1)

La classe EtudiantService dépend des classe EtudiantRepository, UniversiteRepository, Etudiant et Universite.

La classe EtudiantRepository dépend des classes DBConnection et Etudiant.

La classe UniversiteRepository dépend des classes DBConnection et Universite.

Schéma :

UniversiteRepository

EtudiantRepository

Etudiant

Universite

DBConnection

EtudiantService

4) le principe de l’inversion de contrôle :

- on ajoute une interface univ\_interface\_dependence qui contient la méthode GetById() de type universite.

- on ajoute les classes univ\_repo\_injection, etud\_repo\_injection et etud\_injection pour appliquer l’injection de dépendance dans la classe EtudiantService.

- on ajoute une interface etudaint\_interface qui contient les méthodes add, Exists.

- l’objectif de l’amélioration : rendre l’application plus stable, la classe EtudiantService dépendra de classe plus stable.

7)

- l’interface Ijournal contient la méthode outPut\_Msg (String message) sans body.

- la classe message\_composant qui contient la méthode outPut\_Msg (String message).

- la classe message\_composite qui implémente la méthode outPut\_Msg (String message) de la classe message\_composant (héritage) et contient la méthode add pour ajouter les instances des classes message\_ecran ,message\_fichier.

- la classe message\_ecran qui permet d’afficher tous les messages directement sur l’écran et implémente la méthode outPut\_Msg (String message) de la classe message\_composant (héritage).

- la classe message\_date qui permet d’enrichir le message avec la date et la classe qui a généré ce message et implémente la méthode outPut\_Msg (String message) de la classe message\_composant (héritage).

- la classe message\_fichier qui permet d’afficher tous les messages sur un fichier et implémente la méthode outPut\_Msg (String message) de la classe message\_composant (héritage).

- la classe App\_composite pour l’instanciation des classes message\_fichier et message\_date, avec la méthode setMessage () pour l’affichage tous les types d’output.

- pour ajouter une autre méthode pour un différend affichage on crée une classe qui hérite de message\_composant et en implémente sa méthode et on l’instancie dans la classe App\_composite pour l’ajouter à la liste pour afficher le contenue.

- à chaque println dans le programme on la remplace avec App\_composite.*setMessage()* .

10) on garde dans la méthode inscription les méthode Exists(email) et Exists(matricule) ils affectent la méthode inscription, et on crée une classe(verif\_pack) qui contient les méthodes d’initialise le nombre de livre mensuel autorisé.

15) le problème dans le cade : si on veut ajouter d’autre forfait on doit vérifier pour chaque type de forfait pour ajouter le nombre de livre mensuel autorisé.

26)

Le package add\_pack contient les classes Etudiant, TypePackage, Universite.

Le package composant\_pack contient les classes App\_composite, Ijournal(interface), message\_compasant, message\_compsite, message\_date, message\_ecran, message\_fichier.

Le package contreler\_main contient les classes ControleurInscription, MainApp, MainAppMvc.

Le package db\_repo\_pack contient les classes DBConnection, etudiant\_interface(interface), EtudiantRepository, univ\_interface\_dependence(interface), UniversiteRepository.

Le package service\_pack contient les classes etud\_injection, etud\_repo\_injection, EtudiantService, univ\_repo\_injection, verif\_pack.

Le package view contient la classe ViewInscription.

27)

Pack : service\_pack

Pack : contreler\_main

Pack : add\_pack

Pack : view

Pack : db\_repo\_pack

Pack : composant\_pack

29) les exceptions sont ajouter exemple : dans la classe ControleurInscription pour la récupération des donnés du formulaire, dans la classe testing de JUnit pour la methode test().