IUT Paris Descartes 2018-2019

Dossier de Projet

Programmation Serveur Java

Maud GELLEE - Gabriel GUISSET - Arsène LAPOSTOLET
Groupe 205

Sommaire

I/ Introduction

Au sein de ce projet, nous devons développer une application permettant de gérer les interactions d'une bibliothèque avec ses abonnés. Cette application a plusieurs services : Un service de réservation pour que l'abonné puisse mettre un document de côté le temps de venir le chercher ; Un service d'emprunt afin de pouvoir récupérer un livre réservé ou libre et un service de retour afin de gérer les retours des documents.

Les différents types de client pouvant interagir avec le logiciel sont : Les postes de la bibliothèque à partir desquelles nous pouvons emprunter ou retourner un livre et à partir des ordinateurs personnels des adhérents, nous pouvons uniquement réserver un livre.

II/ Ressources partagées

Les différentes ressources partagées sont :

La liste d'abonnés et la liste de documents : Afin de les protéger, nous avons utilisé la méthode statique appelée *synchronizedList()* de la classe Collections qui permet de construire une liste Thread-Safe depuis une factory. Ainsi, les différents Threads accéderont chacun à leur tour à ses listes.

Les instances de *Document :* Certaines fonctions de Document comme *reserver()* ont besoin d'être Thread-Safe. Par exemple, un document ne peut pas être réservé par deux personnes en même temps. Si deux instances d'Abonné font cette requête sur un même document, alors ces 2 requêtes doivent être traité l'une après l'autre afin de soulever une exception lors de la 2ème demande de type PasLibreException. C'est pourquoi nous avons utilisé des blocs de code *synchronized (this){ … }* pour verrouiller l'accès concurrent au livre courant grâce à son moniteur de Hoare. Nous avons essayé de réduire au maximum la taille de ces blocs.

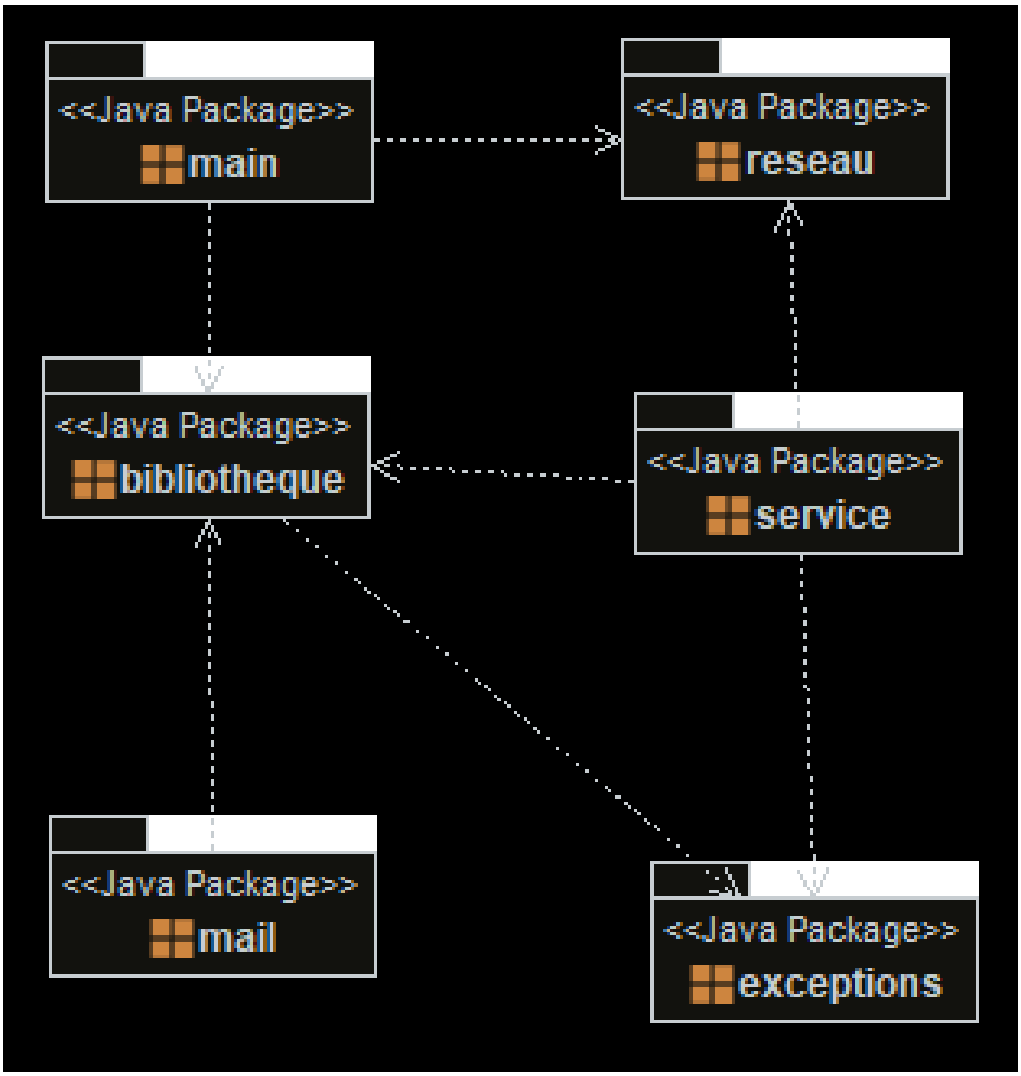
III/ Serveur

Côté serveur, nous avons mis en place une architecture robuste aux perspectives d'évolution grâce à de l'abstraction. Ainsi, le package de logique métier de la bibliothèque n'accuse aucune mauvaise dépendance. Nous avons également utilisé le patron de conception *Singleton* afin de pouvoir accéder à l'unique instance de la *Bibliothèque* dans l'application. Enfin, les dépendances vers les *ADocuments* se font via l'interface *Document*, et les comportements de base d'un document sont factorisés dans la classe abstraite *ADocument* ce qui permet de rajouter facilement de nouveaux types de documents.

Nous avons également introduit de l'abstraction dans les services réseaux, ce qui a permis de factoriser beaucoup de code, notamment pour la récupération des utilisateurs et documents auprès du client. De même, l'implémentation du patron de conception *Factory* assure la robustesse à l'ajout de nouveau de services réseaux.

Pour ce qui est des *TimerTask* , étant données que les implémentations de leurs méthodes *run()* sont très succinctes et qu'aucune autre fonctionnalité n'est utilisée, nous avons choisi de les déclarer à volée sous la forme de classes anonymes internes.

IV/ Diagramme de package



V/ Client

Nous avons créé une classe client qui permet de dialoguer avec le serveur de façon synchrone. Cette classe est appelée par une classe main qui prend deux arguments en ligne de commande afin de déterminer :

1) Le type de client (emprunt, retour, réservation)

2) L'adresse IP du serveur

Nous avons à cet effet inclut dans le projet des fichiers *.bat* qui contiennent les scripts pour lancer un *.jar* exécutable avec les bons paramètre (avec *localhost* en adresse de serveur).

VI/ Certification BretteSoft

Papoose eclairé

La première fonctionnalité de la certification BretteSoft© permet de bloquer des abonnés rendant leurs documents trop tard ou abîmés. Pour cette certification nous avons dû apporter quelques modifications au code d'Abonné et au Code de livre.

Dans *ADocument* nous avons ajouté un *Timer tEmprunt qui* permet de bloquer un *Abonne* si ce dernier n'a toujours pas rendu son *ADocument* après 1 mois d'emprunt. Ce *Timer* déclenche lorsqu'il se termine la *TimerTask taskBlocage (*que nous avons ajoutée dans *ADocument*) appelant la fonction *bloquer (TEMPS_BLOCAGE)* que nous avons définie dans *Abonné.*

Cette fonction permet de démarrer le *Timer tBlocage*, pendant une durée passée en paramètre, et de passer le booléen *bloque* d'*Abonné* à *true* pendant toute la durée du blocage. Une fois que le *Timer tBlocage* s'est terminé, cela signifie que le client a de nouveau le droit d'emprunter des *ADocument,* on passe donc le booléen *bloque* à *false.*

Pour ce faire nous avons stocké des constantes pour les durées de blocage en millisecondes. Elles sont calculées dynamiquement avec la fonction *pow()* pour une meilleure lisibilité.

Grand Chamann

Afin de valider l'ultime certification de ce projet, nous avons avant tout essayé de gérer les alertes liées à la disponibilité du *ADocument.* Nous avons d'abord initialisé une liste d'Abonne au sein de *ADocument*. Ainsi, les personnes ayant tenté de réserver un *ADocument* alors que celui-ci était déjà emprunté, peuvent ajouter une alerte afin d'être prévenues au retour du *ADocument*. Si un abonné place une alerte, il sera stocké dans la liste Abonné de ce *ADocument*.

Afin d'envoyer des notifications par mail, nous avons utilisé la bibliothèque javax.mail officielle disponible sur le site d'Oracle©. Nous avons opté pour le protocole SSL car il est sécurisé et présente une configuration moins lourde que le protocole TLS par exemple. Nous avons utilisé les services Gmail avec un compte créé spécialement pour ce projet. Vous pourrez donc tester cette fonctionnalité grâce à votre compte d'abonné numéro 4 (relié à votre adresse mail universitaire). Le mail reçu en cas de retour de *ADocument* et de demande d'alerte est au format HTML et adapté au contexte de la Brett'Othèque.

VII/ Perspective

Nous aurions pu, pour une meilleure qualité de code, mettre en place le patron de conception *State* pour modéliser les différents états d'un *ADocument*. Dès l'implémentation, de nouveaux types de documents auraient pu être réalisées même s'ils n'étaient pas dans le cahier des charges. Afin de faciliter cette implémentation, nous avons donc délégué les actions possibles par livre sur une classe abstraite appelée *ADocument*. Ainsi l'ajout d'un type de document se fait par la création d'une nouvelle classe héritant de *ADocument*.

VIII/ Conclusion

Pendant ce projet, nous avons découvert la programmation client-serveur. Nous avons ainsi pu expérimenter l'utilisation de nouveaux composants comme les threads et les sockets et de nouveaux concepts comme la communication synchrone et la thread safety. De même nous avons découvert de nouvelles fonctionnalités de Java concernant la planification de tâches grâce aux classes *Timer et TimerTask.*

Afin d'améliorer notre productivité et notre confort de travail collaboratif, nous avons utilisé le système de gestion de version Git. Nous avons donc utilisé l'hébergement gratuit framagit.org ainsi que le client graphique Fork afin de disposer d'un client avec une interface graphique.

D'un point de vue plus général, ce projet a été une très bonne expérience; il nous a permis de confirmer les compétences déjà acquises en Java et en programmation objet ainsi que d'en acquérir de nouvelles dans de bonnes conditions. Le sujet, à la fois ludique et intéressant, nous a permis de ressentir l'esprit du développeur. De plus, la création d'une application client-serveur était pour nous une expérience nouvelle, tout à fait enrichissante.

Enfin, l'aspect collaboratif de ce projet fait partie de ses points forts. En effet, la motivation mutuelle est un moteur puissant qui permet de surmonter tout type de difficulté. De plus, la résonnance de plusieurs esprits focalisés sur le même objectif, communiquant sur un problème, est une des formes les plus efficaces de réflexion. Chacun comblant les faiblesses des autres, nous avons pu nous tirer mutuellement vers le haut. Ayant déjà l'expérience du travail en commun, nous avons renforcé notre niveau de collaboration et cela nous a permis d'avancer encore plus, chacun connaissant les spécificités des autres.

Pour ce qui est des difficultés rencontrées (mis à part le manque de chaises en salle de TP), l'étape qui nous a demandé le plus d'effort est la synchronisation de la communication client-serveur de façon à ce qu'elle soit la plus portable possible.