ScholarProjects
Rapport Developpeur

Programmation Objet avancée

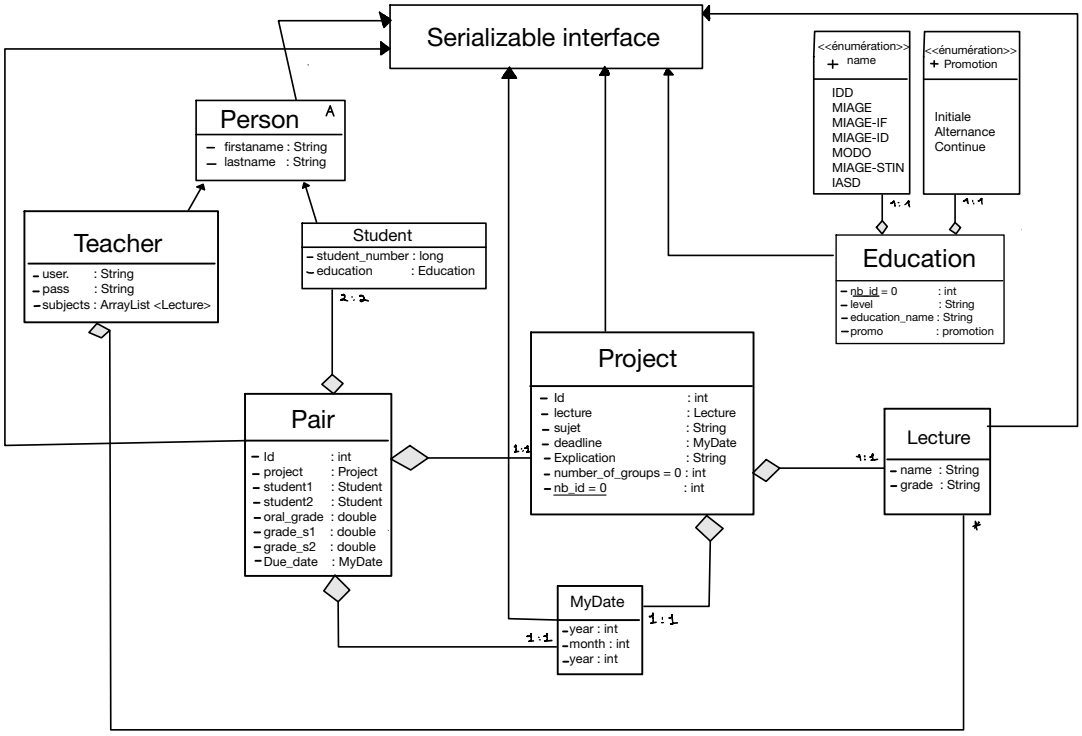
Léa Cohen-Solal
Melvin Guirchoun
Dirigé par ZELLAMA KHADDOUJA

1 Introduction

Notre projet propose une solution pour la gestion des projets étudiants, exclusivement conçue pour les professeurs. Cette plateforme leur permet d'accéder à tous leurs cours de l'année en cours, de gérer les listes d'étudiants, d'ajouter et de superviser des projets, et de former des binômes. Les professeurs peuvent également noter les étudiants, analyser les performances du projet (moyenne, meilleure note, etc.), et supprimer les projets si nécessaire. Notre objectif est de rendre la gestion des projets plus efficace et intuitive pour les enseignants, en centralisant toutes les informations et fonctionnalités nécessaires dans une interface unique.

2 Architecture

2.1 Diagramme UML



2.1.1 Types de classes

- **Fabrique d'Instances**
  Les classes **Person, Teacher, Student, Project, Pair, MyDate, Lecture, Education** permettent de créer des instances de ces classes, et de manipuler les attributs. Elles ne 'font' aucune action et sont géré par les classes **ActionListener** (expliquées ci-dessous)

- **ActionListener** :
  La classe **ActionListener** est une interface conçue pour gérer les actions de l'utilisateur, généralement les clics sur les boutons dans l'interface graphique. Elle possède une méthode, actionPerformed(ActionEvent e) qui va entraîner une action spécifique pour chaque bouton cliqué.C'est cette classe qui nous permet de rendre dynamique et interractive l'interface.
  La classe **ActionStudents** est conçue pour exécuter une action spécifique : ajouter un étudiant à un cours. Initialement, elle présente la liste des étudiants non inscrits au cours ('print_students_not_in_lecture'), permettant ainsi à l'enseignant de choisir individuellement les étudiants à inscrire. Une fois que

l'enseignant confirme son choix, la liste des étudiants inscrits dans le cours est mise à jour dans General_List('add_student_in_lecture(Student student, Lecture lecture)')
Nous avons aussi crée les classes **ActionLectures** (resp **ActionProjects**) qui gèrent les actions liés au cours (resp au projets).
Les attributs de la classe **ActionLectures** et **ActionProjects** sont définis comme publics car ils sont uniquement utilisés à l'intérieur de la classe, ce qui minimise les risques associés à l'encapsulation. Cette décision contribue à simplifier le code, en rendant l'accès aux attributs plus direct et en éliminant le besoin de méthodes getters et setters.

Dans **ActionLectures** nous avons définit 4 attributs publics

```
    public Lecture lecture ;
    public JFrame frame ;
    public String action ;
    public int case_action ;
```

Listing 1: Attributs de la classe ActionLectures

Les attributs frame, action et case_action sont définis lors de l'instanciation de la classe ActionLectures. Par contre, l'attribut lecture est déterminé ultérieurement à travers l'appel de la méthode find_Lecture(String name, String grade). Lorsqu'un utilisateur sélectionne un cours spécifique, ce cours est stocké dans la variable 'action'. En activant la méthode actionPerformed d'**ActionLectures**, l'interface considère alors ce cours comme le cours actuel, permettant à l'utilisateur d'y accéder.
Il y a deux scénarios principaux pour gérer les actions liées à un cours spécifique, et c'est le rôle du case_action de déterminer lequel des deux l'utilisateur souhaite exécuter :

Case 1 : Cette option configure et affiche le menu associé à un cours spécifique. Elle permet par exemple à l'utilisateur d'accéder à des fonctionnalités telles que suivre l'avancement d'un projet ou créer un nouveau projet. Cette branche s'occupe principalement de la gestion et de l'interaction avec les projets liés au cours.
Case 2 : Il est lancé par le clic "Generate the student list". Ce cas est utilisé pour accéder à la liste des étudiants inscrits dans le cours en question. Cette liste est indépendante de tout projet spécifique et se concentre uniquement sur les participants au cours.

```
        Find_lecture(String name, String grade)

```

Lorsqu'un utilisateur sélectionne un cours dans l'interface de l'application, ce cours est enregistré sous la forme de son nom et de son niveau. La méthode est ensuite appelée avec ces deux paramètres (nom et niveau). Elle parcourt la liste General_List.list_lecture pour trouver un cours correspondant à ces critères et renvoie le cours correspondant.

```
    print_student()
```

Quand l'utilisateur souhaite voir la liste des étudiants inscrits à la matière spécifiée dans 'action', la fonction print_student() est appelée. Cette fonction récupère les données nécessaires à partir de la structure list_lecture_student_list (de la classe General_List) . Pour l'affichage de ces étudiants dans l'interface graphique, elle utilise la classe Table_students, dont les détails seront expliqués ultérieurement dans le rapport. Cette approche permet de présenter de manière claire et structurée les informations sur les étudiants suivant le cours.

Accès à **ActionStudents**

Lors de la sélection de "Add a new student in this lecture", la méthode 'actionPerformed' de la classe ActionStudents est activée. Si par exemple, un enseignany décide d'ajouter un étudiant à son cours, il cliquera sur 'Add a new student in this lecture'. Cette action est gérée par la ligne de code

```
add_new_student.addActionListener(new ActionStudents(frame, lecture));
```

Listing 2: Code 1

La méthode 'actionPerformed' est conçue pour générer une liste d'étudiants absents du cours 'print_students_not_in_lectu offrant ainsi à l'enseignant la possibilité de choisir et d'ajouter un étudiant spécifique à la liste. Lorsque l'enseignant sélectionne un étudiant, la méthode add_student_in_lecture(Student student, Lecture lecture) intervient. Elle commence par récupérer toutes les formations contenant le cours concerné. Pour chaque formation identifiée, elle procède à l'ajout de l'étudiant choisi. Cette addition s'effectue en faisant appel à la méthode statique get_student_from_education de la classe **General_list**, qui s'occupe de l'intégration de l'étudiant dans les listes appropriées.

Accès à **ActionProjects**

Lors de la sélection de 'Add a new Project' ou 'My Projects', la méthode 'actionPerformed' de la classe ActionProjects est activée. Si, par exemple, un enseignant décide d'ajouter un projet à son cours, il cliquera sur 'Add a new Project'. Cette action est gérée par la ligne de code

```
add_project.addActionListener(new ActionProjects("Add a new project", frame, lecture));
```

Listing 3: Code 1

,qui initie une nouvelle instance de ActionProjects avec les paramètres définis. Cette instance va ensuite gérer l'action choisie, dans ce cas, l'ajout d'un nouveau projet.

Dans **ActionProjects** Dans la classe ActionProjects, trois attributs sont définis de manière intuitive, grâce à leurs noms explicites

```
public JFrame frame ;
public String action ;
public Lecture lecture ;
```

Listing 4: Attributs de la classe ActionProjects

Ces attributs reflètent clairement leur fonction et leur utilisation. Prenons l'exemple mentionné précédemment lors de l'exécution du Code 1, "Add a new project" est attribué à la variable action.

```
public void actionPerformed(ActionEvent e){
        switch (action) {
            case "Add a new project":
            Menu.refresh(frame,0);
            create_project();
            break ;

            case "My projects":
            Menu.refresh(frame,0);
            check_all_projects();
            break;

            case "create2 my project":
            create2_project();
            break;
        }
    }
```

Lorsque la méthode actionPerformed est appelée (comme illustré ci-dessus), elle entre dans le premier cas et lance donc 'create_project()'.

La méthode 'create_project()' permet à l'utilisateur de saisir les informations pour un nouveau projet : le nom du projet, sa deadline, et une description. Ces détails sont recueillis via des champs de texte (JTextField) intégrés à la frame. Après que l'utilisateur confirme la création du projet (via le boutton), la fonction 'create2_project()' est appelée, à travers une nouvelle instance de la classe ActionProjects.

La méthode 'create2_project()' s'occupe de récupérer les données saisies dans les trois JTextField. Elle les convertit en chaînes de caractères (String) qui seront ensuite assignées au nouveau projet créé.
Il a fallut faire attention lors de la récupération des composants appropriés de la frame. Par exemple, le JTextField pour entrer le sujet du projet est le 4e composant dans la frame (les éléments du menu n'étant pas comptés comme des composants et les trois premiers éléments étant des labels indiquant où saisir les informations requises).

Nous avons opté pour la création de deux méthodes distinctes au lieu d'une seule pour la création d'un projet ('create_project' et 'create2_project'), principalement pour des raisons esthétiques du code et afin d'éviter des méthodes trop longues.

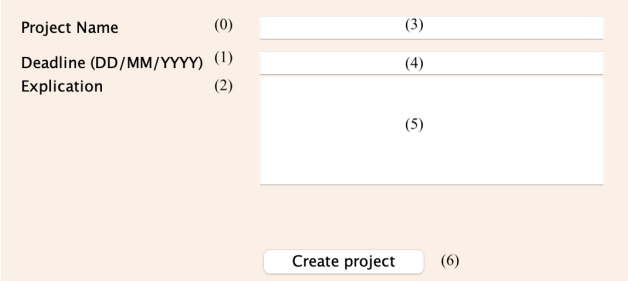


Figure 1: Numérotation des composants de la frame

```
String subject = ((JTextField)(components[3])).getText();
String deadline = ((JTextField)(components[4])).getText();
String explication = ((JTextField)(components[5])).getText();
```

Quand l'utilisateur, veut consulter tous ses projets, il clique sur 'My Projects', ce qui déclenche l'exécution de la méthode 'check_all_projects'. Cette méthode récupère les projets lié au cours 'lecture' et vérifie si l'enseignant a des projets en cours.

```
ArrayList<Project> projects = lecture.get_lecture_project();
[...]
if (projects.size() == 0 ){message d' information}
```

Si ce n'est pas le cas, un message lui indique qu'il n'a aucun projet en cours. Ce message s'affiche sur l'interface grâce à la commande Java suivante :

```
JOptionPane.showConfirmDialog(frame, "You don't have any project", null, JOptionPane.
    DEFAULT_OPTION);
```

L'utilisateur peut simplement cliquer sur 'OK' pour fermer ce message d'information. Si l'enseignant possède des projets, la méthode 'print_projects()' est lancée et les projets s'affichent via un tableau, semblable à celui utilisé pour afficher la liste des étudiants d'un cours. Ce tableau est créé à l'aide de la classe 'Table_projects'.

La méthode 'print_project' s'occupe ensuite d'afficher une série de boutons qui permettent diverses manipulations sur chaque projet. Ces manipulations incluent l'ajout de binômes, l'évaluation de ces derniers, la suppression du projet, ou encore l'accès à toutes les caractéristiques du projet. Ainsi, l'interface fournit à l'enseignant une vue d'ensemble de ses projets en cours ainsi que des outils pour les gérer efficacement.

Lorsque l'utilisateur clique sur un des boutons la méthode 'manipulation_project' et lancé est traite de tous les cas de figure.

```

public void manipulation_project(Project project,String action_project) {
       switch (action_project) {

           case "Check this project" :
           project_features(project) ;
           break ;
```

```
8
9               case "Grade students" :
10              grade_students(project);
11              break ;
12
13              case "Add Pairs" :
14              add_pairs(project);
15              break ;
16
17              case "Remove project" :
18              int result = JOptionPane.showConfirmDialog(frame, "Are you sure you want to
     delete this project?", "Confirmation", JOptionPane.OK_CANCEL_OPTION);
19              if (result == JOptionPane.OK_OPTION) { remove_project(project); }
20              else {if (result == JOptionPane.CANCEL_OPTION) {print_projects(frame,
    lecture);}}
21              break ;
22          }
23  }
```

La fonction 'project_features' fonctionne de la meme maniere que 'print_students' et 'print_projects' en affichant les caracteristiques du projet sous forme d'un tableau.
NB : Une note à -1 signifie qu'elle n'a pas été encore affecté.

Ajouter et Supprimer un Binôme

Lorsque l'utilisateur choisit l'option 'Add Pairs' pour un projet, voici ce qui se produit en résumé :

Les binômes existants du projet sont récupérés. La liste des étudiants de la matière est également obtenue. La fonction Project.isAlone() génère la liste des étudiants sans binôme pour éviter des doublons dans les attributions. Les classes Table_student et Table_pair sont utilisées pour afficher deux tableaux côte à côte : l'un montrant les étudiants seuls, et l'autre les binômes actuels. Le processus d'ajout est le suivant :

Sélectionner un étudiant et cliquer sur 'add'. Sélectionner un second étudiant et cliquer sur 'add'. Si les deux étudiants sont différents, le binôme est ajouté au tableau des binômes, et les deux étudiants sont retirés de la liste des étudiants seuls grâce à la méthode IsAlone() et le processus continue.

Lorsque l'utilisateur sélectionne un étudiant dans l'interface, cet étudiant est enregistré dans l'attribut 'selectedstudent1' de la classe Table_pair. Si l'utilisateur sélectionne ensuite un autre étudiant, la méthode correspondante vérifie, à l'aide de la méthode equals(), que les deux étudiants ne sont pas identiques avant de les ajouter comme binôme. Une fois le binôme formé, ces attributs sont réinitialisés à null pour permettre de nouvelles sélections.

```
1 private static Student selectedstudent1 = null;
2 private static Student selectedstudent2 = null;
```

Listing 5: attributs de la classe Table_pair

Le principe est le même pour la suppression d'un binôme.

Noter les binômes

Quand l'utilisateur sélectionne l'option 'Grade students', la méthode 'grade_students(Project project')' est exécutée. Cette méthode commence par vérifier si les binômes ont déjà reçu une note, en utilisant la condition suivante :

```
1 if (pair.getGrade() != -1)
```

Si un binôme a déjà une note (c'est-à-dire, la note est différente de -1), le programme interroge l'enseignant pour savoir s'il souhaite modifier cette note existante.
Si l'utilisateur décide de changer une note existante ou d'attribuer une note à un binôme qui n'en a pas encore, la méthode 'write_grade' est appelée. Cette méthode suit une procédure similaire à celle de 'create_project'. L'utilisateur saisit dans des champs **JTextField** les notes attribuées au binôme, ainsi qu'aux étudiants individuellement, et la date de rendu du projet. Cette date de rendu est utilisée pour calculer la note finale.

Les méthodes v'erif_grade(String g1, String g2, String g3)' et 'verif_date(String date)' sont conçues pour vérifier que les entrées pour les notes et les dates sont dans le bon format syntaxique.

La méthode 'verif_grade()' est déclarée comme **static** car elle est utilisée pour valider les notes avant de les affecter aux binômes. Lorsque cette méthode est exécutée, les attributs de note, qui sont :

```
private double oral_grade;
private double grade_s1;
private double grade_s2;
```

sont encore non assignés (à moins qu'une note ne soit en train d'être modifiée). Cette méthode ne nécessitant pas l'accès aux attributs d'instance de la classe, elle peut donc être **static**. Bien qu'elle soit indépendante des attributs de la classe, 'verif_grade()' a été placée dans la classe Pair car elle se rapporte directement à la validation des champs spécifiques de cette classe.

Une fois que l'utilisateur confirme les notes saisies, la note finale du binôme est calculée avec la méthode 'calcul_final_grade'. Les notes finales des étudiants sont ensuite déterminées en faisant la moyenne entre la note du binôme et leur note individuelle.

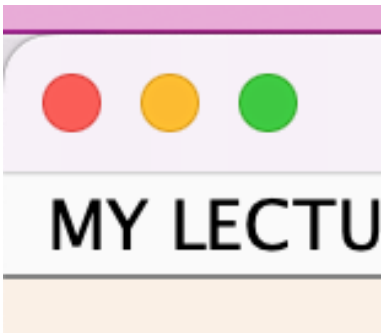
NB : Si la date de rendu est antérieure à la deadline, le système suppose qu'il s'agit d'une erreur de la part de l'enseignant la note du binôme ne change pas.

```
Math.min(final_grade, grade)
```

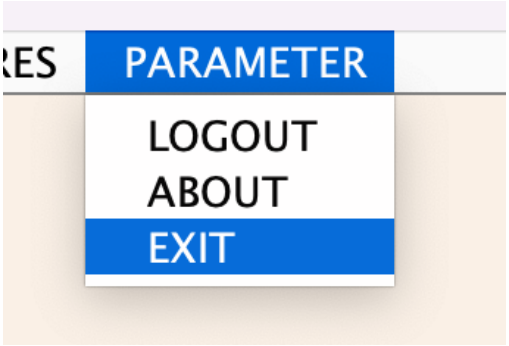
Parametre

Cette classe permet d'éffectuer 3 actions :Se déconnecter, fournir des informations supplémentaires sur l'application et quitter l'application. Elle déclenche des actions simples, facilement compréhensibles dans le code.

Point très important ! Pour quitter l'application, l'utilisateur doit obligatoirement utiliser le bouton "EXIT" situé dans la section "PARAMETRE". Ce processus est essentiel car c'est ce bouton qui déclenche la sauvegarde des données, garantissant ainsi leur préservation pour la prochaine utilisation de l'application.



(a) A NE PAS FAIRE



(b) A FAIRE

- **AbstractTableModel**
  Les classes **Table_student, Table_lecture, Table_project, Table_pair, Table_features** héritent toutes de la classe **AbstractTableModel**. Elle permettent d'afficher des tableaux d'étudiants, de projets.... et de selectionner des éléments de ceux-ci. **Pour l'implémentation de ces classes nous nous sommes aidés du site** `https://www.demo2s.com/java/java-abstracttablemodel-tutorial-with-examples.html`

3 Gestion des difficultés

3.1 Stockage des données

Nous avons opté pour l'utilisation de structures de données telles que les ArrayList et les Hashmap pour le stockage de nos données. Les ArrayList servent à conserver les données, comme les listes d'étudiants, de formations et de cours. Cette approche nous permet une gestion efficace de ces collections, facilitant l'accès et la manipulation des données. En ce qui concerne les données plus complexes, comme les informations relatives aux projets, nous avons choisi d'utiliser des Hashmap. Ces derniers associent chaque projet à une liste correspondante de binômes, offrant ainsi une méthode structurée et intuitive pour relier les projets à leurs équipes respectives.

L'utilisation de Hashmap et ArrayList en Java facilite grandement la gestion des données pendant l'exécution de l'application. Cependant, ces structures ne sont pas conçues pour maintenir les données après la fermeture du programme. Ainsi, toutes les modifications ou ajouts effectués sont perdus une fois l'application arrêtée, car elles sont réinitialisées à chaque nouvelle exécution.

Pour remédier à cette problématique, nous avons opté pour l'intégration de fichiers de sauvegarde qui permettent de conserver les données des ArrayList et Hashmap d'une session à l'autre. En procédant ainsi, chaque fois que l'application s'exécute, elle peut lire ces fichiers pour restaurer les données précédemment enregistrées.

3.1.1 Fichiers.bin et sauvegarde des données

Pour assurer la sauvegarde et la persistance des données dans notre application, nous avons mis en place un système de fichiers .bin pour chaque type d'information clé. Ces fichiers conservent de manière efficace les données suivantes :

Liste des Étudiants : Toutes les informations concernant les étudiants inscrits sont enregistrées dans un fichier dédié (student_list.bin).
Liste des Formations : Un fichier séparé stocke les détails relatifs aux différentes formations disponibles (list_education.bin).
Liste des Cours: Les informations concernant les différents cours offerts sont également sauvegardées dans un fichier spécifique (list_lecture.bin).
Correspondance Cours-Formations: Un fichier enregistre les associations entre les cours et les formations correspondantes (lecture_education.bin).
Correspondance Formation-Étudiants : Ce fichier conserve les liens entre les formations et les étudiants qui y sont inscrits. (list_education_student.bin).
Correspondance Projet-Binômes: Un fichier distinct est utilisé pour garder la trace des projets et des binômes d'étudiants qui y travaillent. (project_pair.bin).
Information sur le professeur utilisateur : Les données relatives au professeur qui utilise la plateforme sont également stockées dans un fichier .bin. (teacher.bin).

Pour recuperer les données en lançant l'execution, le programme lance la fonction *read* de la classe *ReadObject*.

Le processus débute par la vérification de l'existence du fichier nécessaire, tel que celui pour les listes d'étudiants. Si le fichier n'existe pas ou est vide, il est créé ou initialisé. Une fois le fichier disponible et contenant des données, le programme utilise *FileInputStream* pour l'ouvrir, puis *ObjectInputStream* pour lire les données. Ces données, d'abord en format binaire du fichier .bin, sont converties en objets Java utilisables. Elles sont ensuite intégrées dans les structures appropriées de la classe General_List, comme la mise à jour de la liste des étudiants. Durant ce processus, le programme gère les éventuelles erreurs de lecture ou d'accès aux fichiers. C'est la Sérialisation.

Pour l'exemple de la **student_list** voici comment la fonction read() récupère les données :

- Vérification de l'Existence du Fichier : Le programme commence par vérifier si le fichier nommé stu-

dent_list.bin existe. Si ce fichier n'existe pas ou est vide, il est automatiquement créé ou initialisé pour être prêt à l'usage.

- Lecture du Fichier : Si le fichier student_list.bin existe déjà et contient des données, le programme utilise un *FileInputStream* pour ouvrir ce fichier. Ensuite, il utilise un *ObjectInputStream* pour lire les données du fichier.
- Conversion des Données : Les données lues, qui sont en format binaire, sont converties en un format utilisable par Java. Elles sont transformées en une liste d'étudiants, ArrayList¡Student¿, qui est un format de liste compréhensible et manipulable dans Java.
- Mise à Jour de la Liste des Étudiants : Après la conversion, cette liste d'étudiants est assignée à General_List.student_list. Cela signifie que les données récupérées du fichier remplacent ou actualisent la liste des étudiants dans la classe General_List.

En stoppant l'exécution du programme , la fonction *write*() de la classe *WriteObject* sauvegarde les données en utilisant FileOutputStream pour ouvrir ou créer des fichiers .bin, comme "student_list.bin" pour les étudiants. Elle utilise ensuite ObjectOutputStream pour écrire ces données dans les fichiers. Après l'écriture, le flux est fermé pour terminer la sauvegarde. Le programme gère aussi les erreurs éventuelles liées à l'écriture des fichiers, assurant ainsi un processus de sauvegarde fiable. C'est la Désérialisation.

3.1.2 Problèmes rencontrés

Lorsque les données sont récupérées par l'intermédiaire des fichiers (Désérialisation) nous avons constatés que les instances d'une exécution précédente n'étaient pas les mêmes que les instances dans l'exécution actuelle. Le test == entre une instance chargée et une instance existante échouera. Nous avons donc ajouté pour chaque classe une méthode *equals*() qui vérifie si 2 objets partagent les mêmes données. Par exemple un étudiant a un numéro étudiant unique. Nous pouvons donc comparer facilement 2 étudiants.
Etudiant1 == Etudiant2 si leur numéro étudiant est similaire.

Ajout de nouvelles méthodes
En plus des méthodes 'equals()', nous avons dû développer de nouvelles fonctions, comme

```
public static boolean contains_the_key(Lecture lecture)
```

Cette fonction remplit une fonction similaire à la commande contains(lecture) pour le dictionnaire **lecture_education**. Comme les instances varient d'une exécution à l'autre, il est nécessaire de parcourir le dictionnaire et d'utiliser equals pour vérifier si un cours lecture1 correspond au cours lecture passé en paramètre.

```
static ArrayList<Education> get_education_from_lecture(Lecture lecture)
static ArrayList<Student> get_student_from_education(Education ed)
static ArrayList<Pair> get_pairs_from_project(Project project)
static void remove_from_list(ArrayList<Student> list, Student student)
```

Ces fonctions utilisent toutes la méthode equals() et nécessitent de parcourir intégralement les listes ou dictionnaires pour localiser un élément spécifique.

3.2 Manipulation des Dates

La classe *MyDate* a été développée pour faciliter la gestion des dates. Cette fonctionnalité est particulièrement utile dans le contexte des projets , où le professeur définit une deadline. Lors de l'évaluation d'un binôme, le professeur saisit la date de rendu du travail et la note orale du binôme est calculée en tenant compte du retard éventuel.

3.2.1 Manipulation et vérification de format des dates

Étant donné que le système est conçu pour que l'utilisateur saisisse lui-même les dates au format (JJ/MM/AAAA), il est nécessaire de vérifier systématiquement que les données entrées correspondent effectivement

à une date valide. Cette vérification se fait par la fonction $verif\_date(Stringdate)$.

De même lorsque la prof saisit la date de rendu , il faut verifier qu'il s'agit d'une date valide $verif\_date(Stringdate)$ mais aussi calculer le nombre de jours de retard.$calcul_final_grade(doublegrade, MyDatedeadline, MyDatedueDate)$.

La difficulté n'est pas tant dans l'implémentation de cette fonction, mais plus dans la détection et l'analyse de tous les cas particuliers liés aux dates, des situations comme une deadline fixée en décembre avec un rendu en janvier, des années d'écart entre la date de deadline et la date de rendu, et des mois de deadline et de rendu différents sans qu'il y ait nécessairement un mois entier de retard. Concernant la note final :

- 1 point est retiré pour 3 jours de retard
- Pour plus d'un mois de retard, la note du binôme sera 0.

NB : Le code de la fonction permettant de calculer la note finale d'un élève a été trouvé sur internet. Les commandes ci-dessous permettent de calculer le nombre de jours de différences entre la deadline et la date de rendu d'un binôme.

```
DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd/MM/yyyy");
LocalDate d1 = LocalDate.parse(deadline.toString(), formatter);
LocalDate d2 = LocalDate.parse(duDate.toString(), formatter);
```

3.3 Interface graphique

3.3.1 Layout

Initialement, nous avions choisi d'utiliser un gestionnaire d'agencement, en configurant notre fenêtre avec

```
frame.setLayout(new BorderLayout());.
```

Toutefois, en ajoutant au fur et à mesure les labels (JLabel) et des champs de texte **JTextField**, nous nous sommes aperçu que les éléments n'étaient pas disposés comme nous le souhaitions. L'agencement automatique proposé par **BorderLayout** ne répondait pas à nos attentes. Nous avons donc du gérer le positionnement précis des composants de la fenêtre en utilisant

```
frame.setLayout(null);
```

Cela nous a contraints à définir manuellement quatre attributs pour chaque composant via la méthode **setBounds(x, y, width, height)**, nécessitant des tests répétés pour assurer un placement correct des éléments.

3.3.2 Difficultés actuelles

Nous faisons face à un problème persistant avec l'affichage de l'interface. Lorsque le programme est lancé, l'interface montre une page vide. Pour visualiser les éléments de l'interface, il est nécessaire d'ajuster légèrement la taille de la fenêtre. Sans cette modification, les composants de l'interface ne s'affichent pas systématiquement.