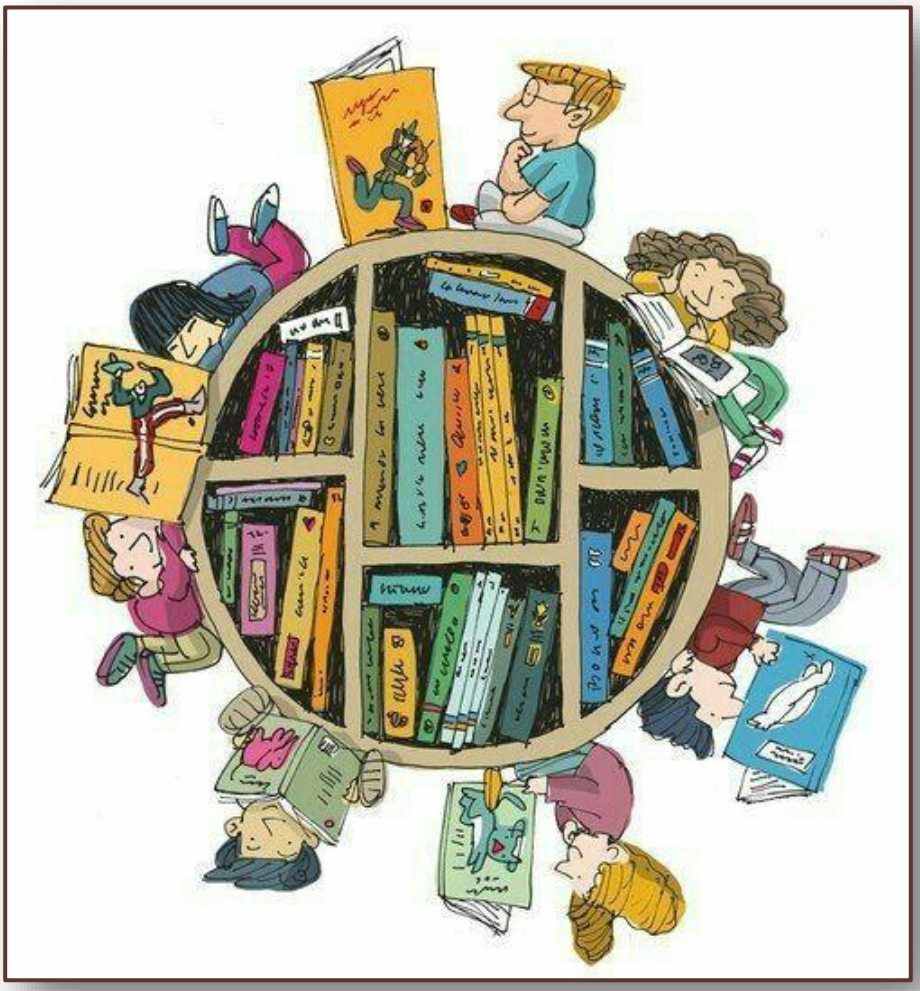
Université de Paris
DUT INFORMATIQUE
2020-2021



RESERVATION/EMPRUNT/RETOUR MEDIATHEQUE

Projet Application Serveur Java

Almuntaser BASHITI 209 - Roya GHARBI 203

Table des matières

Présentation du projet

Ce projet nous a permis de mettre en avant ce que nous avons appris durant toute la période B dans ce module. Pour cela, nous devions créer pour une médiathèque, un serveur, qui en fonction des services, effectuera différentes tâches. Les trois services demandés sont les services de réservation, d'emprunt et de retour des DVD. En fonction du port avec lequel le client se connectera, il pourra soit réserver un DVD pendant une durée définie c'est-à-dire pendant 2 heures soit emprunter un DVD soit retourner un DVD. Ils possèdent respectivement le port 3000, 4000 et 5000. Le service Emprunt devra vérifier certaines conditions pour pouvoir accorder la permission à l'abonné de l'emprunter. Si cet abonné a déjà réservé le DVD et qu'il n'a pas dépassé le délai, il pourra le prendre sinon il sera trop tard. Cependant, si celui-ci est toujours disponible, il pourra l'emporter avec lui s'il respecte bien les conditions. Le retour se fait juste avec le numéro du DVD.

Principes vus en cours

- Accès concurrentiel

Pour éviter un accès concurrentiel c'est-à-dire que plusieurs threads veulent accéder au même moment à la même ressource, nous avons choisi de mettre un *synchronized*. Prenons cet exemple :

```
if (abonne != null && Objet != null) {
        synchronized (Objet) {
            try {
                getObjetAEmprunter(IdObj).reservationPour(getAbonne(IdAb));
                out.println("Vous venez de réserver l'objet pendant deux heures. Pensez à l'emprunter!");
                Timer timer = new Timer();
                timer.schedule(getObjetAEmprunter(IdObj), 1000 * 60 * 120);
            } catch (ReservationException e) {
                out.println(e.getMessage());
            }
        }
}
```

Ici nous avons mis un *synchronized* la partie où nous réservons ou non l'objet pour vérifier qu'un seul thread puisse accéder à ce bloc de code. Ici nous évitons tout parallélisme car nous verrouillons la section critique. Imaginons que plusieurs abonnés souhaitent réserver un DVD sur le port 3000. Ils accèderont tous au même moment à la partie où nous vérifions si cet abonné répond ou non aux critères de réservation et si cet objet peut être réservé. S'ils souhaitent tous avoir le même objet mais qu'il y a qu'un seul DVD (objet) avec cet identifiant, il y aura une erreur car tout le monde souhaite le prendre et donc personne ne pourra le réserver. Cependant, dans notre cas, ils ne pourront pas accéder en même temps et nous vérifierons, un par un si l'abonné peut réserver ou non. Si oui, le DVD demandé ne sera plus disponible et les autres abonnés ne pourront plus le réserver. De plus, si nous ne mettons pas de *synchronized*, le timer se réinitialisera à chaque fois qu'un client exécute ce code car nous avons qu'un seul Timer.

- Communication Client/serveur

Pour permettre à la médiathèque de proposer ces services à ces abonnés, nous avons créé un serveur et plusieurs clients. Pour établir une connexion entre les deux, nous avons créé une classe « AppliServeur » où nous créons un serveur (en créant un thread avec

comme paramètre un nouveau serveur et qui s'attache à un numéro de port d'E/S). Dans la classe Serveur, nous attendons les demandes d'un client sur un port défini et lorsqu'un client se manifeste, en fonction de son numéro de port, un service sera créé avec comme paramètre la socket.accept(). Nous avons alors créé une socket entre le serveur et le client et donc accepter une connexion. Chaque service est un thread. Grâce aux services, nous pouvons avoir un échange personnaliser en fonction du numéro de port sur lequel on se situe. Donc grâce à la socket, nous pouvons envoyer et recevoir des données au client tout en codant dans la partie serveur. Pour pouvoir écrire ou lire des données, il faut utiliser respectivement les méthodes outputStream et inputStream avec comme paramètre la socket. Donc si on veut récupérer le numéro de l'abonné, il faut « lire » mais si on veut écrire dans la partie client il faut faire « out.println(« ... ») ; » avec « out » une variable défini auparavant et qui est égale à « new PrintWriter(client.getOutputStream(), true); » Après avoir fini le service, nous devons fermer la socket pour clore une connexion.

- Thread

Nous avons créé plusieurs threads pour avoir des processus « légers » . Nous avons donc un thread pour le serveur mais aussi pour chaque service. Dans chaque classe, nous implémentons un Runnable et nous créons une méthode « run » qui contient la ou les tâches à exécuter lorsque nous lançons le thread avec « new Thread(...).start() ; ». Prenons un exemple dans la classe « AppliServeur ». Ici, « *new Thread(new Server(tab)).start();*», nous créons un thread d'un serveur avec comme paramètre, le numéro de port d'E/S. Nous avons aussi un Timer qui est un Thread. Nous nous retrouvons avec deux types de Thread, des threads que nous créons et ceux déjà créer par java comme le Timer.

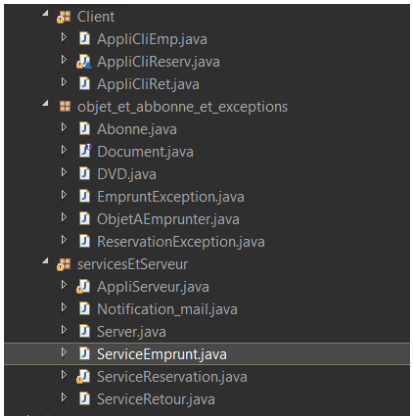
Découplage

- Séparation dépendances

Pour ce projet, il fallait créer comme objet des DVD. Cependant, comme indiquer dans l'énoncé, les types de documents peuvent évoluer et pourront contenir des livres, des CD… Nous avons alors comme classe mère l'interface « document » et comme classe fille « DVD » mais plus tard nous ajouterons comme classe fille « Livre »… Pour respecter le principe SOLID vu en cours de CPOAV, il ne faut pas tout coder dans la classe DVD car sinon plus tard, il va falloir tout coder à nouveau. Il faut toujours prendre en compte les futures maintenances et donc les changements. Pour cela, à partir des méthodes de l'interface « document » qui contient tout ce qui sera en commun entre les classes filles, nous avons créé une classe « ObjetAEmprunter » qui implémente « Document ». En @Override, nous avons codé ces méthodes. Nous avons codé tout ce qui pourra être en commun donc le fait de vérifier si l'objet est encore disponible, sinon il pourra le réserver/emprunter. La méthode DVD ne contient que ce qui lui est propre comme la limite d'âge. Nous reprenons ces méthodes en @Override et nous vérifions si en fonction de son âge, il peut réserver/emprunter. S'il a le droit, nous appelons les méthodes « super.reservationPour(ab); » ou « super.empruntPar (ab); » qui vérifieront si l'objet est toujours disponible.

- Package

Dans ce projet, nous avons mis dans un package les applications clients. On aurait pu le mettre dans un projet à part, rien n'aura changé. Nous avons mis dans un deuxième package tout ce qui est en rapport avec le serveur donc la classe serveur, appliServeur, les différents services et Notification_mail. Dans un troisième package, nous avons mis tous ce qui est en rapport avec les objets (ObjetAEmprunter, DVD…) et la classe Abonné.



Conclusion

Nos réussites

Nous avons réussi à finir tout le projet. Nous avons effectué tous les services demandés tout en respectant les contraintes. Prenons l'exemple de la contrainte d'âge. Toute personne souhaitant réserver ou emprunter un DVD mais que celui-ci n'est accessible que pour les plus de 16 ans, pourra ou non en fonction de son âge l'emprunter/réserver. Nos principales réussites sont selon nous les classes DVD et ObjetAEmprunter car nous n'avons pas pris énormément de temps à les coder. Les services aussi n'ont pas été très compliqué. De plus, coder la partie serveur, application serveur et application client n'étaient pas compliquées car nous avons déjà un exemple vu en TP.

Nos difficultés

Sur l'ensemble du projet, nous n'avons pas eu de grandes difficultés mises à part l'option où nous devons envoyer un mail à l'abonné si le DVD qu'il souhaite réserver est indisponible. Cette partie nous a pris un peu plus de temps car il fallait bien comprendre comment envoyer des mails.

Les éventuelles améliorations

Nous savons que notre code n'est pas parfait et qu'il faut encore plus l'optimisée car il y a de nombreuses redondances. La partie client n'est pas optimiser car nous répétons plusieurs fois « *line = sin.readLine(); System.out.println(line);* ». De plus, nous répéter les mêmes méthodes dans service emprunt et réservation. Nous pouvons améliorer cela en les mettant dans une classe commune.