Université d'Alger1
Faculté Des Sciences
Département MI
Master 1 ISH
Module : Architecture des logiciels
2020-2021

TP2

Objectif. Apprendre à appliquer les principes SOLID et les patrons de conception afin d'amé1iorer la cohésion et le couplage.

Enoncé. Le système étudié dans ce TP est le système d'une bibliothèque numérique qui fournit des livres électroniques (e-book). Plusieurs universités mets ce système a la disposition de leurs étudiants. L'utilisateur (1'étudiant) peut se connecter au système en utilisant les paramètres donnés par 1'université et télécharger des livres protégés par mot de passe qu'i1 va lire en offline. Chaque livre téléchargé est accessible pour une durée limitée (le mot de passe qui protège le livre expire après cette durée en interdisant ainsi 1'acces au livre).

De plus, le nombre des livres que 1'étudiant peut télécharger est limité à un nombre fixe chaque mois (c'est le nombre de livre mensuel autorisés). Cela dépend du forfait (packageType) acheté par 1'université à laquelle appartient 1'étudiant qui peut être : limité standard, limité premium ou illimité. 11 est à noter que d'autre types de forfait peuvent s'ajouter au système selon le besoin.

Au début (à 1'inscription de 1'étudiant), le nombre de livre mensuel autorisé est égale à 10 quand le forfait est standard, a 20 quand le forfait est premium et à zéro quand le forfait est illimité. De plus, 1'étudiant avec un forfait limité peut bénéficier d'un bonus qui augmentera le nombre de livres mensuel autorisé. L'augmentation dépend du forfait de 1'université, on augmente par s quand le forfait est Standard et par 10 quand i1 est premium.

Pour faire ce TP vous devez récupérer le projet Eclipse « ArchitTP2 » et 1 améliorer en répondant aux questions posées dans les trois parties du TP.

Partie 1. Refactoring du code

1. Ouvrez le projet dans Eclipse
2. Créez un dépôt sur GitHub et envoyer votre projet vers ce dépôt. Apres cela et pour tout le reste du TP vous allez effectuer à chaque étape un commit et un Push pour mettre à jour votre code distant (via 1'invite de commande ou 1'interface de Eclipse). Mettez dans le message du commit le numéro du commit et un titre décrivant ce qui a été fait.

3. Donnez le diagramme de dépendance entre ces classes

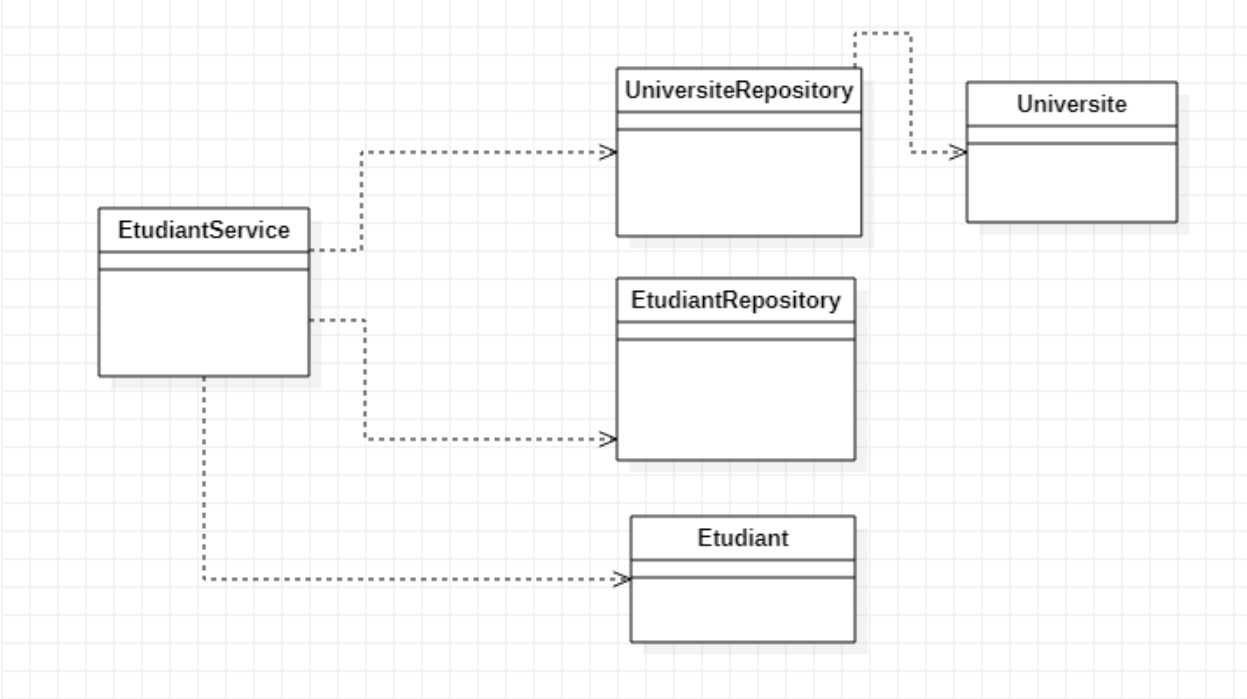

Figure1 : Diagramme de dépendance

4. Utilisez le patron singleton dans la classe « DBConnection » pour avoir un seul point d'accès a la base de données (une seule base de données).

s Appliquez le Commit et push pour mettre à jour le code sur GitHub

6. Appliquez le principe de 1'inversion de contrôle pour améliorer ce code. Expliquez 1'objectif de cette amélioration

- Les modules de haut niveau ne doivent pas dépendre de modules de bas niveau. Les deux devraient dépendre d'abstractions.
- Les abstractions ne doivent pas dépendre des détails. Les détails doivent dépendre des abstractions.
- En regardant notre exemple, il est évident que le EtudiantService fait partie d'un module de haut niveau et qu'il dépend du EtudiantRepository qui persiste les étudiants dans la base de données et appartient à un module de bas niveau. Il est également clair qu'aucun d'eux ne dépend d'abstractions.
- Nous mettons les deux classes derrière des interfaces (abstractions) et injectons l'abstraction de niveau inférieur, le nouveau IEtudiantRepository, dans le constructeur EtudiantService. Il s'agit d'injection de dépendance, mais c'est suffisant pour satisfaire le DIP.

- IEtudiantRepository est l'abstraction du détail EtudiantRepository. Afin de satisfaire à cette règle, tout changement dans le EtudiantRepository doit être déclenché par un changement dans le IEtudiantRepository. Tout ça pour éviter la collision entre les classes.

7. Modifiez le programme principal et exécutez
8. Appliquez le Commit et push pour mettre à jour le code sur GitHub

Dans cette application, la gestion du journal se fait par des affichages sur 1'écran, comme c'est le cas dans la méthode « inscription » de la classe « EtudiantService » et la méthode « add » de la classe «EtudiantRepository ». On souhaite extraire de ces classes les détails d'implémentation du journal. Pour cela nous utilisons une interface « IJournal » contenant une méthode abstraite avec la signature suivante void outPut Msg (String message). On veut créer des classes qui vont implémenter cette interface :

- Une classe qui permet d'afficher tous les messages directement sur 1'écran.
- Une classe qui permet d'afficher tous les messages sur un fichier.
- Une classe qui permet d'enrichir le message avec la date et la classe qui a généré ce message.
- Une classe qui permet d'afficher les messages sur différents types d'output en même temps, par exemple sur 1'écran, sur le fichier et sur d'autre si 1'on veut ajouter.

9. Implémenter ces classes et modifiez le code des classes qui utilisent le journal (pour la quatrième classe i1 s'agit d'un patron composite).
10. Modifie le programme principal afin d'afficher le journal sur 1'écran et sur un fichier en même temps avec un message détaillant la date et la classe.
11. Appliquez le Commit et push pour mettre à jour le code sur GitHub.

La méthode « inscription » de la classe « EtudiantService » est chargée d'ajouter un étudiant mais aussi elle vérifie le format de 1'emai1, vérifie 1'existence de 1'emai1 et du matricule et aussi elle initialise le nombre de livre mensuel autorisé.

12. Analysez chacune de ses responsabilités, puis décidez pour chacune si vous la gardez dans la méthode « inscription » ou 1'affectez à une autre classe.
13. Appliquez le Commit et push pour mettre à jour le code sur GitHub

L'étudiant peut bénéficier d'un bonus qui augmentera le nombre de livres mensuel autorisé comme expliqué plus haut.

14. Ajouter dans la classe « Etudiant » une méthode qui implémente ce besoin.

1s Ajouter dans la classe « EtudiantService » une méthode qui permet d'ajouter le bonus a tous les étudiants avec forfait limité standard et limité premium

16. Appliquez le Commit et push pour mettre à jour le code sur GitHub

A 1'initia1isation du nombre de livre mensuel autorisé ou à 1'ajout du bonus, les traitements dans les deux cas dépendent du forfait de 1'université à laquelle appartient 1'étudiant.

17. Analysez le code de ces deux fonctionnalités et expliquez le problème qui se trouve dans ce code.

- On ne peut pas avoir une sous-classe avec des préconditions plus fortes que celles de sa superclasse. (Les sous-types doivent être remplaçables pour leurs types principaux).
- AjouterBonus, qui obtient une liste d'étudiants (les critères ou la source n'est pas pertinent) et pour chacun ajoute une allocation de bonus. Parce que les étudiants standard ou premium peuvent obtenir un bonus, nous pouvons utiliser la classe abstraite Etudiant et, par polymorphisme, appeler AjouterBonus appropriée sur l'objet sous-jacent. Dans le contexte actuel, cela fonctionne bien et suit le LSP car je peux en toute sécurité remplacer EtudiantStandard ou EtudiantPremium (sous-types) par Etudiant (type de base).
- Nous pouvons remplacer en toute sécurité tout type d'étudiant par la classe de résumé Etudiant sans obtenir de résultats inattendus. En suivant la même logique, nous pouvons également remplacer EtudiantStandard et EtudiantPremium par EtudiantLimite.

18. Utilisez 1'héritage et 1'abstraction afin de proposer une solution (respectez le principe LSP).
19. Appliquez le Commit et push pour mettre à jour le code sur GitHub
20. Utilisez le patron Abstract Factory afin de finaliser votre solution
21. Appliquez le Commit et push pour mettre à jour le code sur GitHub

On souhaite organiser notre application selon le patron d'architecture MVC (Modèle/vue/contrôleur). Pour la partie vue de 1'application en utilise java Swing.

22. Implémenter la classe « ViewInscription » qui permet de gérer la présentation du formulaire d'inscription.
23. Implémenter une classe « ControleurInscription » qui permet de gérer le traitement lié à 1'inscription qui sont déclenchés par le formulaire d'inscription. Cette classe permet de recevoir les requêtes des utilisateurs et de les traiter en assurant la communication entre I le modèle et la vue.
24. Appliquez le Commit et push pour mettre à jour le code sur GitHub

2s. On souhaite que le contrôleur « ControleurInscription » dépend de 1'abstraction de la présentation et non pas de son implémentation. Comment peut-on réaliser ça ?

26. Ecrivez un nouveau programme principal «MainAppMvc» afin de tester l'app1ication. Ce programme permet la création des objets de l'app1ication et le lancement de la fenêtre d'inscription.
27. Appliquez le Commit et push pour mettre à jour le code sur GitHub.
28. Organisez vos classes en packages.
29. Donnez le diagramme de dépendance entre les packages.

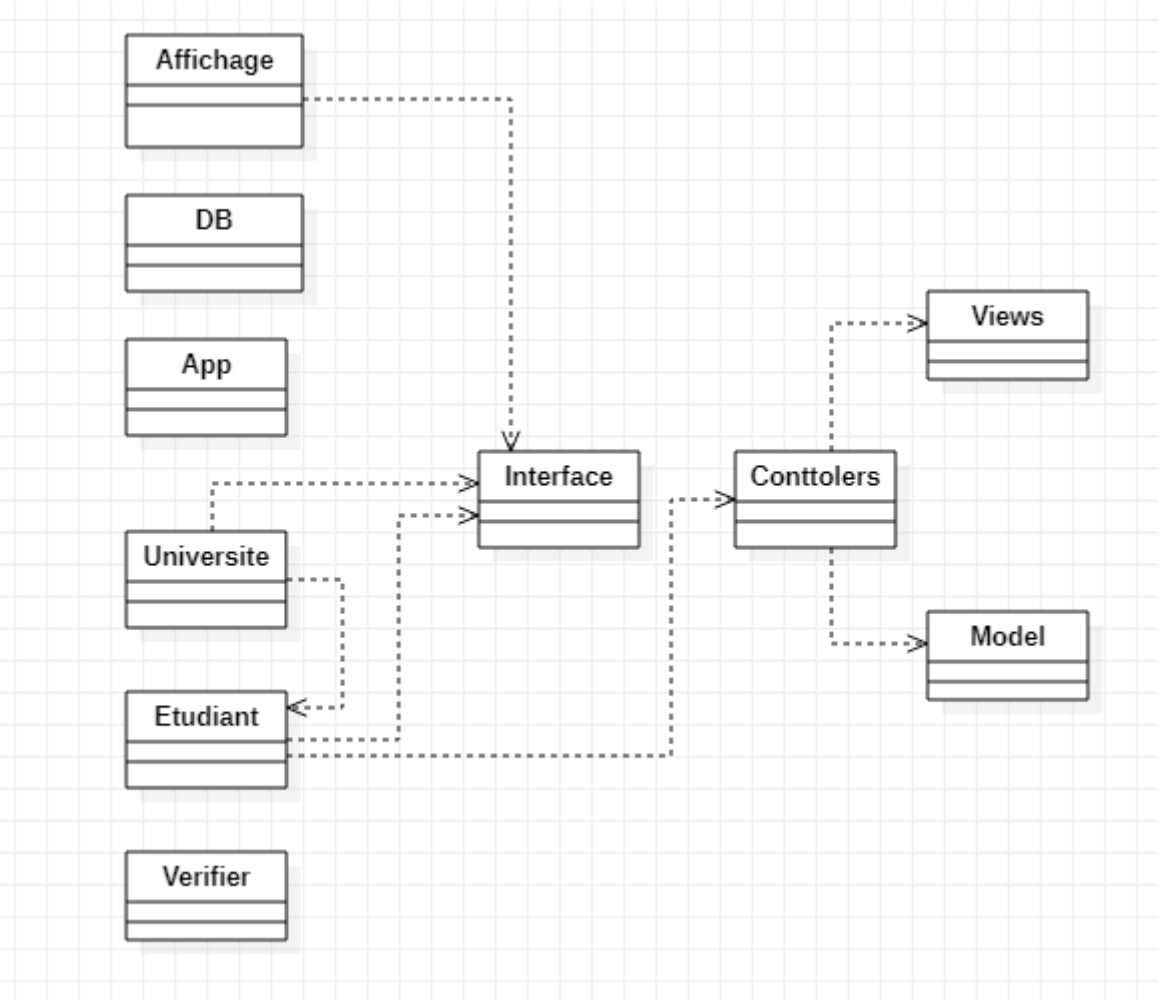

Figure2 : Diagramme de dépendance MVC

30. Tester la fonction d'inscription par JUnit.
31. Ajouter la gestion des exceptions à votre projet.
32. Appliquez le Commit et push pour mettre à jour le code sur GitHub.