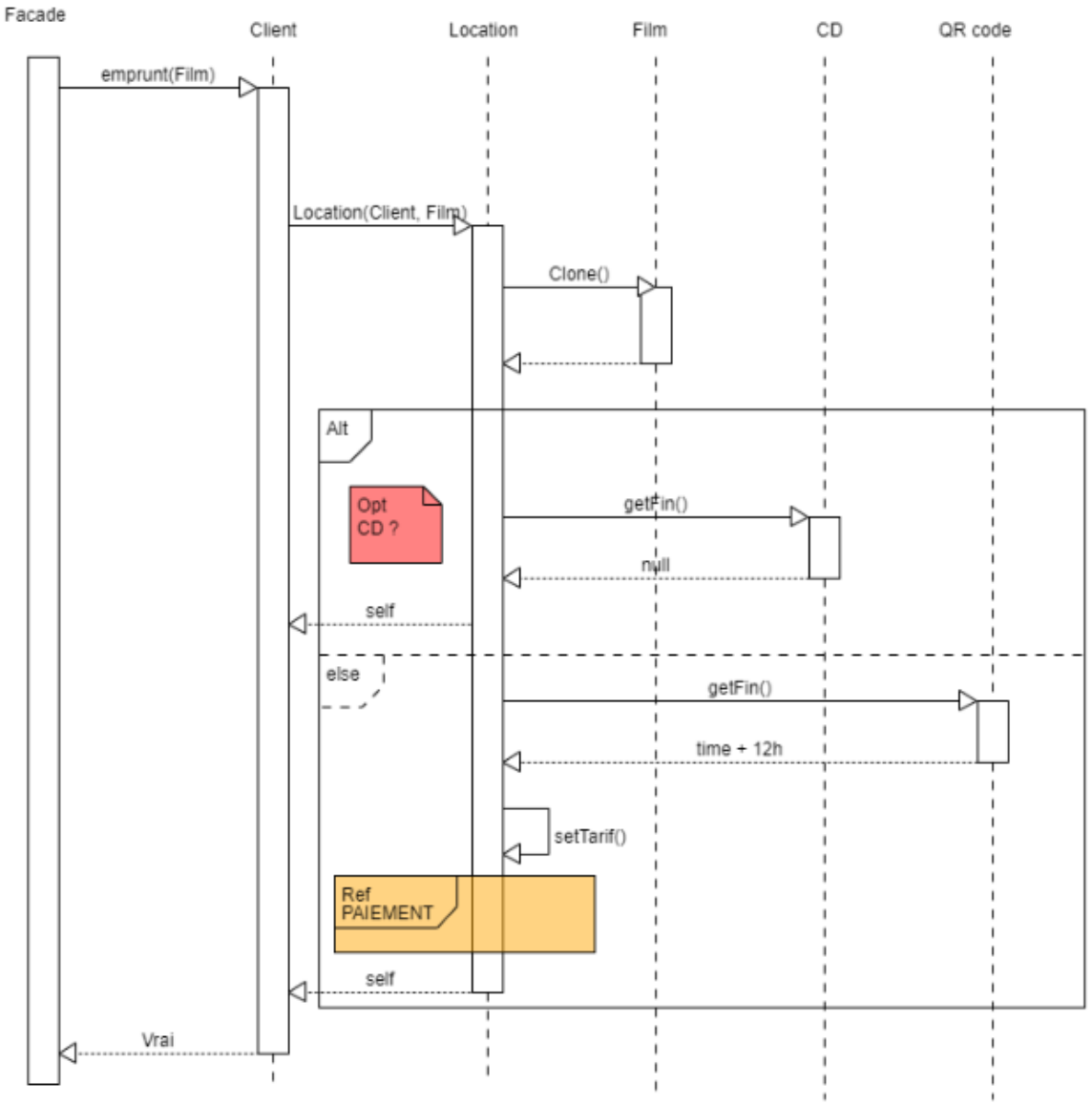
Fonction empruntAdhérent :

L’adhérent choisit son film. On vérifie si sa carte n’est pas bloquée, puis si l’adhérent a pris un blue-ray, il sera servi, sinon si c’est un QR code, l’adhérent paie et le QR code sera valable 12 heures à partir du moment de la location. Notez bien qu'une location est un film pris.



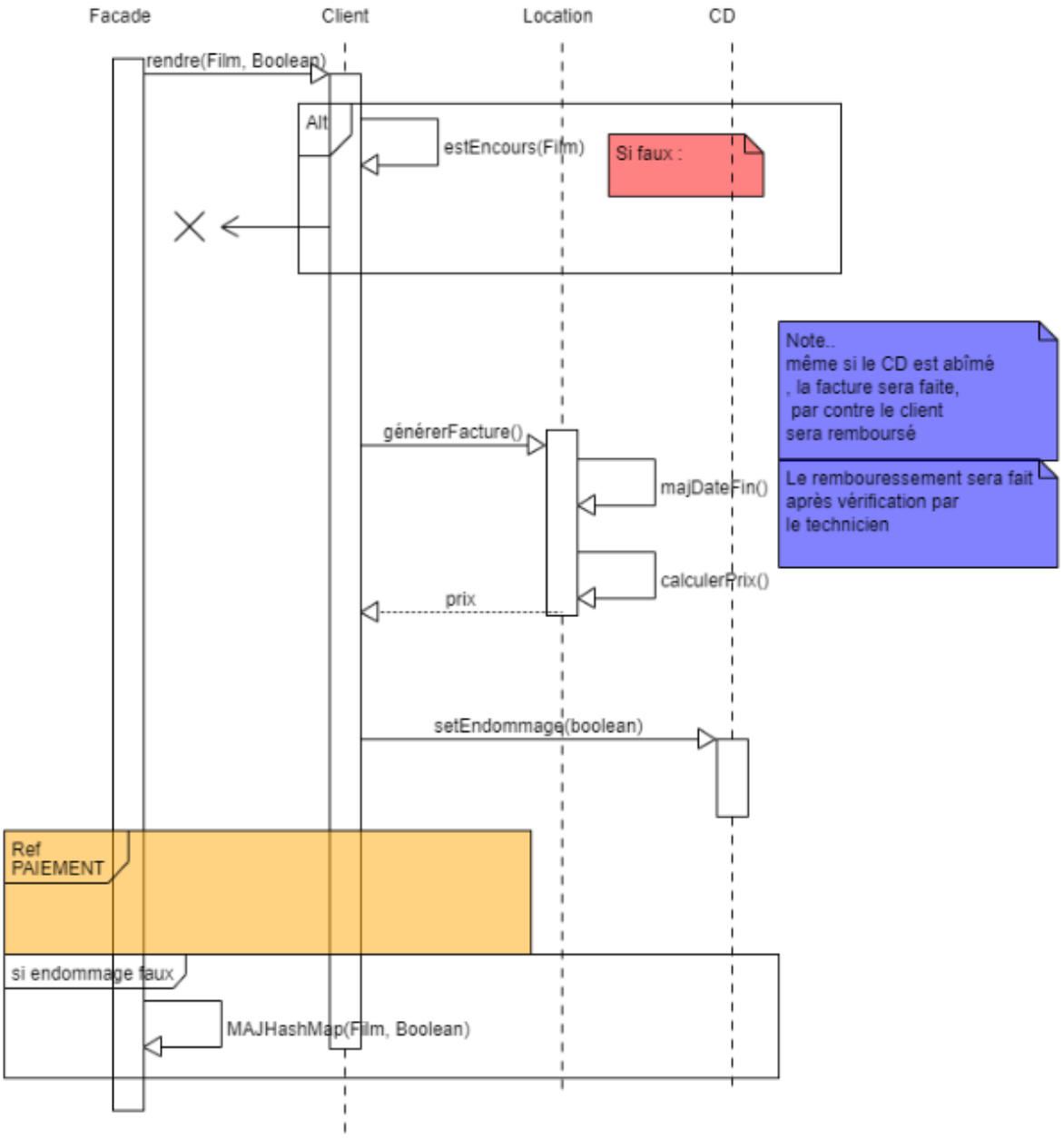
Fonction empruntClient

Le client choisit son film, si il le prend en blue-ray, le disque sera directement délivré, si c'est un QR code,le client paie, le QR code sera valable 12 heures à partir du moment où le client le reçoit



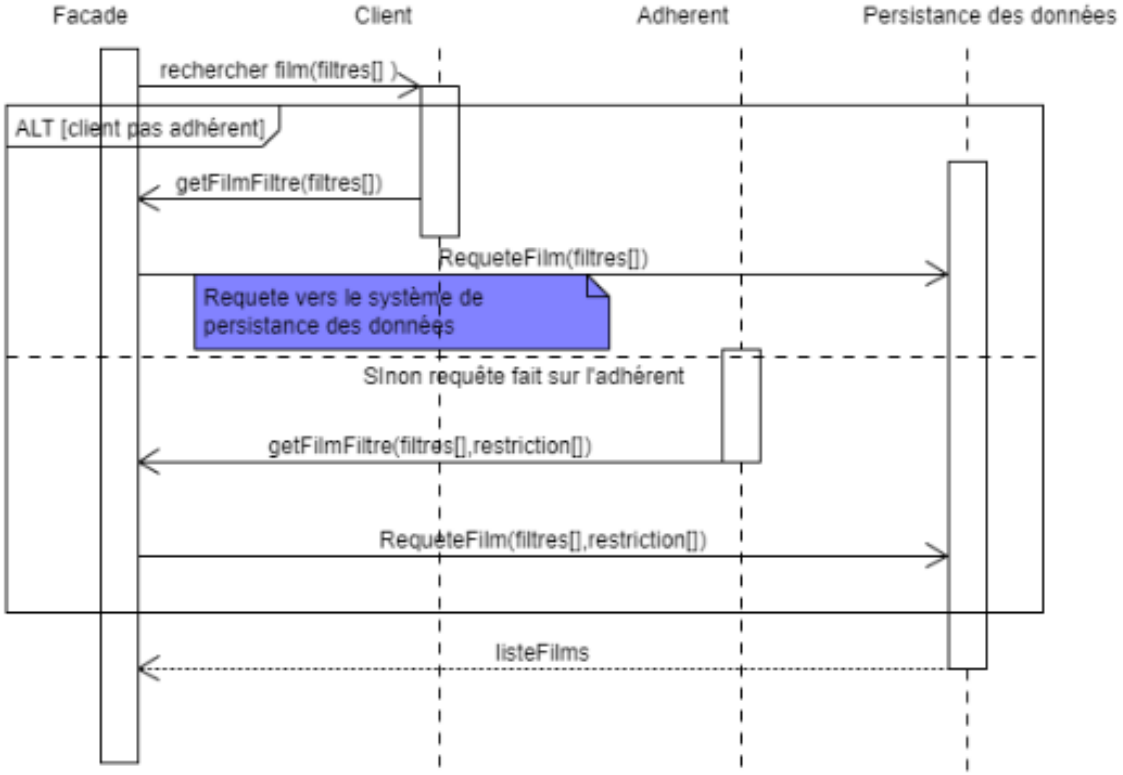
Fonction rendre

Quand le client/adhérent vient rendre un blue-ray, en premier lieu il sera vérifié si c'est bien le bon disque qui est rendu, sinon on redonne le disque, et on sort de la fonction, si c'est le bon disque une facture sera générée, le prix à payer sera calculée et la date de la fin de location sera écrit, le client pourra signaler que le blue-ray est endommagé, puis le payement sera effectué, le blue-ray n'est pas ajouté à notre liste de blue-rays disponibles sur notre distributeur.



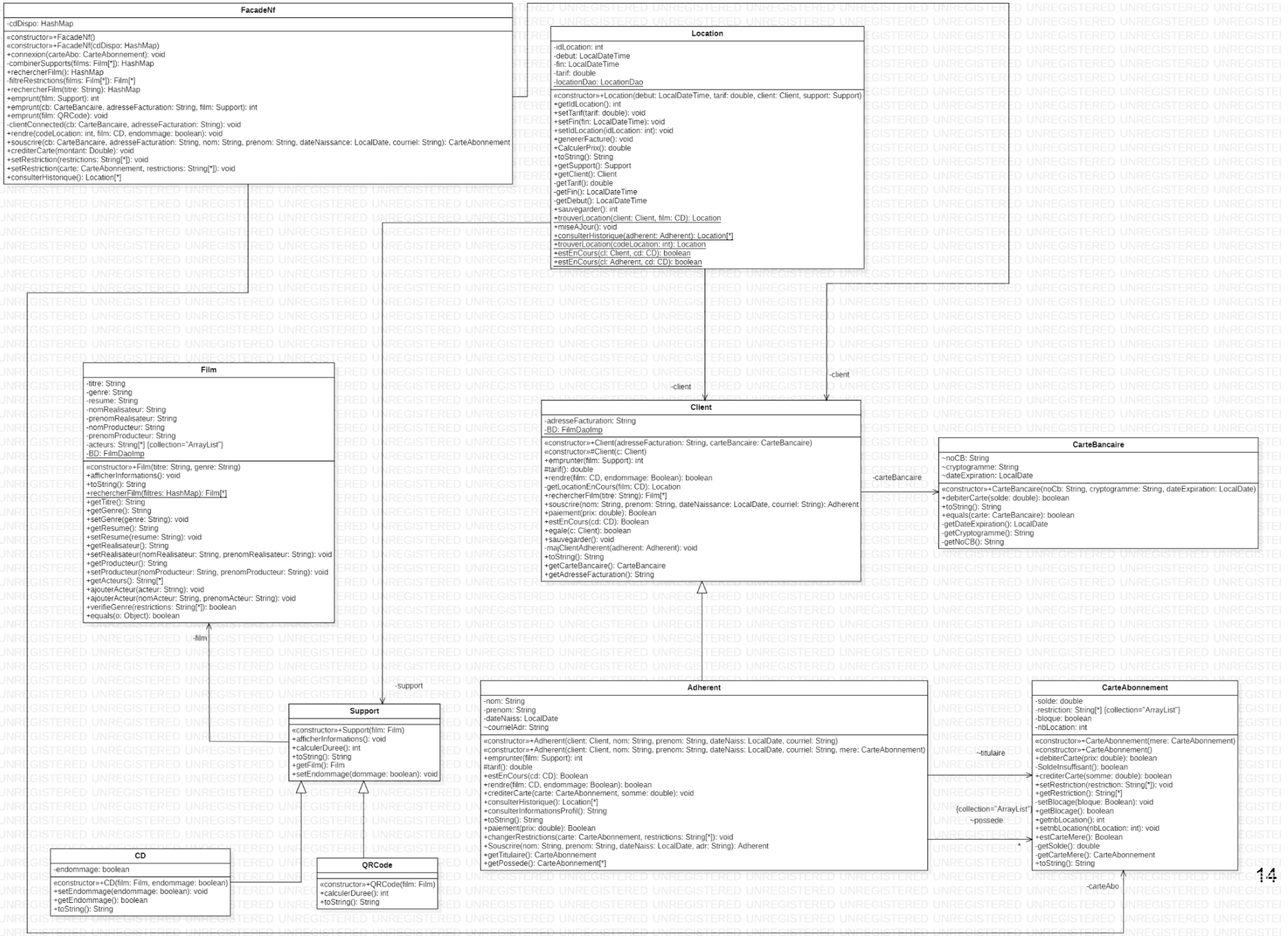
Fonction rechercher_film

Le client choisit la sélection de filtres (titre, acteurs, producteurs, catégorie,...) à appliquer à sa recherche. S'il s'agit d'un client anonyme, une requête sera faite à la base de données avec les filtres imposés par le client. Sinon s'il s'agit d'un adhérent, la requête sera faite avec les filtres et les restrictions de la carte abonnement actuelle. La liste des films est retournée à la façade.



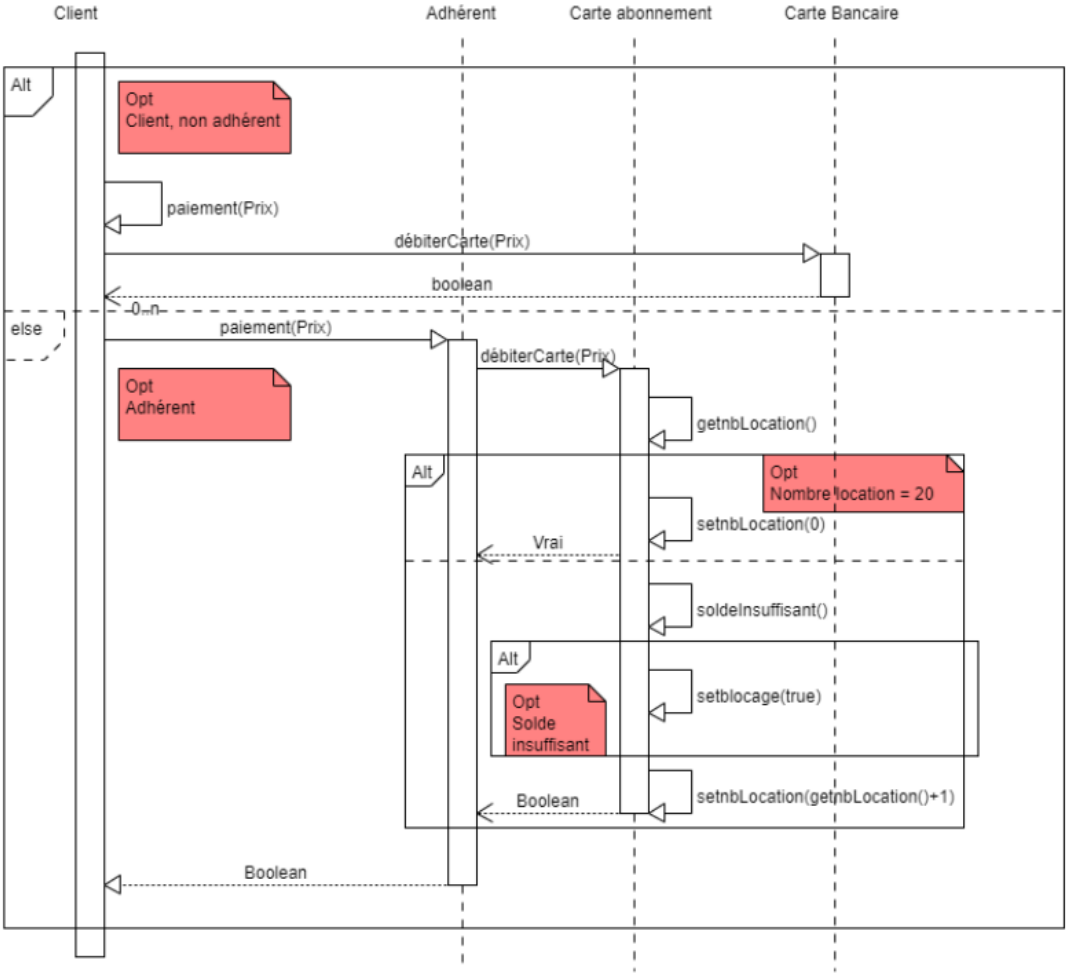
V Diagrammes actualisés

Diagramme de classes du noyau fonctionnel ou coeur métier

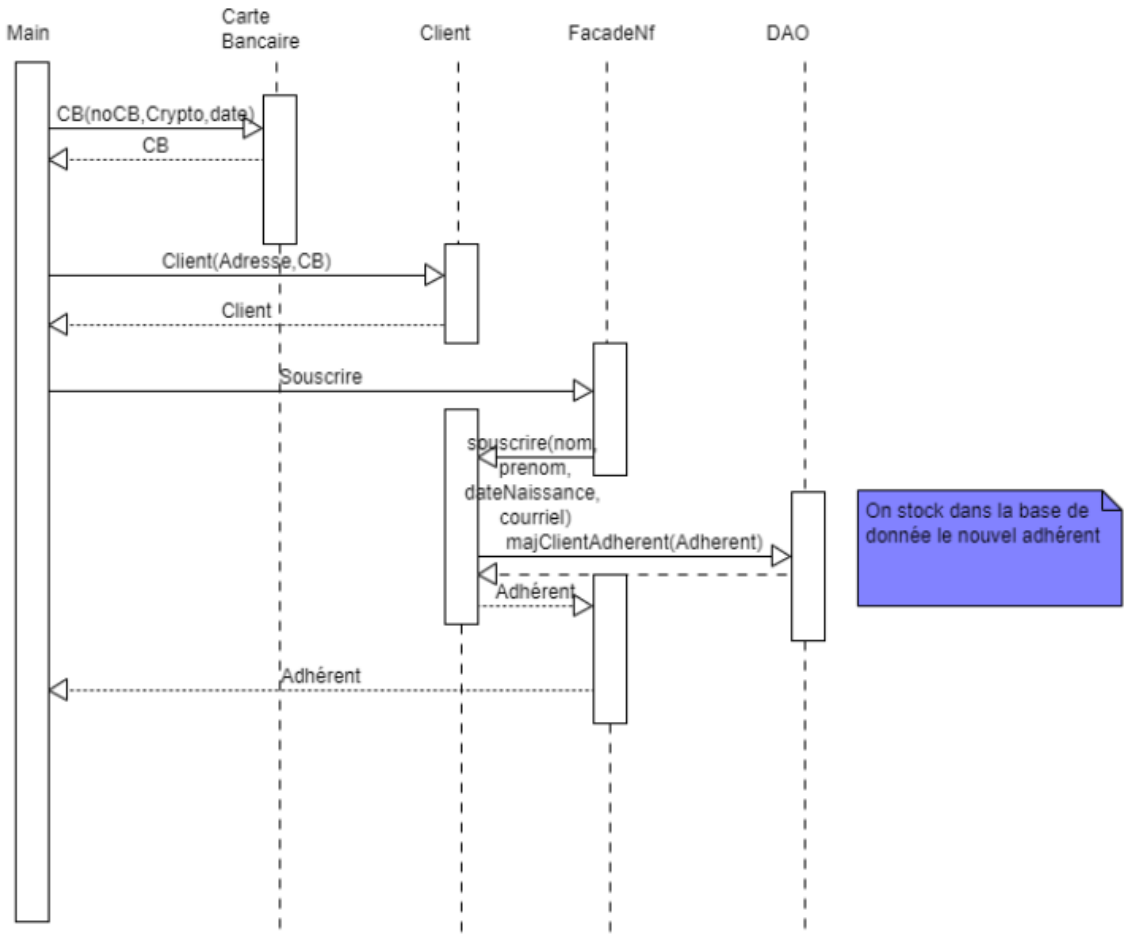

FacadeNf
-cdDispo: HashMap
«constructor»+FacadeNf()
«constructor»+FacadeNf(cdDispo: HashMap)
+connexion(carteAbo: CarteAbonnement): void
-combinerSupports(films: Film[*]): HashMap
+rechercherFilm(): HashMap
-filtreRestrictions(films: Film[*]): Film[*]
+rechercherFilm(titre: String): HashMap
+emprunt(film: Support): int
+emprunt(cb: CarteBancaire, adresseFacturation: String, film: Support): int
+emprunt(film: QRCode): void
-clientConnected(cb: CarteBancaire, adresseFacturation: String): void
+rendre(codeLocation: int, film: CD, endommage: boolean): void
+souscrire(cb: CarteBancaire, adresseFacturation: String, nom: String, prenom: String, dateNaissance: LocalDate, courriel: String): CarteAbonnement
+crediterCarte(montant: Double): void
+setRestriction(restrictions: String[*]): void
+setRestriction(carte: CarteAbonnement, restrictions: String[*]): void
+consulterHistorique(): Location[*]
Location
-idLocation: int
-debut: LocalDateTime
-fin: LocalDateTime
-tarif: double
-locationDao: LocationDao
«constructor»+Location(debut: LocalDateTime, tarif: double, client: Client, support: Support)
+getIdLocation(): int
+setTarif(tarif: double): void
+setFin(fin: LocalDateTime): void
+setIdLocation(idLocation: int): void
+genererFacture(): void
+CalculerPrix(): double
+toString(): String
+getSupport(): Support
+getClient(): Client
-getTarif(): double
-getFin(): LocalDateTime
-getDebut(): LocalDateTime
+sauvegarder(): int
+trouverLocation(client: Client, film: CD): Location
+miseAJour(): void
+consulterHistorique(adherent: Adherent): Location[*]
+trouverLocation(codeLocation: int): Location
+estEnCours(cl: Client, cd: CD): boolean
+estEnCours(cl: Adherent, cd: CD): boolean
Film
-titre: String
-genre: String
-resume: String
-nomRealisateur: String
-prenomRealisateur: String
-nomProducteur: String
-prenomProducteur: String
-acteurs: String[*] {collection="ArrayList"}
-BD: FilmDaoImp
«constructor»+Film(titre: String, genre: String)
+afficherInformations(): void
+toString(): String
+rechercherFilm(filtres: HashMap): Film[*]
+getTitre(): String
+getGenre(): String
+setGenre(genre: String): void
+getResume(): String
+setResume(resume: String): void
+getRealisateur(): String
+setRealisateur(nomRealisateur: String, prenomRealisateur: String): void
+getProducteur(): String
+setProducteur(nomProducteur: String, prenomProducteur: String): void
+getActeurs(): String[*]
+ajouterActeur(acteur: String): void
+ajouterActeur(nomActeur: String, prenomActeur: String): void
+verifieGenre(restrictions: String[*]): boolean
+equals(o: Object): boolean
Client
-adresseFacturation: String
-BD: FilmDaoImp
«constructor»+Client(adresseFacturation: String, carteBancaire: CarteBancaire)
«constructor»#Client(c: Client)
+emprunter(film: Support): int
#tarif(): double
+rendre(film: CD, endommage: Boolean): boolean
-getLocationEnCours(film: CD): Location
+rechercherFilm(titre: String): Film[*]
+souscrire(nom: String, prenom: String, dateNaissance: LocalDate, courriel: String): Adherent
+paiement(prix: double): Boolean
+estEnCours(cd: CD): Boolean
+egale(c: Client): boolean
+sauvegarder(): void
-majClientAdherent(adherent: Adherent): void
+toString(): String
+getCarteBancaire(): CarteBancaire
+getAdresseFacturation(): String
CarteBancaire
-noCB: String
-cryptogramme: String
-dateExpiration: LocalDate
«constructor»+CarteBancaire(noCb: String, cryptogramme: String, dateExpiration: LocalDate)
+debiterCarte(solde: double): boolean
+toString(): String
+equals(carte: CarteBancaire): boolean
-getDateExpiration(): LocalDate
-getCryptogramme(): String
-getNoCB(): String
Support
«constructor»+Support(film: Film)
+afficherInformations(): void
+calculerDuree(): int
+toString(): String
+getFilm(): Film
+setEndommage(dommage: boolean): void
Adherent
-nom: String
-prenom: String
-dateNaiss: LocalDate
~courrielAdr: String
«constructor»+Adherent(client: Client, nom: String, prenom: String, dateNaiss: LocalDate, courriel: String)
«constructor»+Adherent(client: Client, nom: String, prenom: String, dateNaiss: LocalDate, courriel: String, mere: CarteAbonnement)
+emprunter(film: Support): int
#tarif(): double
+estEnCours(cd: CD): Boolean
+rendre(film: CD, endommage: Boolean): boolean
+crediterCarte(carte: CarteAbonnement, somme: double): void
+consulterHistorique(): Location[*]
+consulterInformationsProfil(): String
+toString(): String
+paiement(prix: double): Boolean
+changerRestrictions(carte: CarteAbonnement, restrictions: String[*]): void
+Souscrire(nom: String, prenom: String, dateNaiss: LocalDate, adr: String): Adherent
+getTitulaire(): CarteAbonnement
+getPossede(): CarteAbonnement[*]
CarteAbonnement
-solde: double
-restriction: String[*] {collection="ArrayList"}
-bloque: boolean
-nbLocation: int
«constructor»+CarteAbonnement(mere: CarteAbonnement)
«constructor»+CarteAbonnement()
+debiterCarte(prix: double): boolean
-SoldeInsuffisant(): boolean
+crediterCarte(somme: double): boolean
+setRestriction(restriction: String[*]): void
+getRestriction(): String[*]
-setBlocage(bloque: Boolean): void
+getBlocage(): boolean
+getnbLocation(): int
+setnbLocation(nbLocation: int): void
+estCarteMere(): Boolean
-getSolde(): double
-getCarteMere(): CarteAbonnement
+toString(): String
CD
-endommage: boolean
«constructor»+CD(film: Film, endommage: boolean)
+setEndommage(endommage: boolean): void
+getEndommage(): boolean
+toString(): String
QRCode
«constructor»+QRCode(film: Film)
+calculerDuree(): int
+toString(): String
-client
-client
-carteBancaire
-film
-support
~titulaire
{collection="ArrayList"}
~possede
*
-carteAbo

Diagrammes de séquence

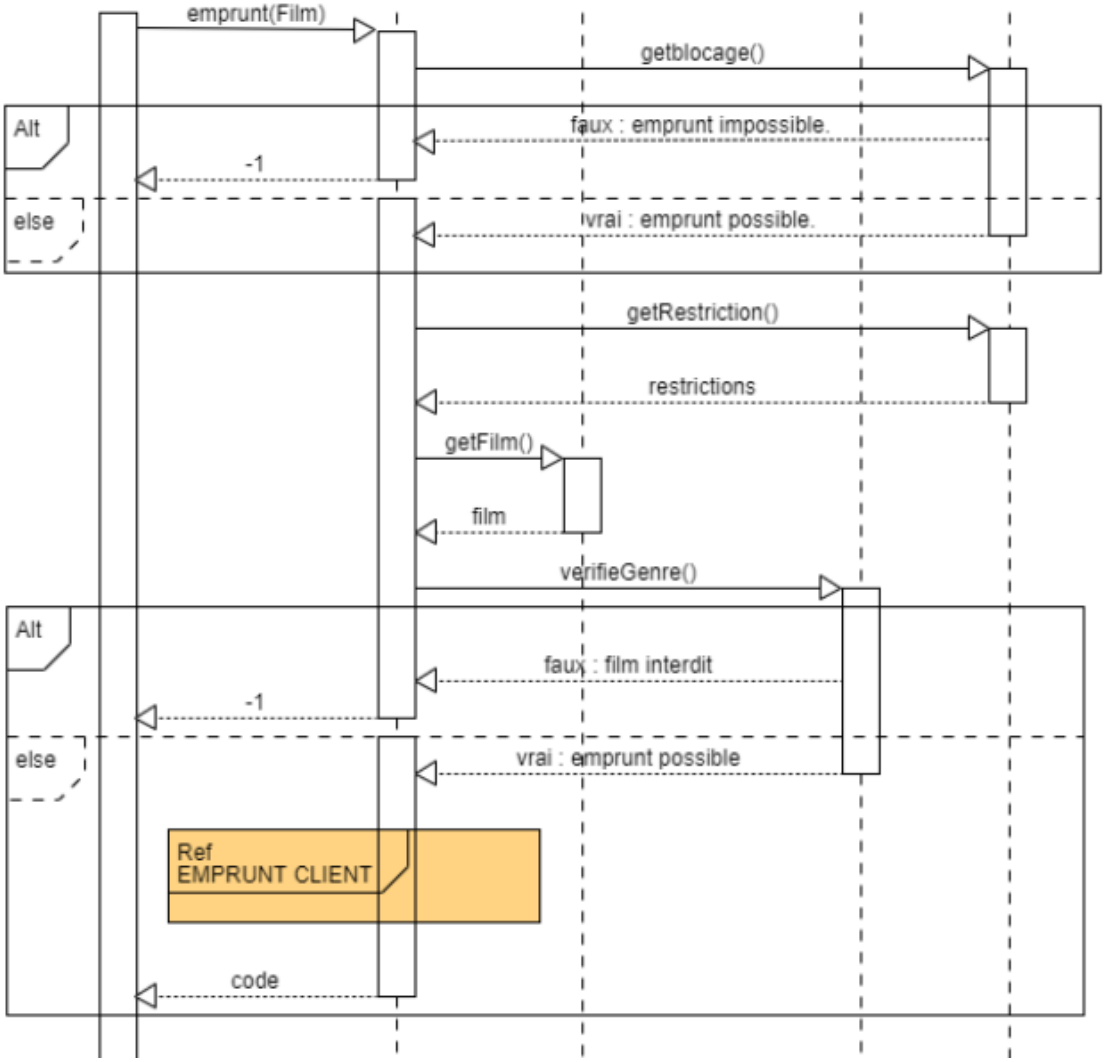
Fonction paiement :



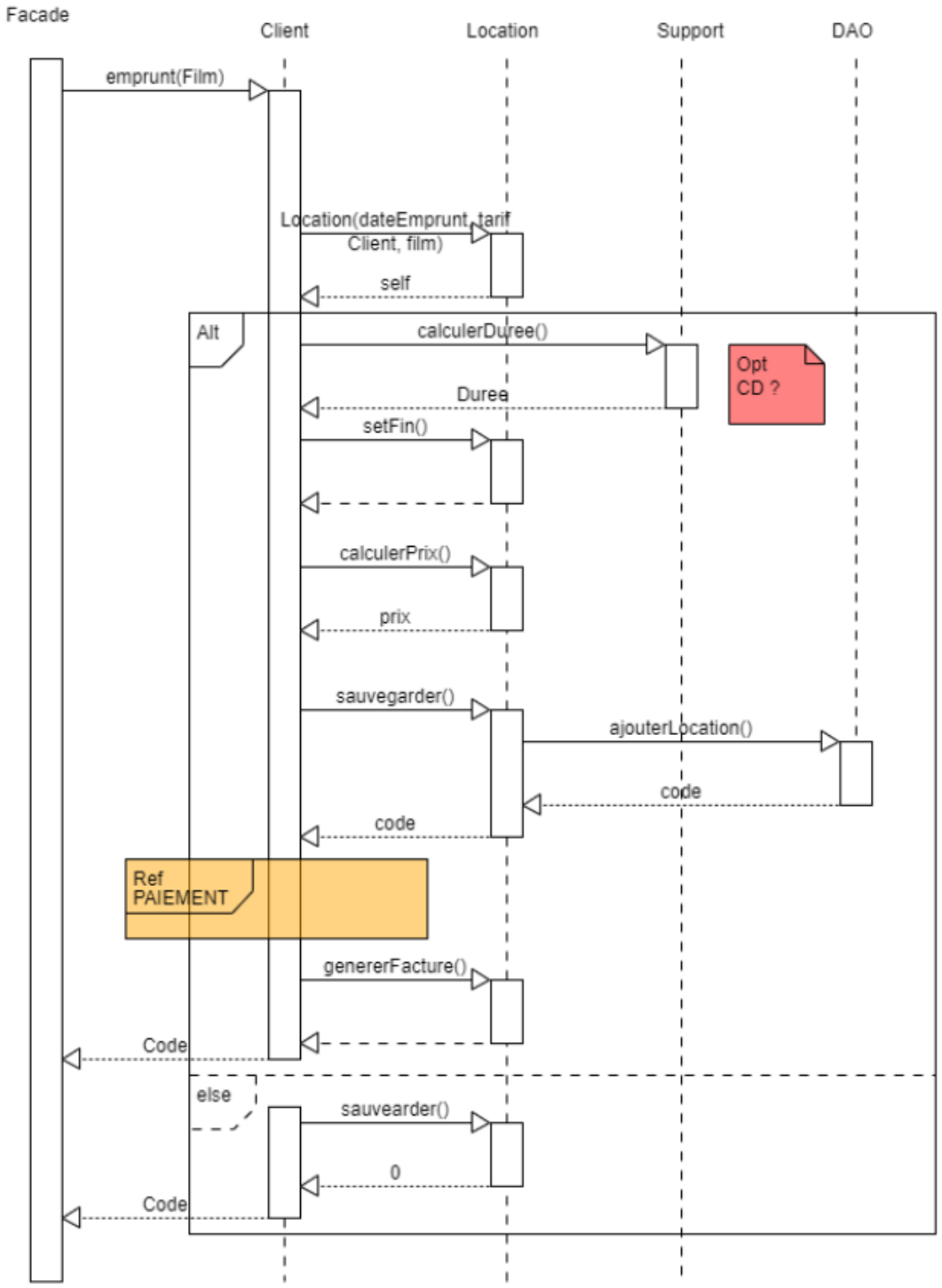
Fonction souscrire :



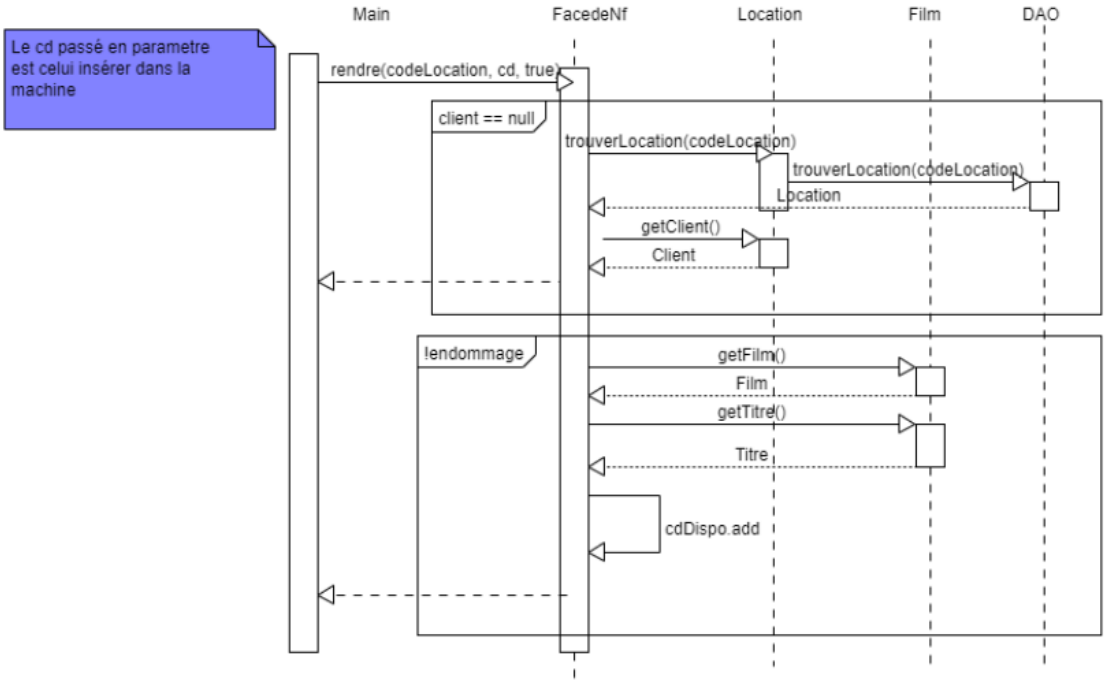
Fonction empruntAdhérent :



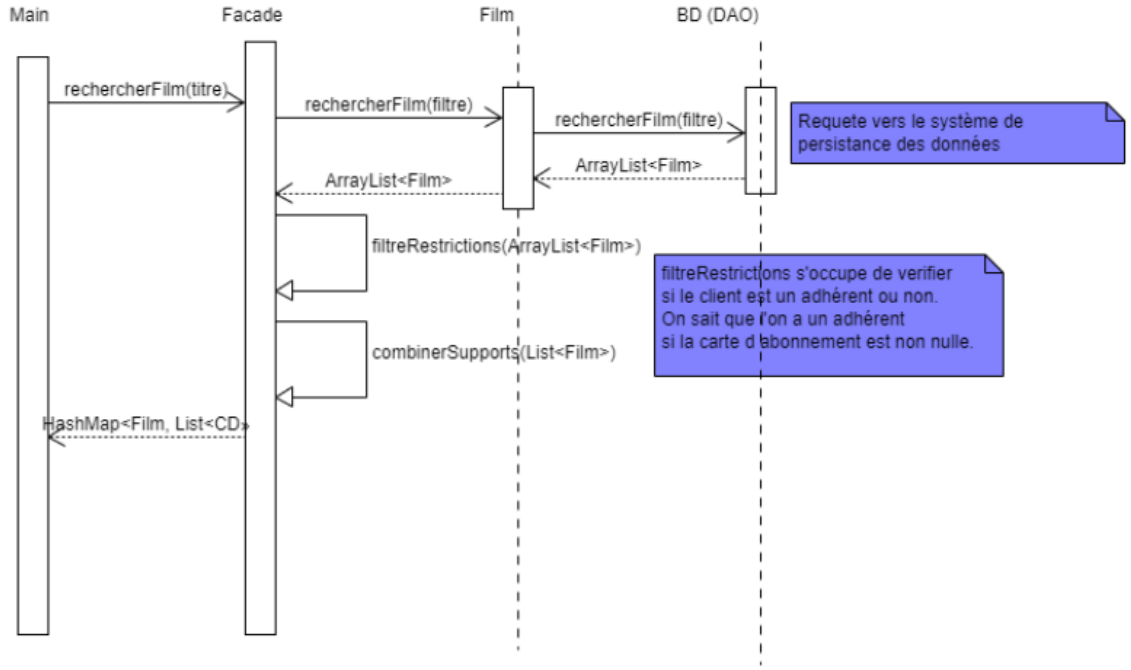
Fonction empruntClient :



Fonction rendre :



Fonction rechercher_film :

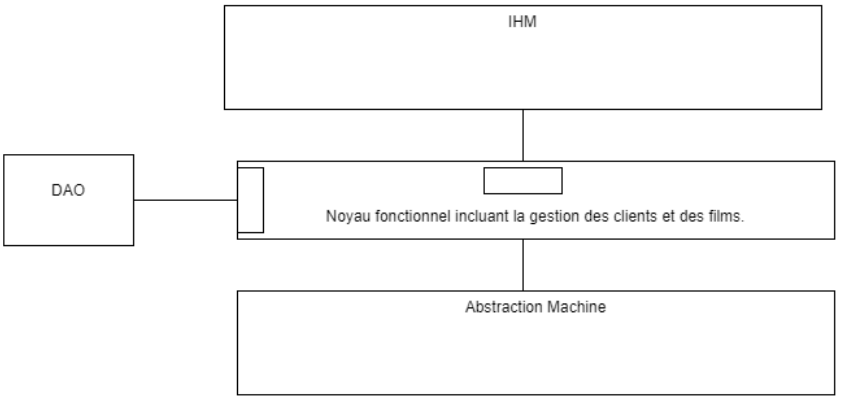


VI Bilan

État du projet

Comparaison architecture de départ et finale

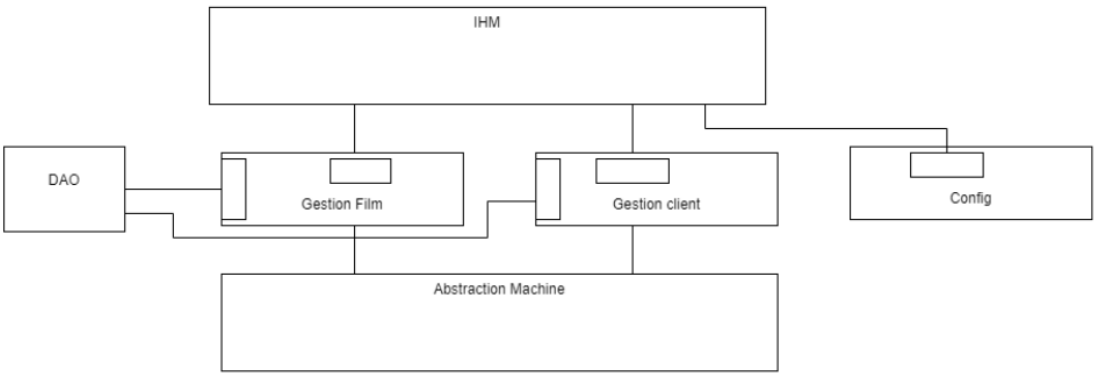
L'architecture a évolué entre la conception initiale et la fin du projet. Premièrement, nous avons réalisé la conception d'un noyau fonctionnel et nous avons supposé l'existence d'autres composants dans l'architecture comme l'IHM. Voici l'architecture que nous avons obtenu :



Finalement, nous avons obtenu différents composants :

- Un noyau fonctionnel qui comporte des façades pour l'isoler des autres composants
- Une couche DAO permettant l'accès à la base de données.
- Une IHM pour l'interface du logiciel

Idéalement, l'architecture devrait tendre vers le schéma suivant :



Le noyau fonctionnel comprend la gestion des films et la gestion des clients. La gestion des films comprend notamment les classes Film, Support, CD et QRCode alors que la gestion des clients comprend les classes Client, Adhérent et CarteAbonnement. La classe Location fait le pont entre la gestion des films et des clients et la classe CarteBancaire fait partie d'un module Banque qui n'est généralement pas développé mais repris ou acheté. Ici, on considère que CarteBancaire fait partie de la gestion des clients.

Le module IHM représenté par les classes de l'interface va interagir avec le noyau fonctionnel à l'aide de façades en rappelant que le noyau fonctionnel ne connaît pas l'IHM. L'interface s'adaptera en fonction de la configuration de la machine.

Le module DAO représenté par les différents DAO des classes (FilmDao, LocationDao, ClientDao,...) du noyau fonctionnel fait le pont entre ce dernier au travers de façades et la base de données.

Choix Techniques

La base de données est toujours une couche à part dans la programmation orientée objet et il est souvent difficile de l'insérer dans le code. C'est pour cela que l'on utilise le DAO afin d'abstraire l'accès aux données et de séparer cet accès du noyau fonctionnel.

Nous avons implémenté dans les DAO des singletons de manière à n'avoir qu'une seule instance de "la base de données locale". Ces singletons permettent d'émuler une base de données. Si nous ne les utilisons pas, nous devons recréer une "base" et l'on perdrait alors les données.

Nous avons régulièrement utilisé les itérateurs des classes utilisées de manière à s'abstraire de la construction interne des classes de Java.
Nous n'avons pas pu l'implémenter dans le code mais nous avons supposé qu'il était possible d'utiliser des visiteurs au niveau des tarifs pour rappeler les visiteurs au moment du paiement.

Il est probable qu'un design pattern créationnel soit utile au niveau de la création de CDs et de QR codes mais, nous ne savons pas lequel est le plus adapté.

La base de données sqlite regroupe les films, genres et acteurs et permet d'effectuer notamment des recherches de films.

Fonctionnalités implémentées

Ce n'est pas tout à fait une fonctionnalité, mais la documentation du projet est complète.
L'emprunt d'un cd s'il y en a de disponibles ou d'un qr code pour un film donnée par un client ou un adhérent fonctionne.
Le rendu d'un cd fonctionne.
La modification des restrictions d'une carte d'abonnement et le filtrage des résultats de la recherche de film fonctionne.
La recherche d'un film précis avec son titre est implémenté en entier.
La recherche avec d'autres filtres est implémentée en interne mais pas dans le main.
Un adhérent peut créditer le solde de sa carte.

Reste à faire

La souscription d'un nouvel abonnement par un adhérent n'est pas complètement fonctionnelle.
Gestion de la relation mère-fille sur les cartes abonnements.
La consultation de l'historique n'est pas fonctionnelle mais ne bloque pas l'exécution.
La recherche de tous les films sans filtres est implémentée dans le noyau fonctionnel mais pas dans le Main.
Ajouter la possibilité d'envoyer d'autres filtres que le titre à partir du Main.

Compilation et exécution

Vous pouvez utiliser un terminal.
Sous windows :

- Placez vous dans le dossier AL2000_grp_1_A
- Pour compiler, lancez la commande :

javac -classpath ".;sqlite-jdbc-3.36.0.3.jar" fc\Main.java
ou
javac -cp ".;sqlite-jdbc-3.36.0.3.jar" fc\Main.java

- Pour exécuter, lancez la commande :

java -classpath ".;sqlite-jdbc-3.36.0.3.jar" fc\Main.java
ou
java -cp ".;sqlite-jdbc-3.36.0.3.jar" fc\Main.java

Il est important de rester dans le dossier AL2000_grp_1_A pour que la compilation et l'exécution se déroulent bien. Vous devez donner le chemin en relatif ou absolu du driver sqlite (sqlite-jdbc-3.36.0.3.jar) afin d'accéder aux fonctionnalités de la base de données.

Sous Linux (Les commandes n'ont pas pu être testées) :

- Placez vous dans le dossier AL2000_grp_1_A
- Pour compiler, lancez la commande :

javac -cp ".:/home/path/sqlite-jdbc-3.36.0.3.jar;" fc/Main.java
ou
javac -classpath ".:/home/path/sqlite-jdbc-3.36.0.3.jar;" fc/Main.java

- Pour exécuter, lancez la commande :

java -cp ".:/home/path/sqlite-jdbc-3.36.0.3.jar;" fc/Main.java
ou
java -classpath ".:/home/path/sqlite-jdbc-3.36.0.3.jar;" fc/Main.java

Vous pouvez aussi utiliser un IDE et il vous faudra ajouter le driver.
Sous Intellij :

1) **File** > **Project Structure** (Dans les menus en haut à gauche, cela ouvrira une fenêtre)
2) **Project Settings** > **Modules** (dans l'arborescence à gauche de la fenêtre ouverte)
3) Sélectionner **Dependencies**
4) Cliquer sur le bouton **'+'** et sélectionner **JAR or Directories**
5) Aller **chercher le driver sqlite-jdbc-3.36.0.3.jar** se trouvant dans AL2000_grp_1_A, **sélectionnez le** et cliquez sur **OK**
6) Cliquer sur **Apply**

Vous pourrez ensuite très aisément compiler et exécuter en cliquant sur les flèches vertes se trouvant à côté des différentes méthodes statiques main dans le projet. Par exemple, dans fc/Main pour lancer le programme principal, dans Test/test_unitaire ou encore fc/Dao/FilmDaoImp.

Difficultés rencontrées

Une des difficultés que nous avons le plus éprouvées durant ce projet est la manipulation des DAO. En effet, bien que ceux-ci soient une solution très pratique pour abstraire l'accès au données, leur implémentation n'est pas simple.

Nous avons aussi eu un problème sur la gestion des dates. De nombreuses fonctionnalités des classes les plus courantes ne sont plus maintenues, ce qui nous à demander de rechercher une classe toujours maintenue.

Notre relative inexpérience avec les design pattern nous cause bien des soucis. Nous ne sommes pas encore habitués à les repérer et nous n'avons donc pas su identifier toutes les occasions où un design pattern aurait pu simplifier notre code.

Le travail de groupe n'a pas été un problème, chacun a réussi à se trouver une place et à fournir une quantité de travail non négligeable.

Améliorations possibles

Au terme de ce projet, nous avons pu débriefer et nous rendre compte que nous aurions pu améliorer certaines choses au cours de développement afin d'augmenter la qualité finale de notre projet.

Tout d'abord, nous sommes conscients de la nécessité de mieux tester notre code. Actuellement, nous nous contentons de quelques tests basiques sans utiliser les outils spécialisés tels que JUnit ou SonarQube pour contrôler différentes métriques qualités comme la couverture de code. Ajouter des tests permettrait de vérifier plus formellement notre gestion des cas limites et de la robustesse de notre noyau fonctionnel. De plus, le faire plus tôt dans la programmation nous aurait probablement permis de faire ressortir certains bogues, notamment ceux de pertes de références.

Une seconde amélioration que l'on aurait faite si nous n'avions pas été pris par le temps, aurait été d'implémenter dans sa totalité une persistance réelle de données. Dans notre cas, nous avons prévu notre conception afin de pouvoir connecter n'importe quelle persistance de données tant qu'elle répondait au contrat fixé par nos DAO. Pour le montrer, nous avons implémenté une partie d'une base de données sur les films, genres et acteurs grâce à SQLite. Cependant les autres données de sont stockées que pour la durée d'exécution du programme.

Par rapport à la lisibilité et à la syntaxe, nous n'avons pas mis en place assez précisément une convention de nommage, de commit, de documentation ce qui fait que l'on obtient par moment des noms de variables plus ou moins explicites, en anglais et en français ce qui amène des incohérences dans la lecture.

Enfin, une dernière amélioration à laquelle nous avons pensé, revient à diviser à nouveau notre noyau fonctionnel en paquetage représentant des fonctionnalités plus précises en se rapprochant du schéma de l'architecture (cf : Comparaison architecture de départ et finale). Cette amélioration nous offrirait une meilleur modularité dans le code et les fonctions et simplifierait chacune des parties respectives et leurs interconnexions.

Outils de travail

Nous avons travaillé avec les éditeurs et IDE Visual Studio Code (VSC), Intellij et Eclipse.

L'outil git a été utilisé pour versionner le projet et un dépôt github a été créé.

Discord nous aura servi pour support audio et un webhook nous prévenait à chaque fois qu'un commit était poussé sur le dépôt.

Avec différents plug-ins de VSC, Intellij et Eclipse nous avons généré la documentation et certaines fonctions (toString) des classes. L'outil LiveShare de VSC nous aura été utile pour collaborer sur une même session de travail.

Les premiers diagrammes UML ont été construits à l'aide du plug-in UMLet de VSC et le diagramme de classes final a été généré à partir d'une extension de StarUML afin de pouvoir les comparer.

Apprentissage personnel

Ce projet nous aura permis de découvrir les diverses phases de développement d'un logiciel et notamment de pratiquer et de mieux comprendre la définition des exigences et la conception. Il nous aura permis de simuler le contact avec le client, d'extraire les exigences, de les formuler et d'en déduire une conception prenant en compte l'aspect interface, fonctionnel et sauvegarde des données. Ces étapes nous ont montré la distinction entre exigences et conception. Très souvent, nous avons été bloqués dans la conception lorsque nous allions trop dans l'implémentation et que nous perdions l'aspect sémantique.

De plus, lors du développement, nous avons remarqué que la conception nous a permis de démarrer plus rapidement, d'établir très vite les classes, leurs champs et au moins les signatures des premières méthodes.

Enfin, certaines décisions faites lors de la conception n'étaient pas adaptées et parfois nous devions revenir aux définitions des exigences, refaire des morceaux de la conception et adapter le code, ce qui nous a mis en évidence le cycle de vie du logiciel.