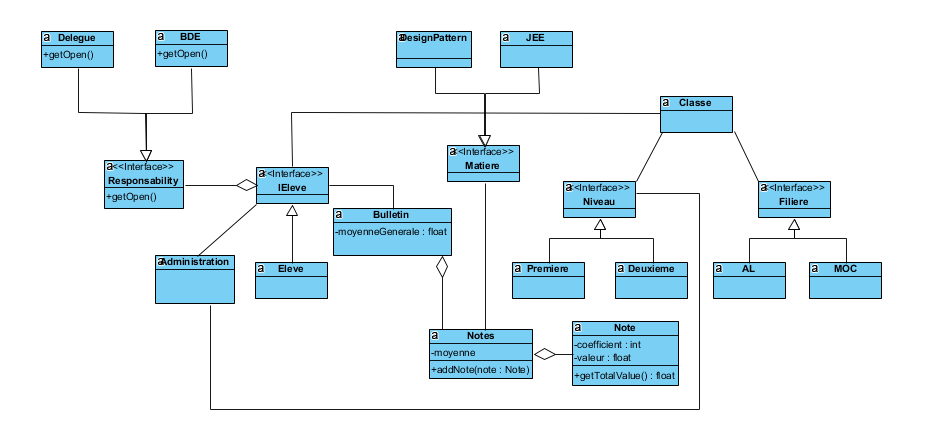
# 5Design Patterns

Nous avons choisi comme sujet le développement d’une application permettant de gérer une école avec leurs élèves, notes, classes, bulletins etc…

Pour notre implémentation nous n’avons créer que les classes de première et quatrième années.

Voici le diagramme de classe simplifié :

La première difficulté que nous avons rencontrée était notre focus trop important sur du CRUD ce qui ne permettait pas l’utilisation de pattern, nous avons donc essayé de créer des méthodes avec des règles de gestion plus complexes.

Pour représenter une classe nous devons créer un objet composé du niveau (4er année, etc.) et d’une filière (AL, MOC). Nous avons trouvé deux patterns utilisables, le « factory method » et le Builder.

La « factory methode » ne nous semble finalement pas pertinente étant donné que nous connaissons déjà tous les objets (niveau et filière) et nous ne créerons pas beaucoup d’objet, seulement deux. De plus la filière peut être optionnelle par exemple si l’on est en première ou seconde année, il y a seulement un tronc commun ; cela nous a convaincu d’utiliser le Builder plutôt que la « factory methode ».

Nous avons également remarqué que pour ne pas avoir à recréer de nombreuses instances de filières ou de niveaux lorsque l’on souhaite créer d’autres~~,~~ classes le pattern « FlyWeight » était parfaitement adapté à cette situation. Ce dernier permet de réutiliser des objets légers déjà existants.

Pour le calcul des moyennes lors de l’ajouter ou la suppression d’une note nous avions dans un premier temps pensé à utiliser le pattern « memento ». Cependant le memento ne permet pas la modification. C’est pourquoi nous avons finalement utilisé le pattern observé afin de pouvoir mettre à jour tous les objets qui dépendent de celui modifié et ainsi garder une cohérence.

En d’autres termes lorsque qu’une note est modifiée, ajoutée ou supprimée, la moyenne de cette matière est recalculée. Puis avec un second « observer » nous pouvons mettre à jour la moyenne générale.

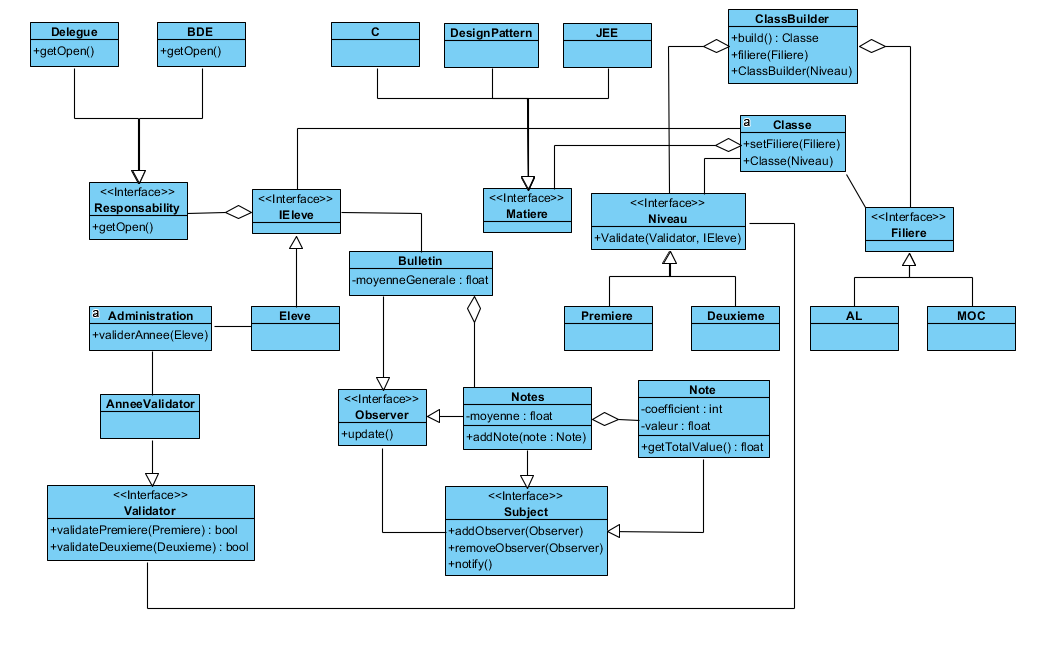
Il y a une fonctionnalité qui permet de savoir si un élève a validé son année scolaire. Cette méthode prend en compte plusieurs paramètres tel que les points Open et les notes.

Pour résoudre cela nous avons utilisé le pattern « Visiteur » car chaque niveau possède sa propre méthode de validation tout~~e~~ en séparant les données. Et cette implémentation reste ouverte à l’évolution du code. Par exemple nous avions penser rajouter par la suite un score de TOEIC minimum nécessaire qui augmente en fonction l’année

Nous avions également pensé à utiliser pattern « states » pour changer le comportement de l’objet quand son état change. Cela permet également de pouvoir rajouter des états pas la suite comme par exemple les rattrapages … Mais nous n’avions pas de réelle utilité pour le moment.

Puis nous avions besoin d’une méthode pour donner des points Open à l’élève. Ces points étaient donnés pour différentes raisons telles qu’être délégué, être au BDE ? ou dans une association.

De plus, chaque activité donnait un nombre de point propre. Nous avons pensé au « Décorateur », mais une fois le schéma réalisé nous nous sommes rendu compte que le « bridge » était sûrement plus adapté. En effet cela permet de découpler l’interface de la classe de son implémentation. Et une fois de plus cela permet de mieux structurer le code ainsi que respecter les règles du « clean code » avec un couplage faible ainsi que de rester ouvert à l’ajout.

Voici le diagramme complet avec tous les patterns :