Rapport final du projet de PCOO

Quentin Garnier (L3 I)
Léa Bloom (L3 MI)
Miléna Kostov (L3 MI)

Professeur : M. Julien Provillard

SOMMAIRE

Introduction

Dans le cadre d'un projet en Programmation et Conception Orientée Objet, nous avons dû réaliser une application permettant la gestion de notes de différents étudiants. Le langage utilisé est Java et nous avons principalement travaillé sur IntelliJ.

I- Fonctionnalités implémentées

En accord avec les attentes du projet, nous avons implémenté les fonctionnalités suivantes :

- L'application ouvre une fenêtre graphique possédant un menu ainsi que 3 onglets. Il est possible de redimensionner la fenêtre.

Le menu comporte deux options : "Sauvegarder" (enregistre les modifications apportées, plus de détails ci-dessous) et "Quitter" (ferme l'application de la même manière qu'en cliquant sur la croix en haut à droite).

- Lorsque l'application se lance, elle **charge les données initiales depuis un fichier xml** (par défaut si l'application est lancée via IntelliJ ou par ligne de code sans argument, le chemin est "data/data.xml"). Cela lui permet de créer les objets de type "Student", "Grade" etc. qui seront utilisés par la suite. **Les moyennes, notes max et min et les écart-types sont calculés à ce moment** (utilisées dans l'onglet 1).

- Dans le premier onglet "Programmes", on peut **visualiser les différents programmes existants** et les **exporter en format CSV**, afin de produire un procès-verbal. Le fichier est enregistré à l'adresse "output/___.csv". Il est également possible de n'**afficher que certains blocs** grâce à un système de filtre avec des *checkboxes*, toutefois cette fonctionnalité n'est pas totalement opérationnelle (voir partie IV).

- Dans le second onglet "Étudiants", on peut **visualiser chaque étudiant spécifiquement** afin d'afficher l'ensemble de ses notes. On peut **ajouter ou supprimer un étudiant**, ainsi qu'**ajouter, supprimer ou modifier les notes** d'un étudiant, et tous les calculs où la note intervient sont instantanément refaits. L'actualisation de la JTable est immédiate.

- Dans le troisième onglet "Organisation des cours", on peut **visualiser la hiérarchisation** des différents cours et programmes existants. Il est également possible d'**ajouter des programmes ou des cours** (avec choix du type de cours (bloc composite, bloc à option ou cours normal ; dans ce dernier cas, on peut le placer dans un bloc existant ou en tant que nouveau bloc simple).

- Enfin, il est possible d'enregistrer les modifications apportées lors de l'utilisation de l'application grâce à l'option "Sauvegarder" dans le menu. En cliquant dessus, **toutes les modifications sont sauvegardées dans le fichier xml** renseigné en entrée, au lancement de l'application (par défaut "data/data.xml"). De plus, si l'on tente de fermer l'application après avoir ajouté, modifié ou supprimé des éléments (étudiants, notes, programmes ou cours) mais sans avoir sauvegardé, l'application avertit l'utilisateur et lui demande s'il souhaite enregistrer avant de quitter.

En résumé, toutes les fonctionnalités demandées dans le sujet du projet ont été implémentées, sauf le filtre par blocs des programmes qui n'a pas pu être terminé (voir partie IV).

II- Organisation du travail et contributions

Afin de travailler de manière efficace, nous avons essentiellement utilisé deux outils pour la gestion de projet : **Discord**, pour la communication et les réunions de projet, et **GitHub**, pour la gestion de projet collaborative.
Sur GitHub, le projet a été découpé en plusieurs *Milestones* (une par Jalon) ainsi qu'en *Issues*, permettant ainsi de traiter les différents aspects à implémenter ou corriger le plus efficacement possible.
Nous avons également utilisé **GitHub Desktop** pour effectuer les différents *commits*.

Pour coder, nous avons utilisé le logiciel **IntelliJ** et avons créé un projet Maven. Les classes ont toutes été réparties dans des packages afin d'avoir une bonne organisation.
Nous avons procédé étape par étape, en commençant par le Jalon 1 avant de faire l'interface graphique requise pour les Jalons 2 et 3. Afin de bien réfléchir à la disposition des éléments dans le JFrame, il a fallu imaginer le rendu de manière globale ; c'est pourquoi les boutons (requis au Jalon 3) ont été ajoutés dès l'affichage demandé au Jalon 2.

Nous avons réparti le travail de la manière suivante : Quentin a implémenté le projet en créant les classes de base (Student, Grade, Block…) et fut chargé de l'écriture en format CSV des données, tandis que Léa et Miléna ont commencé par la lecture de fichier XML.
Par la suite, Quentin s'est occupé intégralement de la création de l'interface graphique en implémentant les fonctionnalités attendues, à savoir la visualisation des programmes et des étudiants ainsi que la hiérarchisation des cours, puis ensuite de la gestion des notes, étudiants, programmes et cours ainsi que la sauvegarde des modifications dans le fichier xml.

III- Choix de conception réalisés

La création du projet a nécessité des choix, et notamment dans l'affichage graphique de l'application.
C'est ainsi que nous avons décidé de séparer les différentes parties dans trois onglets distincts afin de clarifier les pages en évitant de les surcharger de boutons.

Le premier onglet est réservé pour les programmes, avec la possibilité d'exporter en format CSV ; c'est l'onglet le plus judicieux pour le faire, l'affichage des notes étant semblable au fichier produit. On ne peut rien y modifier dessus, juste visualiser (ainsi que filtrer avec des *checkboxes*, une par bloc). Une fenêtre d'avertissement apparaît lors de l'exportation en format CSV pour confirmer la réussite de l'opération à l'utilisateur.

Le second, pour les étudiants, possède donc des boutons pour ajouter/supprimer les étudiants ainsi que des boutons pour gérer leurs notes. Le choix de la gestion de notes ici a été effectué car il est

possible pour un étudiant d'avoir des notes dans des cours qui ne sont pas dans le programme qu'il suit.
La sélection de l'étudiant courant est actualisée intelligemment lors de l'ajout ou de la suppression d'un étudiant afin d'améliorer l'ergonomie de l'application.
De la même manière, il y a des fenêtres d'avertissement qui s'affichent à chaque opération interdite (champs vides, valeur incorrecte…) ou, parfois, lorsqu'une action a été effectuée avec succès (lors de la sauvegarde via le menu par exemple).

Le troisième onglet permet de visualiser toute la hiérarchie des programmes. Chaque bloc dans un programme est décalé, et chaque cours dans un bloc l'est aussi ; cela montre rapidement l'organisation des cours au sein des programmes.
Étant un onglet réservé aux programmes et cours, il semblait également judicieux d'y intégrer les boutons d'ajout de cours/programme ; l'actualisation se fait, ici encore, instantanément.

Le choix d'instaurer une barre de menu s'est fait assez vite, cela permettant à l'utilisateur de sauvegarder à tout moment les modifications apportées (et accessoirement de quitter s'il le désire). Les couleurs du menu et des boutons ont été choisies pour améliorer le design, afin de briser la monotonie d'une application intégralement blanche.

En ce qui concerne les choix lors de l'implémentation des classes, ils ont été assez vite décidés. Les packages ont été organisés afin de regrouper les classes fortement liées entre elles (par exemple les éléments graphiques, ou les classes Block et ses classes héritières).

IV- Difficultés rencontrées, limitations et bugs connus

1. Les difficultés rencontrées

Les difficultés rencontrées par Quentin : Mes principales difficultés auront été tout d'abord la prise en main des différentes fonctions de swing pour afficher les composants (notamment les *JTables*), jusqu'à ce que je prenne bien en main les différents éléments. Ensuite, ma plus grande difficulté aura été le tri des blocs dans le premier onglet, que je n'aurais finalement pas réussi à implémenter : j'avais des soucis récurrents de *ArrayOutOfBoundException* lors du tri des blocs composites et à options.

Les difficultés rencontrées par Léa : Tout d'abord, nous avons pris connaissance du sujet assez tôt. Au départ, nous étions déjà trois dans le groupe. Donc avec notre premier trinôme, nous avons déjà fait des appels pour s'organiser un peu. Nous avions déjà travaillé sur les classes de bases et avons réussi à avoir les classes principales correctes. Ensuite, notre trinôme s'est désisté et a quitté le groupe ce qui nous a retardé. Après donc, nous avons fait le groupe avec Quentin et de là nous avons commencé à travailler ensemble.
Par rapport aux problèmes que j'ai rencontrés : nous avons choisi de travailler sur l'exploitation, la lecture et la sauvegarde du fichier XML. Et ceci s'est révélé beaucoup plus complexe que ce que j'avais pensé. Notamment j'ai eu des problèmes avec le stockage des données (est-ce qu'il fallait les stocker dans un dictionnaire ? Dans une liste ?). Finalement, on a choisi les *ArrayList* pour leur utilisation facile. Ensuite s'est posé le problème des sous-nœuds, comment faisait-on pour y accéder ? J'ai donc fait plusieurs tests pour essayer de récupérer : deux boucles *for*, *hasChildNodes()*… et

nous avons finalement utilisé les fonctions connues du prof. Et finalement nous avons abouti à une classe compilable et qui marche avec les résultats voulus. Durant cette partie, j'ai travaillé en parallèle avec Miléna où nous échangions régulièrement sur Discord.
Ensuite pour l'interface graphique, j'ai réfléchi au meilleur moyen d'afficher la fenêtre en m'aidant d'internet et des cours.

Les difficultés rencontrées par Miléna : Tout d'abord un membre du premier trinôme s'est désisté (abandon projet) après que l'on ait mis en place une organisation de travail et commencé à travailler sur le projet, ce qui nous a fait perdre du temps. Nous avions commencé à réfléchir et à créer les classes de base. Par la suite le trinôme actuel s'est formé.
Pour le fichier XML les principales difficultés ont été de pouvoir récupérer correctement les données contenues dans les sous-noeuds (les différentes notes, les différents types de blocs) et de correctement stocker les données obtenues. Plusieurs méthodes (*hasChildNodes()*, boucles *for* etc) ont été testées avant de réussir grâce à des *ArrayList*. Ce travail a été effectué en parallèle avec Léa et les échanges sur Discord ont été réguliers et fréquents.
Enfin, j'ai réfléchi pour l'affichage d'onglets supplémentaires en consultant Internet ainsi que mes cours.

2. Les limitations du programme

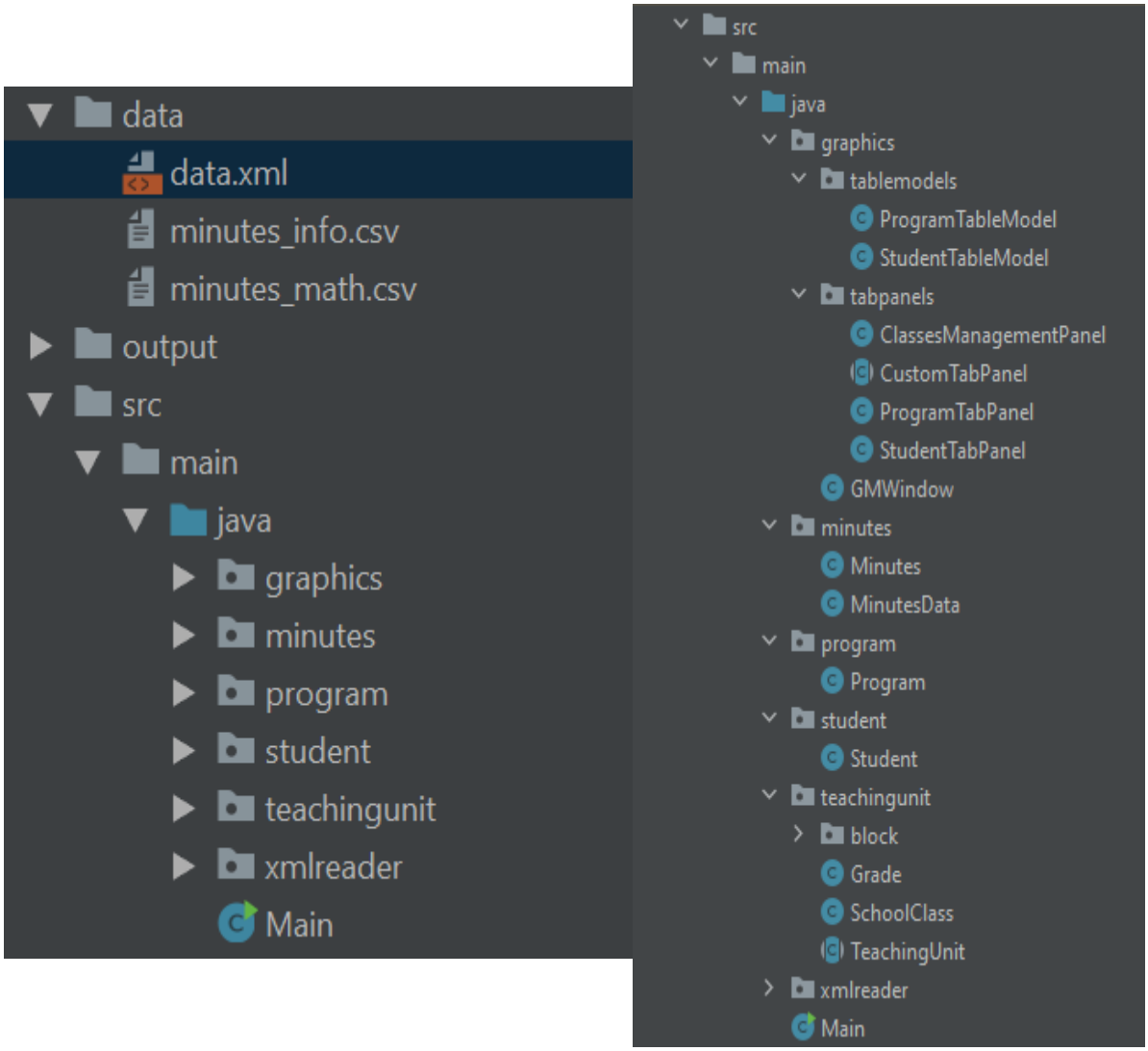
Notre programme connaît encore quelques limites, notamment sur le tri par bloc des programmes dans le premier onglet : en effet, bien que les *checkboxes* fonctionnent correctement et que le header du tableau agisse comme désiré, les notes quant à elles ne suivent pas et ne correspondent plus correctement aux cours indiqués.
Malgré de nombreux efforts, nous n'avons pas eu le temps de réussir à finir le filtrage (une version du code filtrait correctement les blocs simples mais de nombreuses erreurs apparaissaient lors du filtrage des blocs composites ou à options).

3. Les bugs connus

Il n'y a actuellement aucun bug connu sur notre application.

V- Modalités techniques pour lancer l'application

Afin de lancer l'application, il faut exécuter le Main.main.
Attention : l'application doit pouvoir accéder à un fichier xml de données ; par défaut, si aucun argument n'est renseigné, l'application tentera d'ouvrir à l'adresse "data/data.xml" (voir images ci-dessous) ; sinon, l'application prendra le premier argument donné comme chemin d'accès vers le fichier source. Tout sera enregistré dans ce même fichier lorsque l'on sauvegarde.



Arborescence du projet (à gauche, emplacement par défaut où placer le data.xml)

VI- Améliorations futures

Avec ce que nous avons déjà, et si de plus dans le fichier data.xml il y avait le contenu des cours, nous aurions pu ajouter des liens hypertextes sur chaque bloc de chaque programme afin d'accéder à son contenu en cliquant dessus (ce qui aurait correspondu à la partie « Pédagogie » sur Moodle).
Dans l'onglet « Étudiants » dans l'interface graphique, quand on recherche un étudiant, on pourrait implémenter une barre de recherche en plus (ou à la place) du menu déroulant. Ainsi, on pourrait taper la première lettre du prénom de l'étudiant et ainsi voir afficher une liste déroulante pour proposer tous étudiants correspondants (utilisation des expressions régulières).
Nous pouvons également essayer d'harmoniser la présentation des notes (même nombre de chiffres après la virgule).

Il serait aussi possible d'améliorer la présentation des programmes dans le troisième onglet (en faisant un affichage dynamique de la hiérarchie afin de pouvoir réduire des catégories (programmes ou blocs).

ANNEXE

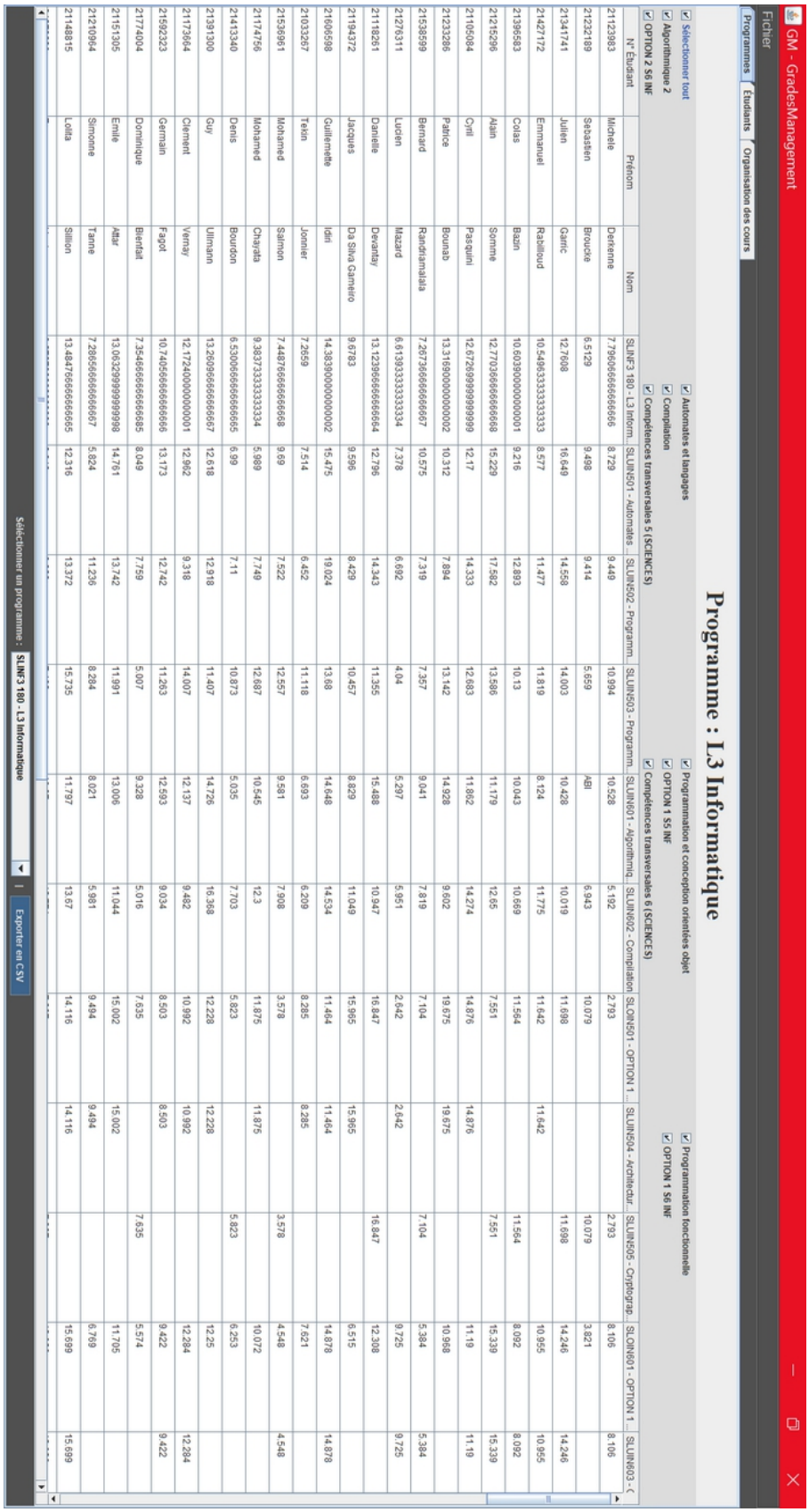


Image 1 : Aperçu de l'onglet « Programmes »

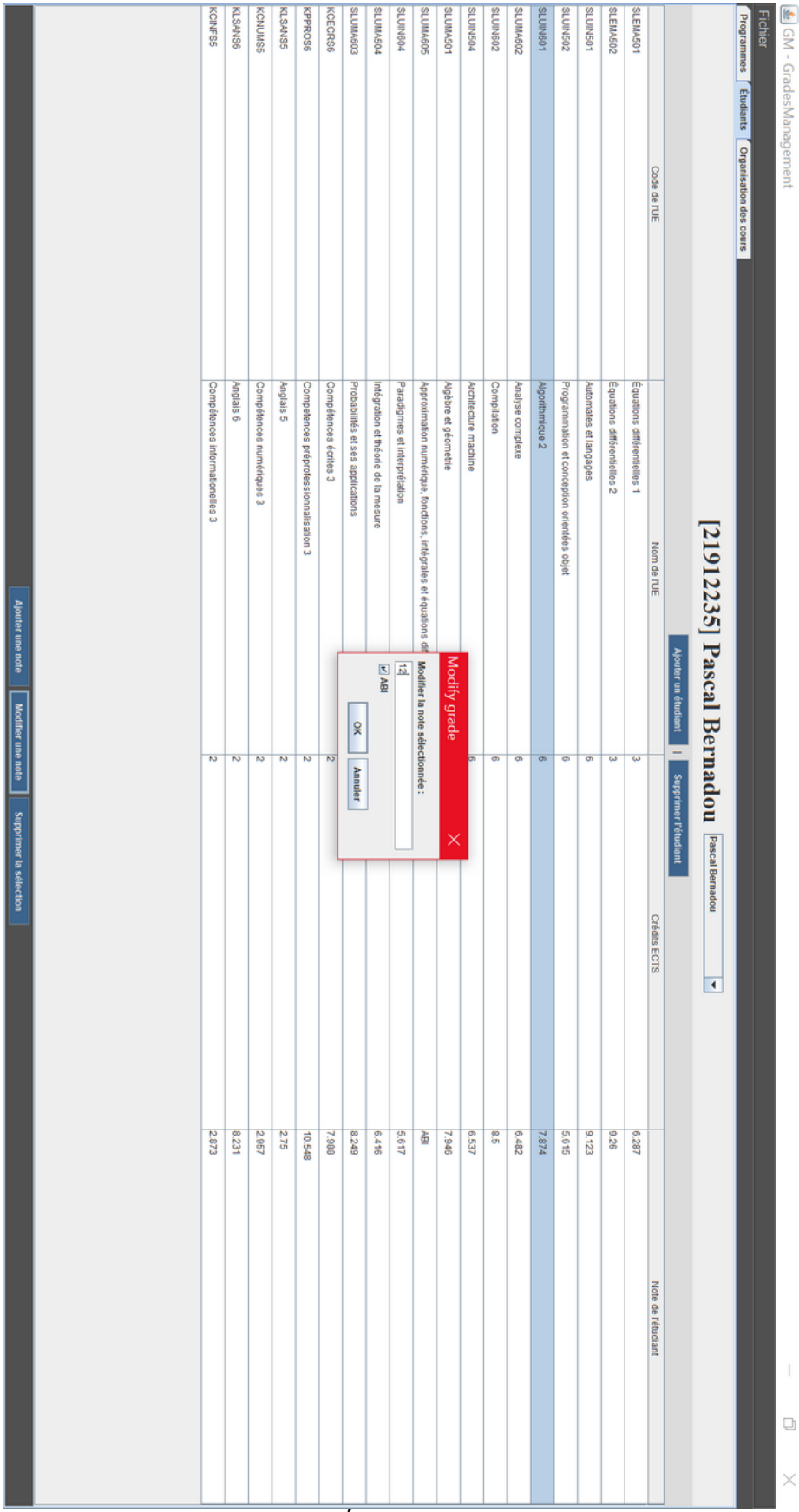


Image 2 : Aperçu de l'onglet « Étudiants » (avec modification d'une note)

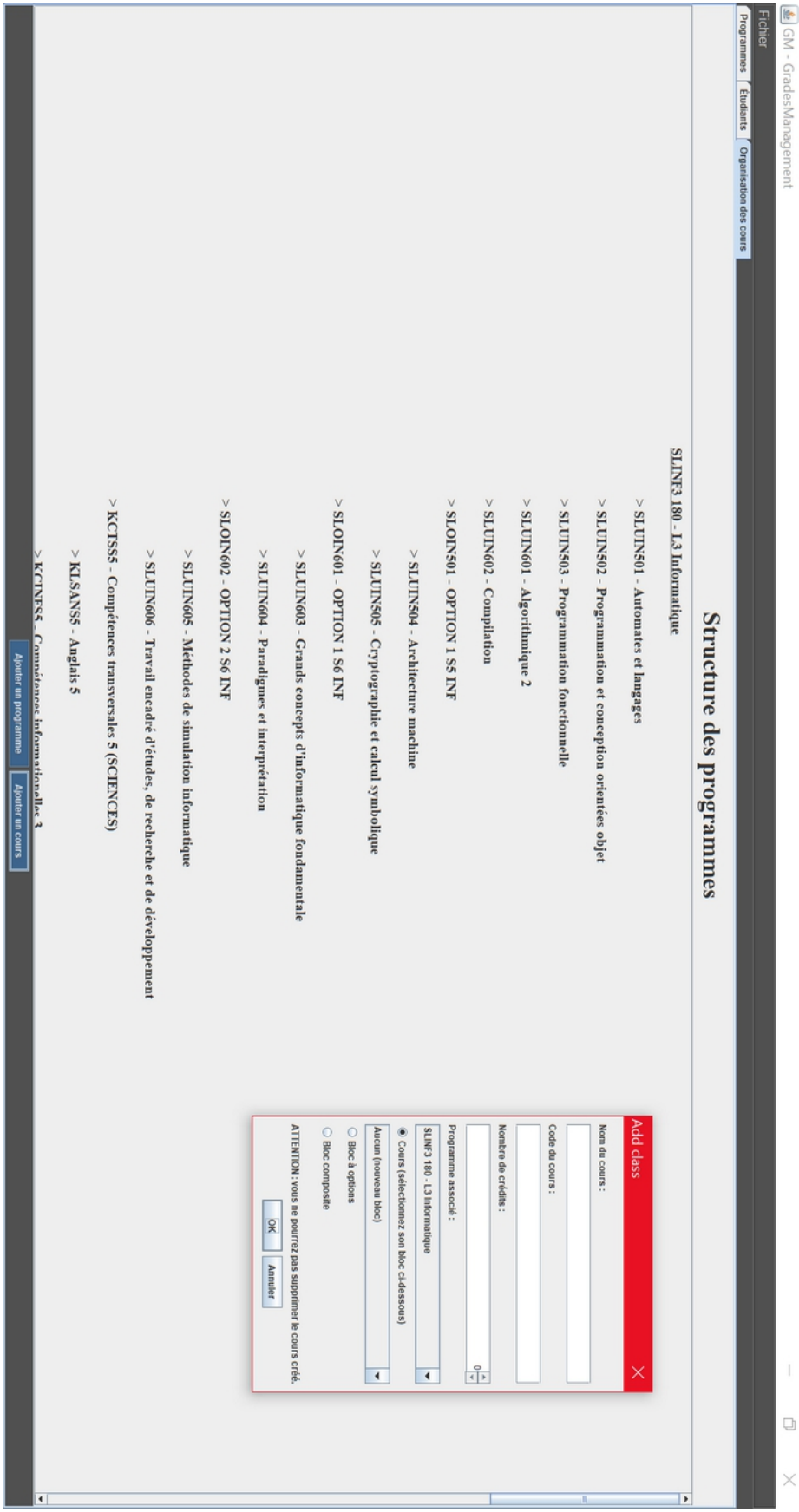


Image 3 : Aperçu de l'onglet « Organisation des cours » (avec ajout d'un cours)