MGL7460

Projet Base de donnée étudiante

François Hébert

Contexte

Pour mettre en pratique les connaissances acquises dans le cours MGL7460, on se propose de réaliser une application qui doit présenter les critères suivantes. Le projet doit pouvoir être testé automatiquement et la sauvegarde du code doit se faire via un logiciel de contrôle de version. L'application présente les contraintes suivantes. Le langage de programmation est Ruby. L'application doit utiliser une interface homme-machine par ligne de commande. L'ossature de l'application gérant l'interaction avec un utilisateur et l'organisation du code seront générés par le glem GLi. L'application doit pouvoir manipuler des données persistantes sous formes de base de donnée. Le paradigme de adopté pour le développement est celui de l'orienté objet.

Application: bde

On se propose de réaliser une application pour gérer une base de donnée étudiante. La persistance de la base de donnée sera faite sous forme de fichier texte. L'application nommé bde, pour **b**ase de **d**onnée **e**tudiant, présente des fonctionnalités restreintes de gestion de base de donnée. L'application doit pouvoir ajouter et ou retirer des cours, rajouter et ou retirer des étudiants à un cours, lister les cours et ou les étudiants d'un cours, afficher les évaluations d'un étudiant, calculer la moyenne d'un étudiant. Dans le paragraphe suivant nous décrirons plus précisément les commandes et leur format.

Commande

Nous ferons une description ici des commandes et options disponible pour notre application de gestion de base de donnée.

Manuel de l'application

```
NAME
    bde - Application pour gerer une base donnee etudiante

SYNOPSIS
    bde [global options] command [command options] [arguments...]
```

```
VERSION
    0.0.1

GLOBAL OPTIONS
    -f, --fichier=nom_fichier - Selectionner archive base donnee (default: .bde.txt)
    --help                    - Show this message
    --version                 - Display the program version

COMMANDS
    ajouter_cours    - Ajouter un cours
    ajouter_etudiant - Ajouter un eleve
    help             - Shows a list of commands or help for one command
    lister_cours     - Lister les cours
    lister_etudiants - Lister les eleves d'un cours
    moyenne          - Calculer la moyenne d'un eleve
    obtenir_eval     - Lister les evaluations
    retirer_cours    - Retirer un cours
    retirer_etudiant - Retirer un etudiant
    saisir_eval      - Saisir une evaluation pour un eleve
```

Option globale fichier

Cette option permet de sélectionner le fichier d'archivage de la base de donnée étudiante. Par défaut cette option est ".bde.txt".

Commande agissant sur les étudiants

Les commandes agissant sur les élèves utilisent des options supplémentaires pour spécifier le nom et le prénom de l'étudiant sur lequel on souhaite réaliser l'action, ainsi qu'une option de définition de la classe. Un exemple de ces commandes est donné plus bas.

```
NAME
    ajouter_etudiant - Ajouter un eleve

SYNOPSIS
    bde [global options] ajouter_etudiant [command options]

COMMAND OPTIONS
    -c, --class=sigle_cours - Selectionner le cours (default: none)
    -n, --nom=arg           - Specifier le nom de l'etudiant (default: none)
    -p, --prenom=arg        - Specifier le prenom de l'etudiant (default: none)
```

Architecture

Analyse fonctionnelle

L'application que nous souhaitons réaliser permet à l'utilisateur d'interagir sur 4 entités distinctes: l'application appelante, le gestionnaire de base de donnée, les cours et finalement les étudiants. L'application présente donc 4 composantes principales:

- Application appelante: cette élément présente l'interface à l'utilisateur, suivant la commande saisie, exécute la composante logique du gestionnaire de la base de donnée
- Gestionnaire base de donnée: cette composante a pour responsabilité de charger la base de donnée, de la modifier, de sélectionner les éléments pour traitement et finalement de sauvegarder la base de donnée. Cette composante peut être vu comme une collection de cours dans le cas de notre application.
- Cours: cette composante est l'élément constitutif de la collection de la base de donnée elle permet de gérer au niveau du cours les étudiants. La composante cours peut être assimilé à une collection d'étudiants.
- Etudiant: cette composante est l'élément constitutif du cours. Il est en charge de son information qui sont son état civil et L'historique de ses évaluations. Dans le

cadre de notre application il est aussi en charge du calcul de la moyenne pour le cours auquel il appartient.

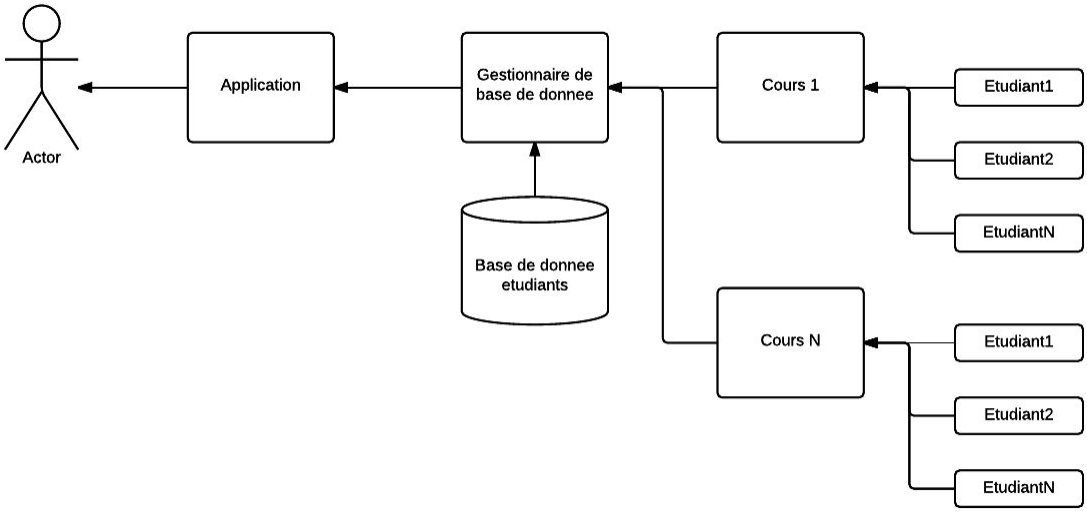
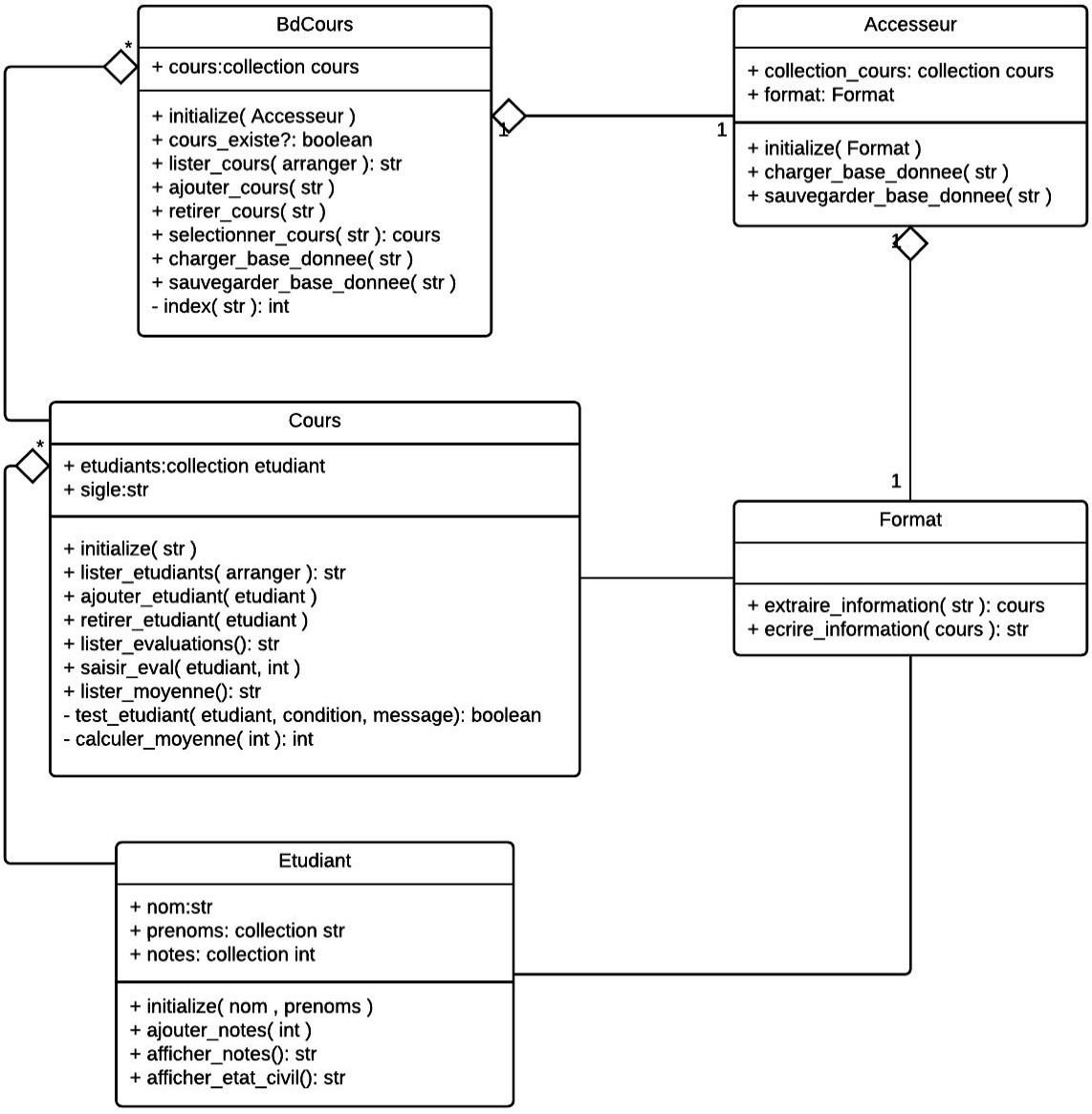


Diagramme de classes

Nous présentons le diagramme de classe de l'application bde. Pour le composant gestionnaire de base de donnée, nous avons délégué les responsabilités de la lecture et l'écriture de la base de donnée à un sous composant appelé Accesseur. La responsabilité du formattage des données lors de la lecture/écriture est elle même déléguée à une classe Format consommée par la classe Accessur. La délégation de ces responsabilités à des sous composants se justifie par le principe de séparation des responsabilités dans un paradigme orienté objet. Cette délégation rend plus robuste l'architecture à un changement de formattage des données.



Réalisation

Méthodologie

Pour le développement de l'application on adopte la méthodologie "Test Driven Development" comme décrite dans le livre "Growing Object-Oriented Software, Guided by Tests" de Steve Freeman et Nat Pryce. Les auteurs préconisent dans ce livre un

développement itératif ou on réalise d'abord un test d'acceptation devant échouer pour réaliser ensuite les tests unitaires des composants nécessaires à la réussite du test d'acceptation. Le schéma représente le cycle de développement tel qu'illustré dans le livre.

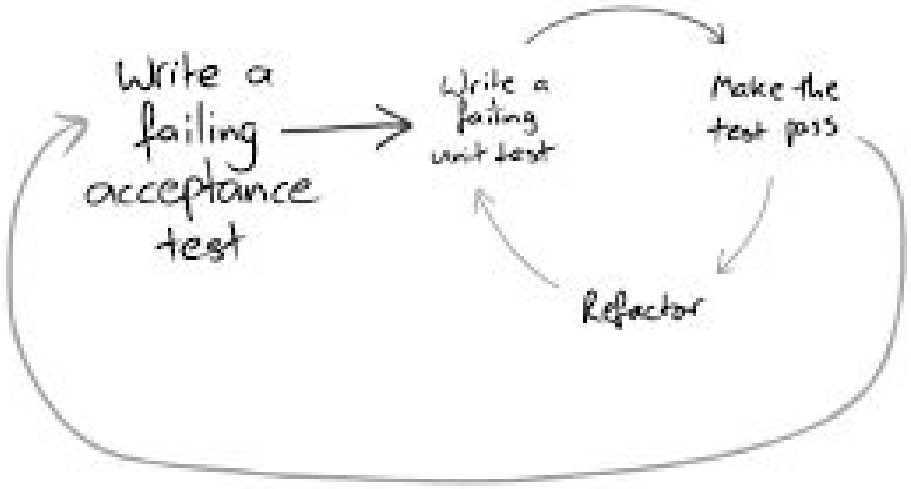


Figure 1: Schema TDD – from Growing Object-Oriented Software, Guided by Tests

Nous attacherons donc un soin particulier lors du codage à réaliser les tests d'application en premier puis le tests unitaires des composants et finalement le codage des fonctionnalités en dernier. Nous décrivons maintenant les responsabilités respectives des 2 types de tests employés dans ce projet.

- Test acceptation: ces tests s'assurent de la bonne marche des fonctionnalités offertes à l'utilisateur finale. Ces tests couvrent aussi l'usage non conforme des commandes. Ces tests doivent s'appliquer à l'application dans son entier et ne concernent nullement les composants qui constituent l'application.
- Test unitaire: ces tests ont pour objectif de vérifier le bon fonctionnement des composants logiciels de l'application. Ils vérifient que le composant remplit le contrat qui lui est assigné à la conception. Ces tests couvrent non seulement le bon fonctionnement mais dénoncent aussi les mauvais usage.

Technologie

- Système d'exploitation: linux Cinnamon mint 18.0 / Mac OS X
- Langage: Ruby 2.3.0
- Environnement développement intégré: Geany / Atom
- Librairie de test: Minitest avec Rtest // A verifier dans enonce
- Librairie d'organisation du code: GLI version
- Systeme de controle de version: Git
- Depot de code: https://github.com/woim/MGL7460_BDE.git

On trouvera en appendice l'arbre d'organisation du code et de l'application.

Retour sur expérience

Méthodologie TDD

La méthode force le concepteur à bien définir ce que l'on souhaite tester, il faut donc isoler cette fonction dans la composante testée pour obtenir des tests pertinents. L'approche est bénéfique dans la mesure où elle aide à mieux distinguer les responsabilités des composants et donc obtenir une meilleure conception en clarifiant la séparation des responsabilités. Cela demande une certaine discipline de la part du développeur. Ce dernier doit rester focaliser sur la fonctionnalité testée et ne pas se disperser sur les fonctionnalités qui restent à développer. Le premier test d'acceptation est certainement le plus long car il implique la mise en place de beaucoup des composants de l'application et des tests afférant. La première fois que l'on passe le test d'acceptation est une étape importante car à partir de ce moment précis le test d'acceptation devient aussi un test de régression. Le développeur peut alors améliorer son code tout en le comparant à une version qui sait qui a fonctionné, montrant ainsi l'importance des systèmes de contrôle de version. Ma critique sur cette méthode se réfère à son essence même à savoir isoler les fonctionnalités et les tester séparément. Dans le cas des logiciels très souvent le tout n'est pas la somme des parties impliquant

que toutes les fonctionnalités peuvent passer des tests mais que l'application dans sa globalité ne fait pas ce que l'on attend d'elle.

Langage Ruby

Le langage Ruby est un langage très souple et versatile, comparé au langage python il a l'avantage de respecter l'encapsulation des données comme devraient le faire tous les langages orientés objet. Il possède des librairies standard avec nombre de méthodes qui rendent le codage facile et rapide. La souplesse et la syntaxe du langage le rende parfois un peu abscons, me rappelant par moment Pearl. On peut synthétiser le code pour le rendre minimaliste mais on y perd de la lisibilité. Pour ce projet un temps non négligeable a dû être consacré à l'apprentissage du langage et à ses particularités. De façon générale j'ai eu plaisir à apprendre un nouveau langage, j'ai trouvé la flexibilité et la puissance du langage un de ses principaux atouts, par contre sa puissance peut lui nuire en rendant le code illisible et intelligible juste par des experts Ruby.

Librairie GLI

Ce gem aide le développeur à créer et organiser une ossature d'application claire et propre pour le développement. Il permet la mise en place des outils de test automatique par l'intermédiaire de rake et de minitest. On pourrait reprocher le manque de documentation claire pour assister le développeur, mais je reconnais que la clarté de l'organisation à démarrer sur le bon pied. J'ai éprouvé plus de mal à saisir la philosophie d'une interface à la Git. Comme Ruby un temps non négligeable a été consacré à l'apprentissage de GLI et de la philosophie d'interface. Dans les 2 cas le temps dédié à l'apprentissage fait sentir qu'on se focalise sur la forme et pas le fond qui est la bonne pratique dans la réalisation et la maintenance de logiciel.