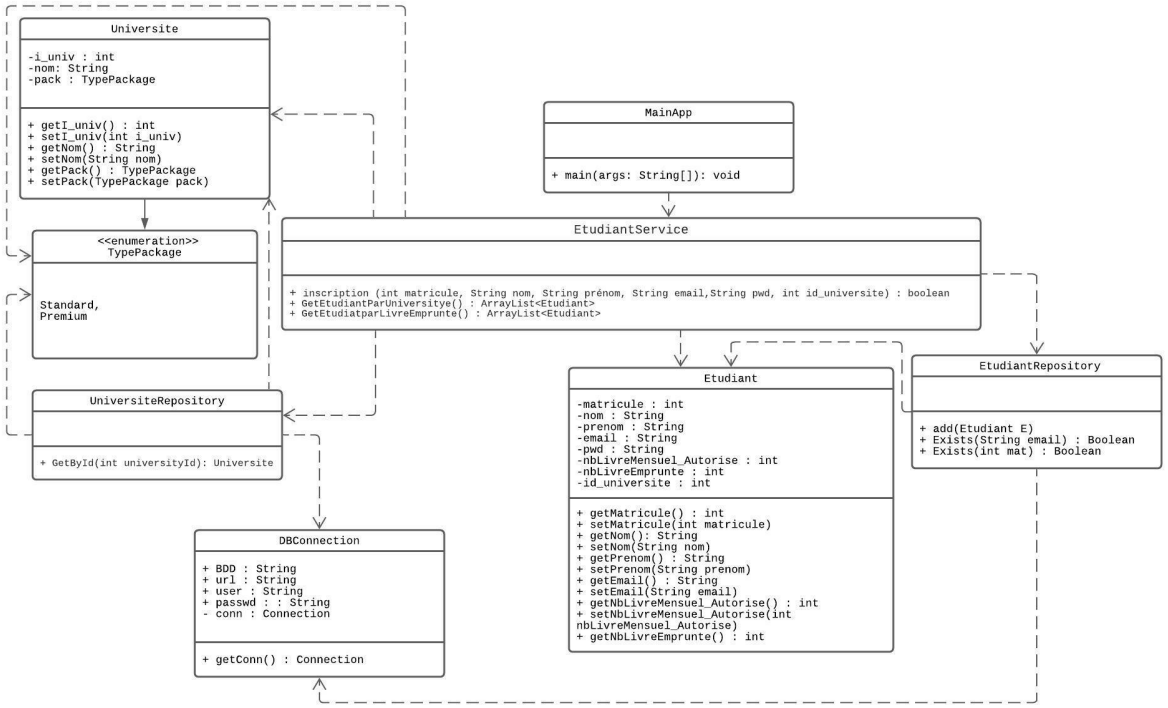
1 - Donnez le diagramme de dépendance entre ces classes :



10 - Analysez chacune de ses responsabilités, puis décidez pour chacune si vous la gardez dans la méthode « inscription » ou l'affectez à une autre classe.

Il convient de noter que la fonction possède de nombreux attributs des entrées, ainsi que des vérifications de ces entrées à l'intérieur, ce qui est très peu pratique car elle ne respecte pas le principe “S” single responsibility SOLID. À titre de suggestion, au lieu que les paramètres de fonction soient des attributs d'une classe, nous remplaçons ces derniers par un objet de cette classe. Suite à cela pour vérifier cet objet, nous utilisons une autre classe Registration. Elle permet aussi à initialiser nombre de livre mensuel autorisé d'un étudiant on passant comme entrée cet objet étudiant et son type de package.

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

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

Le problème qui ce pose c'est que la fonction n'est pas respecté principe O "Open for extension, Close for modification" car est condamné par des conditions.

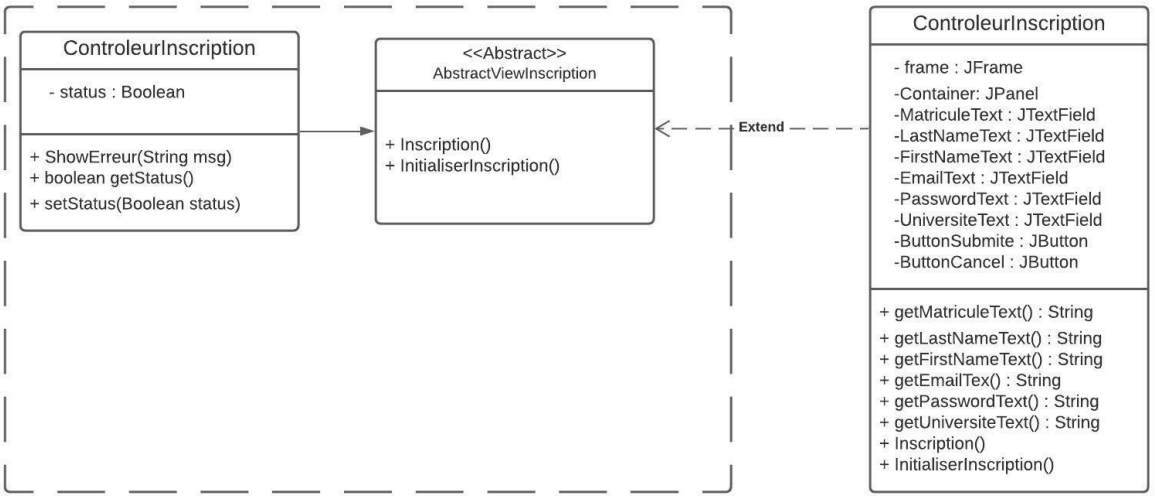
```
if (univ.getPack() == TypePackage.Standard)
            {
                 stud.setNbLivreMensuel_Autorise(10);
            }
            else if (univ.getPack() == TypePackage.Premium)
            {
               stud.setNbLivreMensuel_Autorise(10*2);
            }
```

Alors comme solution, on crée une interface et des classes concrètes qui vont implémenter des fonctions de cette classe. et retourner sa valeur correspond. Cette solution nous permet d'ajouter des conditions et des règles dans la classe **Registration** sans affecter la classe **EtudiantService.** Donc, on a résoudre le problème de principe O

```
public void setNbLivreMensuelAutorise(Etudiant E, TypePackage P){
      if(P == TypePackage.Standard){
          E.setNbLivreMensuel_Autorise(10);
      }
      if(P == TypePackage.Premium){
          E.setNbLivreMensuel_Autorise(20);
      }
  }
```

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

Afin de réaliser ça on a besoin d'ajouter une classe abstract **AbstractViewInscription** qui va être héritée par la classe **ViewInscription** et appliquer l'injection de dépendance dans la classe **ControleurInscription**.



27. Donnez le diagramme de dépendance entre les packages.