Université du Québec à Chicoutimi

Département des sciences appliquées

Architecture des logiciels 2025 – 6GEI311

Rapport du laboratoire 2

Présenté à

Oussama Jebbar et

Mikaël Brassard

Par

Samuel Brassard, BRAS13020400

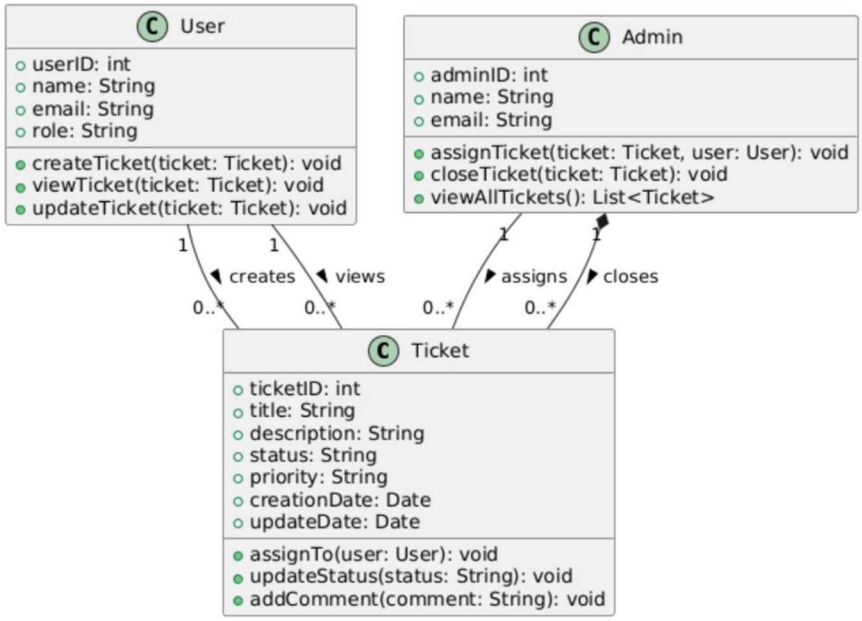
Il'aina Ratefinanahary, RATI24030207

08-10-2025

Table des matières

1. Partie 2 du laboratoire 2

1.1 Critique du diagramme de classe initiale



Le diagramme de classe original (ci-dessus) présente trois classes qui semblent à première vue être logiques cependant, il n'y a aucune encapsulation, les méthodes qui touchent au ticket se trouvent dans les classes admin et user. Il est logique de penser que l'utilisateur utilisera le ticket, cependant, le fait qu'une méthode modifiant un ticket se retrouve dans une autre classe que le ticket n'est pas une bonne pratique d'encapsulation. C'est une preuve de couplage fort ou de contenu, car les méthodes se relient directement les unes et les autres, ce qui est à éviter. D'une autre part, la cohésion est faible, étant donné que les méthodes d'une classe ne servent pas au même objectif, ce qui est aussi à éviter. Une autre preuve de cohésion faible, les méthodes peuvent facilement être divisées pour redistribuer les responsabilités, puisque, dans la partie 1, nous avions codé les vérifications reliées à chaque méthode directement dans celles-ci.

De plus, il y a beaucoup de redondance, par exemple, le ticket contient déjà la méthode assignTo, pourquoi on l'ajouterait à l'admin ? Dans la partie 1, nous avons ainsi procédé à l'appel direct de la méthode qui était dans ticket, pour simplifier le code, étant donné qu'elles font la même chose.

De plus, nous avons remarqué que l'utilisateur et l'admin se ressemblent beaucoup, ce qui veut dire qu'il y a sûrement une façon pour éviter cette redondance d'attributs.

1.2 Améliorations apportées

Au niveau du diagramme, nous avons voulu ajouter de l'encapsulation, étant donné qu'elle était peu présente au départ et nous avons aussi réduit la redondance. Du même coup, nous aurons une cohésion plus forte avec un couplage plus faible.

Premièrement, nous avons supprimé la redondance des attributs des utilisateurs et des admins en supprimant la classe admin. En effet, un admin sera simplement implémenté à l'aide de l'attribut rôle de la classe utilisateur. Une méthode dans la classe utilisateur permet de savoir si l'utilisateur est un admin et deux autres méthodes ajoutées, dont l'une permet de savoir si l'utilisateur peut assigner un ticket et l'autre s'il peut fermer un ticket.

Deuxièmement, la redondance des méthodes a été supprimée en intégrant des controller (de ticket, de statut, d'assignation, etc.) [GRASP], aussi appelé manager dans notre cas, qui, eux, contiennent l'ensemble des opérations qu'il est possible de faire sur un objet (modifier l'assignation, ajouter un commentaire, etc.) en s'assurant de faire les vérifications nécessaires, à moins qu'on appelle une méthode qui les contient déjà. Cela permet de contrôler l'extensibilité et d'améliorer la sécurité, car la classe manager permet de s'assurer du bon fonctionnement de la méthode. Le ticketManager est de loin le plus complet, logique, car il est le cœur du système de gestion de ticket. C'est lui qui contient la liste des tickets ainsi, dans le manager, on ajoute les tickets précréés (ce n'est pas sa tâche de créer) et on peut retrouver un ticket par son ticketId, son statut, l'utilisateur assigné, etc. On améliore beaucoup la modifiabilité, la maintenabilité et l'extensibilité en faisant cela, car il suffit maintenant d'appeler des méthodes dans cette classe pour tout ce qui a en rapport avec les tickets.

Par la suite, nous avons implémenté le concept de créateur du GRASP pour la création des tickets et des utilisateurs. Ainsi, cela améliore l'extensibilité en incluant toutes les vérifications avant création, sans les inclure directement dans la classe précise.

Ensuite, nous avions comme objectif général de respecter le Single Responsability Principle (SRP) [SOLID] qui augmente la modifiabilité, la maintenabilité et la cohésion. Cela justifie également les choix de conceptions mentionnés ci-dessus (creator, manager), mais également la création d'une classe pour faire les différentes validations (validator) pour chaque creator. On se retrouve alors dans une situation où les controller/manager n'ont presque aucune validation à faire.

Exemple de fonctionnement : pour créer un ticket, on passe par le ticketCreator, qui lui passera automatiquement par le validateur pour valider la création avant de créer le ticket et de le renvoyer. Si le manager de ticket avait été relié au créateur de ticket lors de la création du créateur (ce qui est le cas dans notre main actuelle), le créateur ajoute automatiquement le ticket dans la liste des tickets. Cependant, si ce n'est pas le cas, il faudra l'ajouter en utilisant le ticket manager (addTicket).

De plus, pour continuer dans notre principe SRP, nous avons ajouté une classe display, pour les différents affichages nécessaires. Cela va nous simplifier la vie pour le laboratoire 3, car nous allons devoir modifier uniquement et simplement cette classe pour faire l'affichage graphique demandée.

Finalement, la dernière modification qui a été faite est l'ajout d'une classe description, incluant, pour respecter nos principes de conceptions mentionnés juste avant, un manager et un validateur, et qui sera créée à partir du TicketCreator. Nous avons fait ce choix puisqu'il était demandé d'implémenter une description permettant d'intégrer des images, des vidéos et du texte, ce qui nous semblait beaucoup plus simple à faire à l'aide d'une nouvelle classe. Ainsi, l'attribut description du ticket est maintenant de classe Description.

Voici un tableau récapitulatif des justifications améliorations apportées :

Modification apportée	Principe(s) concerné(s)	Justification/Effet recherché
Suppression de la classe Admin (fusion avec Utilisateur via un attribut rôle).	Encapsulation, Réduction du couplage	Évite la redondance d'attributs et de méthodes entre Admin et Utilisateur. Centralise la logique liée aux permissions (ex. assigner/fermer un ticket).
Ajout de méthodes dans Utilisateur (isAdmin(), canAssignTicket(), canCloseTicket()).	Encapsulation, Cohésion forte	Clarifie les responsabilités de l'utilisateur et évite de dupliquer les vérifications dans d'autres classes.
Introduction des classes Manager/Controller (TicketManager, StatusManager, etc.).	GRASP – Controller, SRP (SOLID)	Centralise les opérations liées à chaque entité (ajout, modification, recherche). Améliore la maintenabilité, la sécurité et l'extensibilité.
Application du pattern Creator (GRASP) pour TicketCreator et UserCreator.	GRASP – Creator, Encapsulation, SRP (SOLID)	Gère la création d'objets avec toutes les validations nécessaires avant instanciation. Facilite l'ajout futur de nouveaux types d'objets.
Ajout des classes Validator pour chaque Creator.	SRP (SOLID), Cohésion forte	Sépare la logique de validation de la création. Réduis les dépendances et clarifie le rôle de chaque classe.
Lien entre TicketCreator et TicketManager (ajout automatique du ticket créé).	Faible couplage, Cohésion forte	Permets d'automatiser la gestion des tickets créés tout en laissant la flexibilité d'ajouter manuellement si nécessaire.

Création d'une classe Display pour la gestion des affichages.	SRP (SOLID)	Sépare complètement la logique de présentation de la logique métier. Simplifie l'évolution vers un affichage graphique futur.
Ajout de la classe Description (avec son Manager et Validator).	Encapsulation, SRP (SOLID), Extensibilité	Permets d'intégrer différents types de contenu (texte, images, vidéos) dans les tickets tout en gardant la logique de validation et de gestion bien structurée.
Réduction générale de la redondance et renforcement de l'encapsulation.	Cohésion forte/Couplage faible, Encapsulation	Facilite la maintenance et les modifications futures sans impacter l'ensemble du système.

1.3 Leçons apprises lors de ce laboratoire

Lors de ce laboratoire, nous avons pu perfectionner nos compétences en programmation Java, mais nous avons surtout pu mieux comprendre et appliquer les principes de conception orientée objet, tels que l'encapsulation, la cohésion, le couplage faible ainsi que les principes GRASP et SOLID. En effet, dans ce laboratoire, pour améliorer la conception du projet de base, nous avons dû faire appel à nos acquis pour créer un système plus extensible et maintenable, tout en assurant sa robustesse grâce à une meilleure organisation du code.