BIBLIO-BRETTE

Projet Java Applications Web

Lucas Pinard – INFO2A Groupe 204

13/01/2020

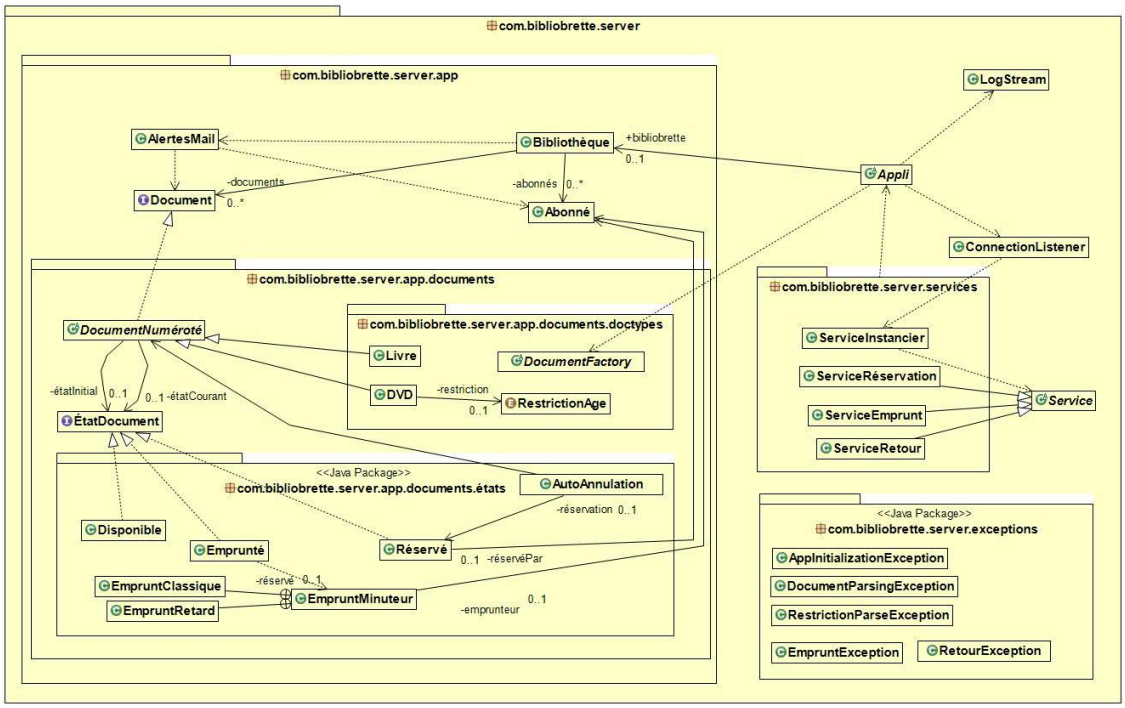
[Cette page est volontairement laissée vierge]

BIBLIO-BRETTE

Projet Java Applications Web

Table des matières

1. L'application coté serveur 3
1.1. La gestion de la bibliothèque et des documents 3
1.2. Les implémentations de documents 3
1.3. Les états de Document 4
1.4. Les certifications 5
2. Communication serveur-client 6
2.1. Sockets d'écoute 6
2.2. Services rendus par l'application 6



1. L'APPLICATION COTE SERVEUR

1.1. La gestion de la bibliothèque et des documents

La classe Abonné représente un inscrit à la bibliothèque. Un inscrit est caractérisé par son numéro attribué automatiquement (section critique nécessitant de verrouiller la classe au thread appelant afin de garantir l'unicité du numéro), nom, son adresse e-mail et sa date de naissance. Un abonné peut être sanctionné, lorsque cela arrive le programme inscrit la date d'expiration de la sanction, et pour savoir si l'abonné est sanctionné le programme compare la date actuelle à la date d'expiration de la sanction.

L'interface Document, telle que définie dans le sujet, définit les actions d'un document. En l'occurrence, un Document dispose d'un numéro, et peut être réservé, emprunté et retourné.

La classe Bibliothèque est la classe centrale des ressources de l'application. Elle regroupe deux *concurrentHashMap*, une pour les abonnés et une pour les documents. Ces derniers sont accessibles par leur numéro, unique, qui leurs servent de clé.

1.2. Les implémentations de documents

La classe DocumentNuméroté est l'implémentation partielle de base de l'interface Document. Cette classe implémente l'attribution automatique des numéros des documents (section critique nécessitant de verrouiller la classe au thread appelant afin de garantir l'unicité des numéros), la vérification d'autorisation de consulter un document pour un abonné, et implémente la *thread-safety* des méthodes réserver, emprunter et retour. Pour faire cela, lorsque la méthode est appelée, une fois que l'on a vérifié que l'abonné était autorisé à accéder au document, on verrouille le document au thread du service appelant jusqu'à ce que le changement d'état se soit fait (ou non). Le fonctionnement intrinsèque des états du document est quant à lui délégué à l'interface ÉtatDocument (cf. *Design Pattern : State*).

Il existe deux classes concrètes de Document à l'heure actuelle. La première, la classe Livre, modélise un livre et est représenté par son titre, son auteur, son éditeur et son année de parution. Elle ne propose aucune fonctionnalité supplémentaire, à l'opposé de la classe DVD, qui modélise un film sur disque représenté par son nom, son réalisateur et son année de sortie, et qui implémente une fonctionnalité de restriction d'âge en surchargeant la méthode de vérification depuis DocumentNuméroté. Cette vérification est déléguée à la classe énumérative RestrictionÂge, qui énumère les différentes restrictions existantes (à l'heure actuelle : « tout public », « -12 » et « -16 »).

La classe DocumentFactory est une classe disposant d'une unique méthode statique permettant d'instancier un nouveau Document en fonction d'arguments passés sous forme de listes de String.

1.3. Les états de Document

Les documents peuvent basculer entre différents états selon les actions des abonnés. Afin d'encapsuler correctement ces différents états, ils sont implémentés par une classe chacun étendant une interface ÉtatDocument permettant de déléguer et d'abstraire le fonctionnement des états vis-à-vis du reste de l'application. Chaque méthode déléguée à ÉtatDocument prend en valeur de retour l'instance d'ÉtatDocument représentant l'état du document une fois l'action effectuée.

La classe Disponible est l'état initial de tout document créé. Lorsqu'un document disponible est demandé à être emprunté ou réservé par un abonné y étant permis, le changement d'état se fait sans restriction. Cependant, un document Disponible ne peut être retourné.

La classe Réservé est l'état faisant suite de la réservation d'un document disponible. Un document réservé peut être retourné, ce qui a pour effet d'annuler la réservation et de changer l'état en Disponible, ou bien être emprunté, mais uniquement par l'abonné l'ayant réservé. La classe Réservé est supportée par la classe AutoAnnulation, qui comme son nom l'indique s'exécute dans un thread à part afin d'annuler la réservation automatiquement si celle-ci dure depuis plus de deux heures (durée définie par le sujet).

La classe Emprunté est l'état faisant suite de l'emprunt d'un document disponible, ou réservé par la personne demandant l'emprunt. Un document Emprunté ne peut être que retourner, ce qui a pour effet de retourner le document à un état Disponible.

1.4. Les certifications

1.4.1. « Geronimo » BretteSoft©

« Certains abonnés rendent les livres en retard (parfois avec un gros retard) ; d'autres dégradent les livres qu'ils empruntent ; un abonné, à la suite d'un retard de plus de deux semaines ou d'une dégradation de livre constatée au retour, sera interdit d'emprunt pendant 1 mois ».

Pour gérer les retards, on ajoute à l'état Emprunté une surcharge de Timer, la classe EmpruntMinuteur. Cette classe s'exécute dans un thread à part et, à partir de deux semaines de retard après la durée normale d'un emprunt, l'abonné emprunteur est sanctionné durant un mois. Pour de qui est des dégradations, le serveur demande l'état du document au client lors d'un retour, et sanctionne l'abonné si le document est dégradé.

1.4.2. « Cochise » BretteSoft©

« On ajoute des DVDs aux documents de cette bibliothèque (qui devient une médiathèque). Certains DVDs sont réservés aux plus de 12 er 16 ans. »

Cf. 1.2, les implémentations de document.

1.4.3. « Sitting Bull » BretteSoft©

« Lors d'une réservation, si le livre n'est pas disponible, on pourra proposer de placer une alerte mail nous avertissant du retour du livre. »

La classe AlertesMail répond à cette certification en mettant en place une *ConcurrentHashMap* contenant, pour un document donné, la liste des abonnés ayant placé une alerte dessus. Le placement d'une alerte se fait par le service d'emprunt et le service de réservation, lorsque l'emprunt/la réservation échoue pour cause d'indisponibilité. L'envoi des alertes mail, lui, est déclenché par le constructeur de l'état Disponible, et donc à chaque fois que le document redevient disponible.

2. COMMUNICATION SERVEUR-CLIENT

2.1. Sockets d'écoute

Au démarrage, l'application instancie 3 ConnectionListener sur les ports 2500, 2600 et 2700. Cette classe s'exécute dans un thread a part et permet d'écouter les connexions sur les ports entrants et de récupérer les sockets issus de ces connexions afin d'associer le client au service demandé.

2.2. Services rendus par l'application

La classe Service fait office d'interface de services concrets tout en encapsulant les flux d'entrée/sortie liés à la socket client. Cette encapsulation permet une communication avec le client plus intuitive à programmer dans les sous-classes.

La communication entre le client et le serveur se fait de manière cyclique, afin d'éviter un deadlock entre le processus client et le thread serveur associé à ce client. Le cycle à respecter pour le service est un cycle de la forme 'lire – écrire – lire – écrire – couper'. De ces trois fonctions, seule la fonction de lecture comporte un risque de deadlock si jamais le client cherche à lire le serveur au même moment. Pour cette exacte raison, la fonction lire, avant de lire directement depuis le serveur, flush le flux de sortie du serveur affin d'envoyer tous les messages en attente comme un seul bloc. La fonction couper, ou *endConnection*, flush elle aussi le flux de sortie afin de garantir que le client recevra tous les messages.

Cette simplification dans la communication a l'avantage de simplifier le code client aussi. En effet, de cette façon le client peut communiquer avec le serveur avec une simple alternance de 'écrire – lire'.