Projet Programmation et Conception Orienté Objet

2020
-
2021

Création d'un outil gestionnaire de notes universitaires

Quentin BEAUCHET

Yann FORMER

Gillian MASSE

Sommaire :

Propos introductifs

Ce document a pour but de décrire le déroulement et l'évolution de notre projet de Programmation et de Conception Orientée objet.

Pour lancer l'application, il suffit de lancer la classe « App » du programme.

Nous verrons dans une première partie une description générale. Dans le but de faciliter la compréhension du projet, nous dresserons dans une seconde partie un cahier des charges. Ce dernier nous permettra ensuite de dégager les fonctions nécessaires que nous allons dans une troisième partie conceptualiser. Comme toute proposition de réponse de ce type d'exercice, des améliorations peuvent être apportées. Nous verrons ceci dans la quatrième partie.

A la différence de travaux dirigés ou de travaux pratiques, un projet comme celui-ci s'étend bien plus dans le temps. Il sera ainsi intéressant dans une cinquième partie de se questionner sur les différents bienfaits de ce type travail et plus concrètement sur ce que ce dernier nous a apporté.

En espérant que ce rapport de projet soit aussi intéressant que ce que ce projet a pu l'être,

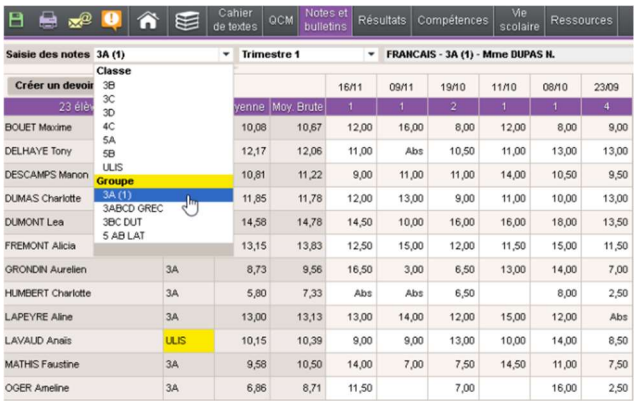
Bonne lecture,

I) Description du projet

a) Présentation du projet

Le projet qui nous a été donné à effectuer est la création d'un outil de gestion de note.

Dans le milieu universitaire, le concept est connu de tous ou presque. Impossible de ne pas citer comme exemple « Pronote », outil plébiscité dans les collèges et lycées, ou encore l'ENT et plus précisément par sa rubrique « notes et résultats ».



Un étudiant, au cours de sa vie universitaire, suit des cours (même en temps de Covid !). Il obtient normalement au sein de ceux-ci des notes qui vont lui permettre d'établir sa moyenne au sein de cet enseignement. En faisant la moyenne de tous les cours, il est possible de calculer le saint-Graal de l'étudiant : sa moyenne générale.

L'intérêt d'un tel outil est de simplifier toute cette gestion.

Même si le concept semble simple, il en reste néanmoins qu'il se cache une foule d'option à définir et par conséquent à programmer.

TDFTS1	Semestre 1 DUT Informatique	14.633/20	ADM
TDFTS1BS	Bonification Sport - Semestre 1	0/20	
TDFTS1PA	Pénalité pour absences - Semestre 1	0/20	
TDFTU11	UE11 : Bases de l'informatique	14.063/20	ACQ
TDFM111	Introduction aux systèmes informatiques	16.143/20	
TDFM112	Introduction à l'algorithmique et à la programmation	16.429/20	
TDFM113	Structures de données et algorithmes fondamentaux	16.4/20	
TDFM114	Introduction aux bases de données	9.071/20	
TDFM115	Conception de documents et d'interfaces numériques	12.38/20	
TDFM116	Méthodologie orientée métier - Découverte	14.25/20	
TDFTU12	UE12 : Bases de culture scientifique, sociale et humaine	15.379/20	ACQ
TDFM121	Mathématiques discrètes	15.66/20	
TDFM122	Algèbre linéaire	14.35/20	

b) Principales contraintes du projet

L'énoncé de cet exercice nous a imposé trois principales contraintes.

La première réside dans le mode de stockage des données. En effet, celui-ci doit se faire via un fichier XML. La seconde se situe dans la structure des données elle-même. Et enfin, comme dans toute application qui se respecte, la nécessité de faire une interface fonctionnelle.

D'après le sujet, il est nécessaire de distinguer plusieurs types de données :

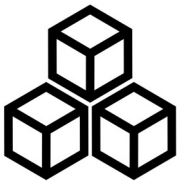
Le cours

Le cours représente l'unité d'enseignement basique de l'université. Identifiés par un code unique, tous les cours disposent d'un nom et se voient associer un certain nombre de crédit.

Le bloc

Le bloc se subdivise en 3 sous-catégories :

- Le **bloc simple** qui est tout simplement un cours.
- Le **bloc à options**, composé de plusieurs cours portant le même nombre de crédit. Ce nombre est de même le nombre de crédit du bloc.
- Le **bloc composite**, composé de plusieurs cours. A la différence du bloc à option, le nombre de crédit représente la somme des crédits des UE qui le composent.



Les blocs à options et composites sont considérés comme des **blocs multiples**.

Tous les blocs sont identifiés également par un code unique et dispose d'un nom et d'un nombre de crédits.

L'UE (unité d'enseignement)

Tous les blocs sont identifiés également par un code unique et dispose d'un nom et d'un nombre de crédits.

Le programme

Le programme est une liste de blocs à obtenir pour valider une étape quelconque (par exemple un semestre universitaire, sans importance pour la conception).

L'étudiant

L'étudiant se définit par un identifiant, son nom et son prénom. Il peut être inscrit à au plus un programme et dispose également de notes dans des UE, indépendamment du programme qu'il suit.

La note

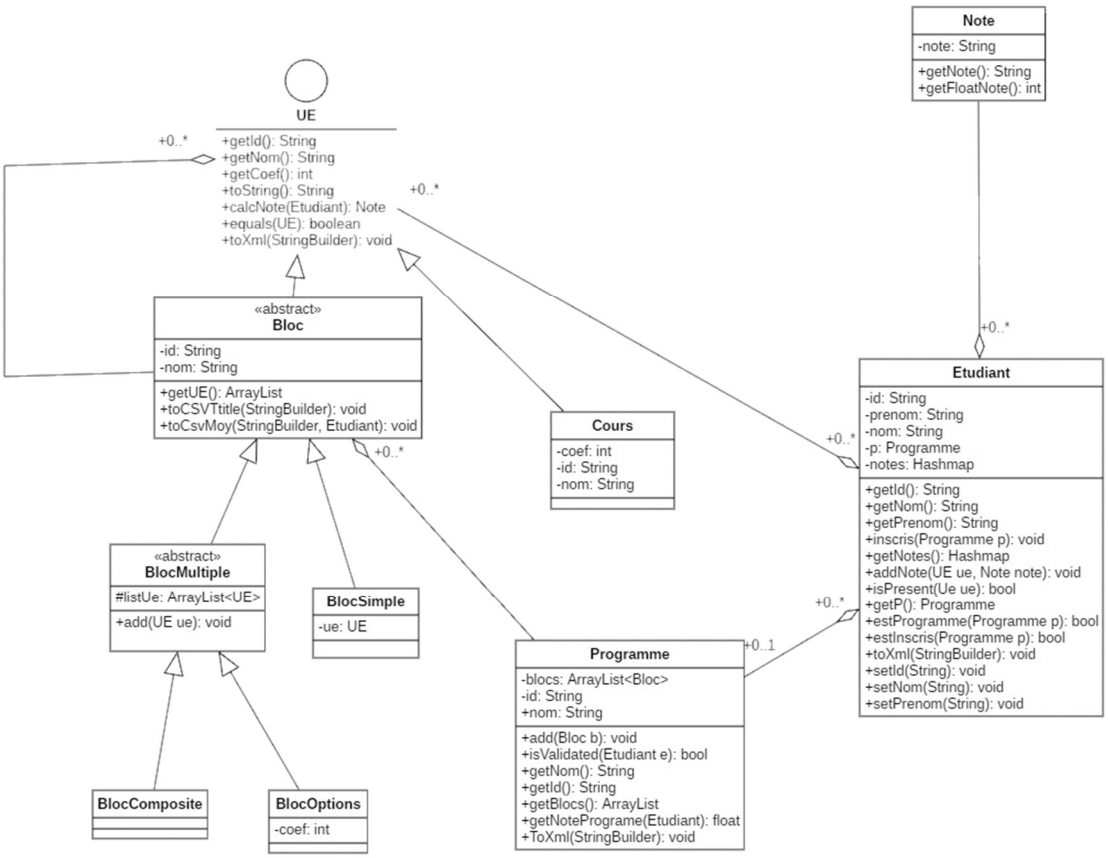
Une note est une valeur numérique comprise entre 0 et 20 ou bien la valeur « ABI » en cas d'absence de l'étudiant aux examens.

Et enfin, pour valider un programme, il faut obtenir la **moyenne pondérée** sur les blocs qui le composent.

- La note d'un *bloc simple* est la note du cours associé.
- La note d'un *bloc à options* est le maximum des notes obtenues sur les cours qui le composent.
- La note d'un *bloc composite* est la moyenne pondérée des UE qui le composent.

c) Modélisation des données du programme

Par souci de clarté et de compréhension du programme, nous avons effectué un Langage de Modélisation Unifié (UML) des données de ce dernier. Cet UML représente notre compréhension du



sujet. Il permet d'avoir une idée de la hiérarchie des données mais nous y reviendrons plus en détail un peu plus loin dans ce rendu.

II) Cahier des charges

Il est important dans n'importe quel projet de bien définir au préalable toutes les fonctions qu'il est nécessaire d'ajouter.

Dresser un cahier des charges complet permet de préciser les objectifs, identifier les contraintes, répartir les charges et missions… En bref, il permet d'avoir un socle de travail solide pour cadre le projet.

Ainsi, nous avons décidé, au début de notre projet, d'en créer un.

a) Fonctionnalités imposées

1) Accéder à l'interface graphique de gestion de notes

L'accession à l'interface graphique de gestion de notes va être conditionnée par la lecture d'un fichier d'étudiants et leurs notes ou sa création.

a. Lire un fichier d'étudiants déjà existant

Sélectionner et lire un fichier de note directement depuis l'application.

b. Créer et afficher un fichier de note vide

Proposer à l'utilisateur de créer un fichier de note vide.

2) Afficher les données récupérées

Afficher de manière claire et précise les données récupérées si l'utilisateur a décidé d'ouvrir un fichier de notes déjà existant.

3) Calculer de certaines valeurs pour chaque colonne

a. Calcul de la note minimale

Répertorier la note minimale dans la colonne correspondante.

b. *Calcul de la note maximale*

Répertorier la note maximale dans la colonne correspondante.

c. *Calcul de la moyenne*

Calculer la moyenne de la colonne.

d. *Calcul de l'écart-type*

Calculer l'écart type de la colonne correspondante.

4) Sélectionner/trier les données

a. *Sélectionner un programme et afficher les étudiants correspondants*

Permettre à l'utilisateur de sélectionner aisément un programme et d'afficher tous les étudiants qui le suivent. Il faudra de même afficher les notes de ces derniers.

b. *Sélectionner des programmes/blocs et afficher les étudiants inscrits*

Sensiblement la même fonctionnalité que précédemment à la différence que cette fois-ci il s'agit de sélectionner des programmes ou blocs et d'en afficher seulement les notes associées.

5) Hiérarchiser les différentes composantes

Apporter une hiérarchie, une arborescence aux données de notre application permettant une facilité de navigation pour l'utilisateur. Cette hiérarchie doit être accessible visuellement depuis l'application.

6) Modifier les données

Contrairement aux fonctionnalités précédentes, il s'agit maintenant de modifier « la base de données. »

a. *Ajouter/Modifier/Supprimer la note d'un cours*

Proposer de manière claire à l'utilisateur d'ajouter, de modifier ou de supprimer la note d'un cours.

b. *Ajouter/Supprimer un étudiant*

Proposer de manière claire à l'utilisateur d'ajouter ou de supprimer un étudiant. Ces opérations ont comme conséquences l'ajout ou la suppression d'une ligne du tableau.

c. *Ajouter un cours ou un programme*

Proposer de manière claire à l'utilisateur d'ajouter un cours à un programme et de choisir le type du bloc auquel il appartiendra. L'utilisateur a la possibilité de choisir un bloc existant ou d'en créer un. Cette opération a comme conséquence l'ajout d'une colonne dans le tableau.

d. *Ajouter un cours ou un programme*

Proposer de manière claire à l'utilisateur d'ajouter un programme.

7) Permettre de sauvegarder les données

Mettre à la disposition de l'utilisateur un outil de sauvegarde de ses données lors de l'utilisation du programme.

a. *Enregistrer*

Ecraser les données du fichier XML ouvert.

b. *Enregistrer-sous*

Créer un fichier XML à l'emplacement précédemment sélectionné.

c. *CSV*

Créer les CSV de tous les programmes du fichier XML.

b) Fonctionnalités ajoutées

1) Ajouter des fonctions graphiques

Bien conscient que la qualité graphique de l'interface est déterminante pour le plaisir des yeux mais aussi pour la lisibilité, nous avons décidé d'ajouter des caractéristiques graphiques à notre application.

a. *Afficher le nom complet de la colonne lors du passage de la souris*

Certaines colonnes peuvent avoir des noms trop longs pour être affichés entièrement. Les afficher entièrement lors du passage de la souris de l'utilisateur permet de pallier ce problème.

b. *Gérer et adapter la colorimétrie de l'application*

Nuancer les couleurs entre les lignes

Distinguer chaque ligne en faisant varier légèrement la couleur permet de faciliter la lecture de l'utilisateur.

Différencier les couleurs entre les types de données

Modifier le type de couleur en fonction du type de données permet de bien identifier les différentes composantes du tableau de données. Par exemple : distinction entre les notes et les informations personnelles de l'étudiant.

Adapter la couleur de la moyenne en fonction de la valeur

Un simple code couleur permet de mieux distinguer si l'étudiant a une moyenne supérieure à 10 ou non. Vert si c'est le cas, rouge sinon.

c. *Ne pas afficher les cours sans note lorsqu'un seul étudiant est résultat de la recherche*

Permettre de ne pas afficher les cours sans note lorsque le résultat de la recherche est un seul étudiant permet d'augmenter la lisibilité du résultat et de supprimer les informations inutiles.

d. *Adapter la taille de la fenêtre hiérarchie en fonction du nombre de données*

Avoir une grande fenêtre lorsque l'arborescence est peu complexe est inutile, et inversement, avoir une trop petite fenêtre peut tronquer l'affichage et être handicapant. Il est ainsi préférable d'adapter la taille de la fenêtre à la complexité de l'arborescence des données.

2) Permettre une plus grande facilité de navigation

a. *Permettre de filtrer le programme depuis la hiérarchie*

Il est imposé dans le sujet le fait de permettre à l'utilisateur de voir la hiérarchie des données mais également de les trier. Nous avons donc regroupé ces deux fonctionnalités. La sélection de cours, de programmes ou de blocs permet lors de la fermeture de la fenêtre d'actualiser l'affichage du tableau selon la sélection.

b. *Ajouter des raccourcis clavier*

L'ajout de raccourcis clavier permet de faciliter l'accès aux différentes fonctionnalités du programme.

3) Ajouter des facilités de sauvegardes

a. *Sauvegarder automatiquement au même emplacement*

La sauvegarde du fichier aura comme effet d'écraser le fichier et de le recréer au même endroit que là où il a été ouvert. On retrouve ici le fonctionnement d'un « ctrl-s » classique.

b. *Mémoriser les informations de sauvegarde*

Lorsque l'on ouvre un fichier où qu'on l'enregistre à un emplacement bien déterminé, le programme mémorise où l'on a effectué l'action. Par exemple, si l'utilisateur ouvre un fichier dans le dossier data, lors de la prochaine ouverture l'utilisateur sera déjà placé dans le dossier data. Ceci persiste même si l'on ferme le projet.

Maintenant que nous avons identifié les principales problématiques à résoudre, il est grand temps de passer à la conception de notre projet !

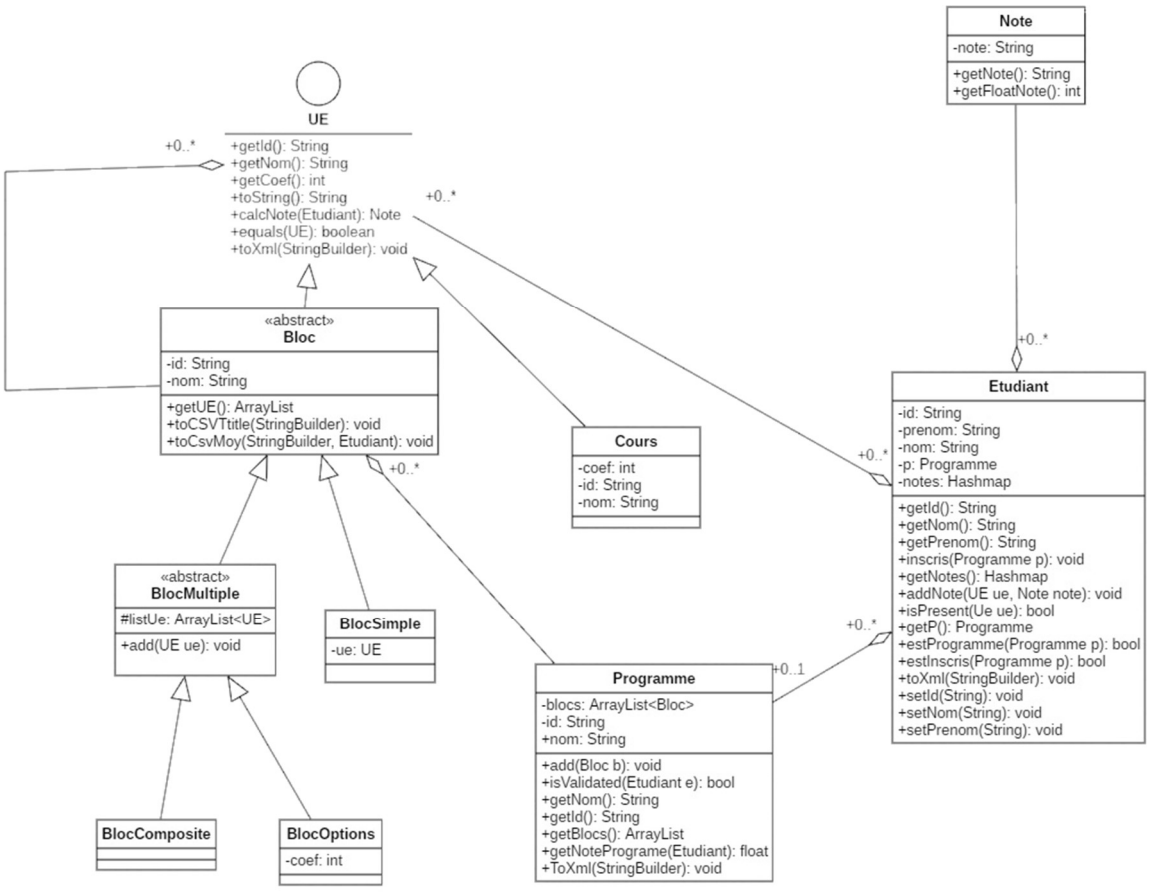
III) Conception

a) Les données du programme

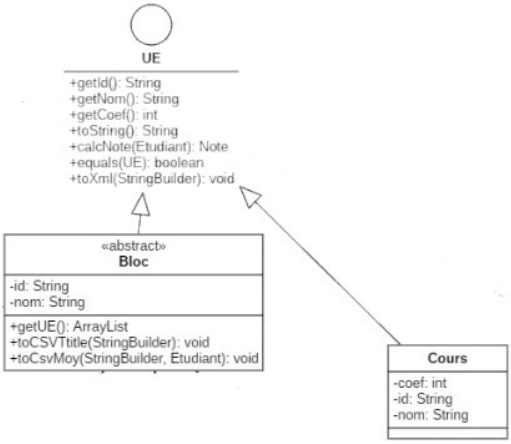
1) La conception des données.

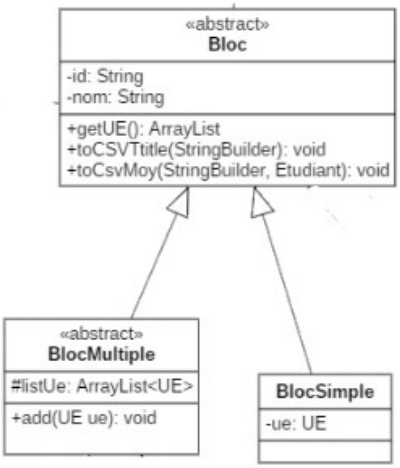
a. Relations d'héritages

Reprenons l'UML que nous avons vu tout à l'heure.



Pour conceptualiser ces données, nous avons d'abords créé une interface « UE » (unité d'enseignement) permettant de définir les fonctions de base d'une UE. On admet que les cours et les blocs sont des UE. Ainsi, la classe « Cours » et la classe abstraire « Bloc » implémentent l'interface « UE » comme vous pouvez le voir ci-contre.





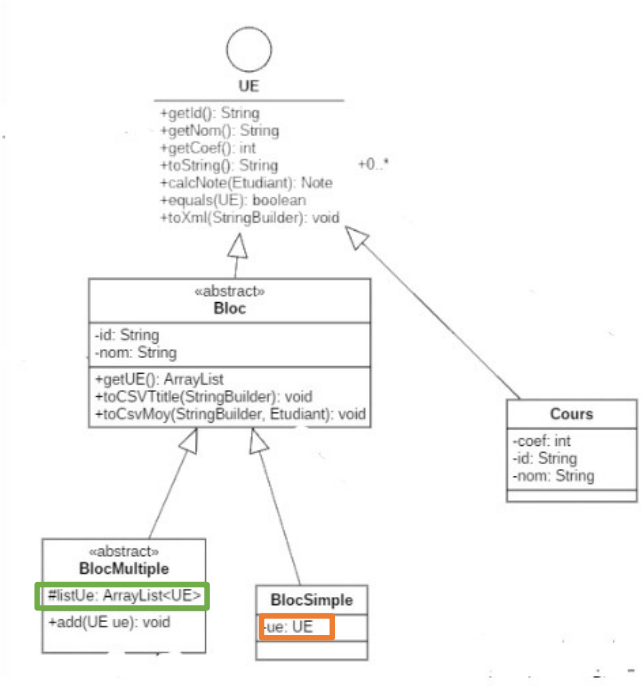
De la même manière, les blocs multiples et les blocs simples étant des blocs, il est logique que les classes correspondantes héritent de la classe abstraite bloc.

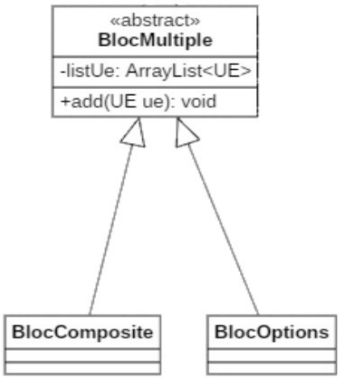
Enfin, blocs composites et bloc options sont des blocs multiples.

b. Interactions entre les classes

Maintenant que nous avons vu les liens d'héritages, attardons-nous sur les interactions entre classes.

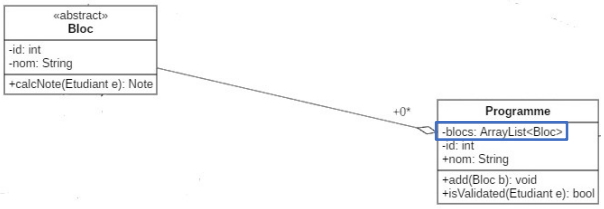
Un bloc multiple contient **une liste d'UE** là où un bloc simple ne contient **qu'une seule UE.** Rappelons qu'un cours est une UE, il est par conséquent tout à fait cohérent de dire qu'un bloc multiple peut contenir une liste de Cours.

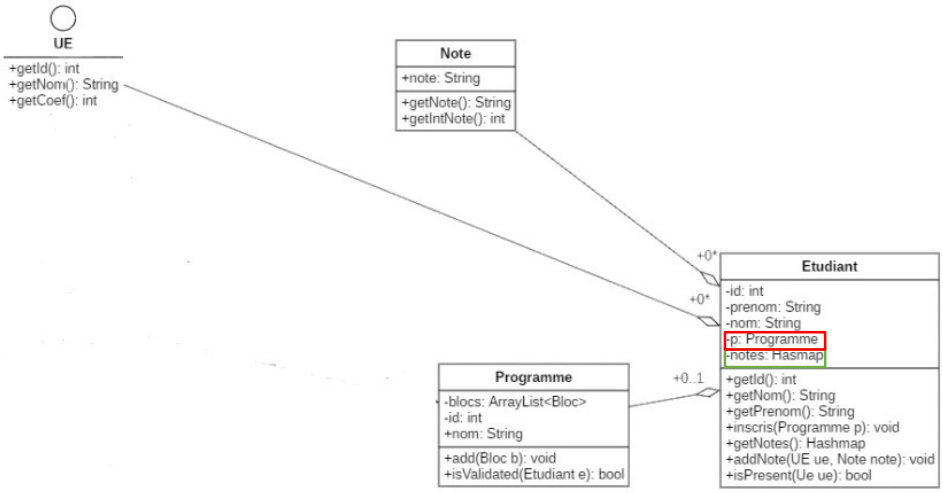




Par lien d'héritage, les blocs composites et à options contiennent de même une liste d'UE

Un programme contient **un ou plusieurs blocs**





Un étudiant est inscrit au maximum à **un programme** mais a **une note dans chaque UE**.

2) Interactions avec le fichier XML

Les données que nous manipulons dans ce projet sont contenues dans un fichier XML.

Nous allons par conséquent établir une suite de classe permettant de communiquer avec ce fichier.

Ainsi, une première classe **XmlReader** a pour but de convertir un fichier xml en données exploitables par le programme.

XmlMaker prend un **XmlReader** en paramètre et le converti en fichier xml qu'il va ensuite créer à l'emplacement prévu.

3) Stockage des données

Les données des étudiants sont stockées dans un XMLReader qui nous servira de « base de données » principale lors de l'affichage. Ainsi, pour l'affichage de 5 étudiants par exemple, le tableau prendra la forme suivante :

N° Étudiant	Nom	Prénom	Programme	Résultats	Automates...	Programm...
21606598	Idiri	Guillemette	L3 Informatique	14.32	15.475	19.024
21673265	Di Rosa	Henri	L3 Informatique	13.89	13.299	14.999
21784991	Brachet	Mehdi	L3 Informatique	13.63	12.163	12.518
21930353	Dormoy	Didier	L3 Informatique	11.39	13.565	8.652
21587965	Ouakkouche	Elodie	L3 Informatique	9.754	12.269	14.790

Les calculs sont affichés dans un second tableau :

Note max		14.32	20.0	19.024	17.554	20.0	17.597	19.675	17.373
Note min		4.656	2.861	2.447	2.218	4.343	2.905	2.251	2.793
Note moye...		10.07	10.45	10.37	10.44	10.53	10.35	10.77	10.08
Écart-type		2.582	3.379	3.34	3.366	3.17	3.279	3.475	3.341

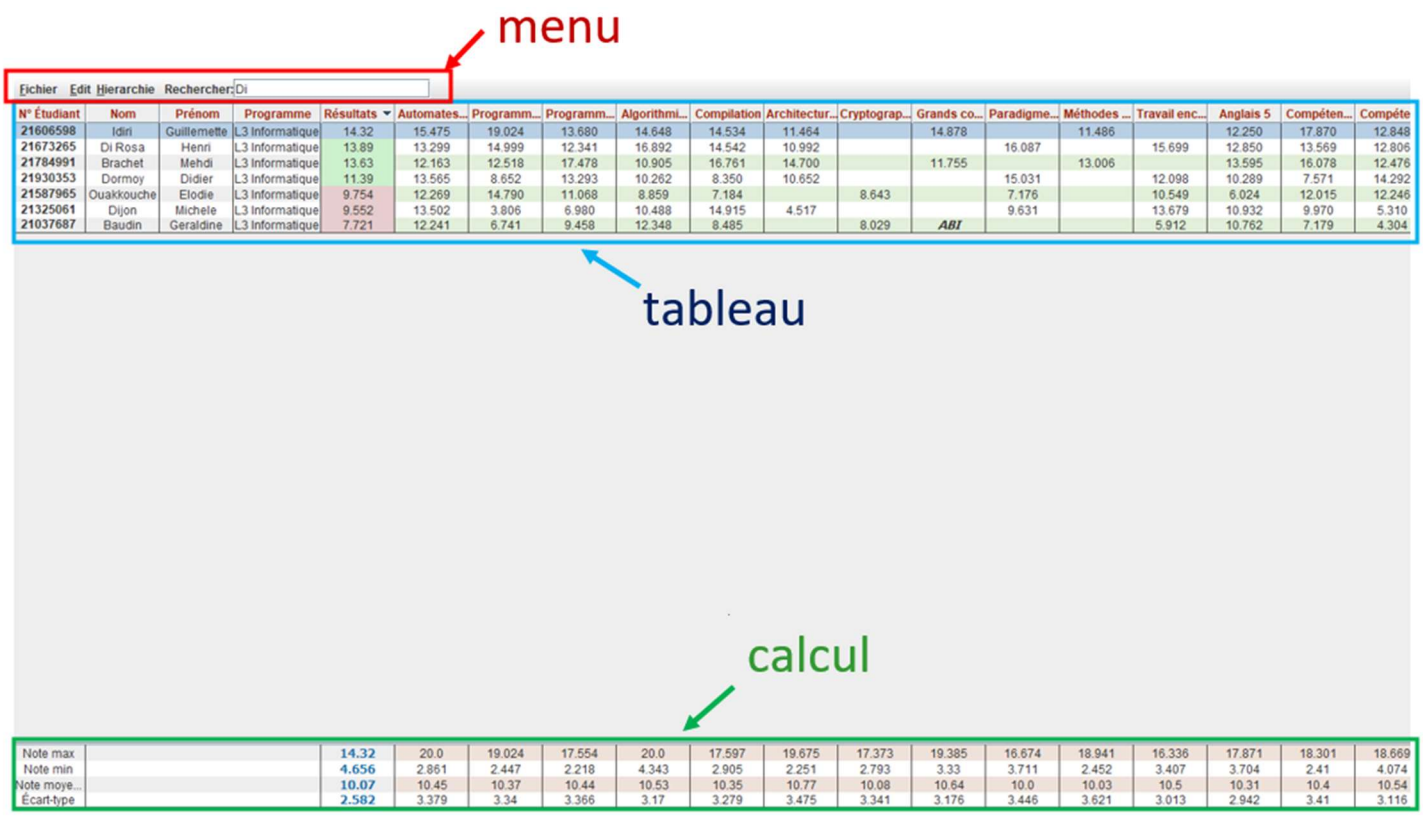
b) L'interface du programme

L'importance d'une belle interface est indéniable. Elle peut apporter une réelle crédibilité à une application. A contrario, une interface mal agencée va immédiatement lever un doute quant à sa qualité. Eviter cela donne une impression bien plus sérieuse et professionnelle.

De plus, celle-ci est également bien plus attrayante et agréable à utiliser. Un joli design et une organisation claire permettent de mieux capter l'attention de l'utilisateur et de lui donner envie de continuer à utiliser l'application. Une belle interface peut par ailleurs parfois suffire à distinguer une application d'une autre qui propose pourtant les mêmes services.

Enfin, il ne faut pas oublier que beaucoup d'utilisateurs ne sont pas à l'aise avec les outils informatisés. Or tout le monde doit pouvoir avoir accès aux mêmes fonctionnalités. Une interface simple et compréhensible est donc essentielle pour ces personnes. Elles peuvent ainsi en profiter sans rencontrer de particulières difficultés.

Nous avons donc essayé de créer une interface simple et efficace, comme vous pouvez le voir ci-dessous.



N° Étudiant	Nom	Prénom	Programme	Résultats	Automates...	Programm...	Programm...	Algorithmi...	Compilation	Architectur...	Cryptograp...	Grands co...	Paradigme...	Méthodes ...	Travail enc...	Anglais 5	Compéten...	Compéte
21606598	Idiri	Guillemette	L3 Informatique	14.32	15.475	19.024	13.680	14.648	14.534	11.464		14.878		11.486		12.250	17.870	12.848
21673265	Di Rosa	Henri	L3 Informatique	13.89	13.299	14.999	12.341	16.892	14.542	10.992			16.087		15.699	12.850	13.569	12.806
21784991	Brachet	Mehdi	L3 Informatique	13.63	12.163	12.518	17.478	10.905	16.761	14.700		11.755		13.006		13.595	16.078	12.476
21930353	Dormoy	Didier	L3 Informatique	11.39	13.565	8.652	13.293	10.262	8.350	10.652			15.031		12.098	10.289	7.571	14.292
21587965	Ouakkouche	Elodie	L3 Informatique	9.754	12.269	14.790	11.068	8.859	7.184		8.643		7.176		10.549	6.024	12.015	12.246
21325061	Dijon	Michele	L3 Informatique	9.552	13.502	3.806	6.980	10.488	14.915	4.517			9.631		13.679	10.932	9.970	5.310
21037687	Baudin	Geraldine	L3 Informatique	7.721	12.241	6.741	9.458	12.348	8.485		8.029	ABI			5.912	10.762	7.179	4.304

Note max		14.32	20.0	19.024	17.554	20.0	17.597	19.675	17.373	19.385	16.674	18.941	16.336	17.871	18.301	18.669
Note min		4.656	2.861	2.447	2.218	4.343	2.905	2.251	2.793	3.33	3.711	2.452	3.407	3.704	2.41	4.074
Note moye...		10.07	10.45	10.37	10.44	10.53	10.35	10.77	10.08	10.64	10.0	10.03	10.5	10.31	10.4	10.54
Écart-type		2.582	3.379	3.34	3.366	3.17	3.279	3.475	3.341	3.176	3.446	3.621	3.013	2.942	3.41	3.116

c) Conception du programme

1) Classes utilitaires

MyTools définit des méthodes statiques présentes dans le projet dont on se sert au travers de multiples classes. Il serait possible de citer pour exemple *getStats* dont on se sert souvent qui nous permet de récupérer la moyenne, le minimum, le maximum et l'écart type.

PathSaver est une classe qui, lors de l'instanciation, va récupérer un chemin vers un dossier (sous forme de string) et qui va l'écrire dans un .txt pour pouvoir orienter l'ouverture du FileChooser vers le bon endroit. Le fait de l'écrire dans un .txt nous permet de sauvegarder l'information même en cas de fermeture du programme. La fonction *readPath*() de *PathSaver* nous permet de récupérer le dernier chemin emprunté par l'utilisateur. En cas d'erreur, la classe renvoie « ».

PopUp est la classe qui génère presque tous les popup, celui pour confirmer avant de quitter, celui pour ajouter un **Cours**, celui pour ajouter un **Etudiant** et celui pour ajouter un **Programme**. Boutons est une classe qui étend *ActionListenner*, puis selon l'action appelle **AjoutCours** / **AjoutEtudiant** / **AjoutProgramme** etc.

2) Cœur du programme

La classe **HomeController** instancie **StartView**.

StartView est la classe qui permet soit de choisir un XML au démarrage soit d'en créer un vierge.

Home est la classe de la *JFrame* principale, depuis **Home** on instancie le tableau avec la classe → **TabCreation** et le Menu avec la classe **Menu**. C'est aussi par elle que passent les échanges d'informations entre le menu et le tableau.

Menu est la classe qui gère tout le menu donc c'est elle qui fait toute la partie graphique, les *listeners du menu* et les raccourcis clavier.

TabCreation est la classe qui génère les composants nécessaires aux *JTable* depuis l'**XMLReader**, elle crée 3 *array d'Object* : un pour *l'header* de la table des étudiants, un second pour le contenu de la table d'étudiants et un dernier pour le contenu de la table des calculs générés grâce à la classe **MyTools**.

Dans la classe Home après avoir instancié TabCreation, on instancie la classe **Tableau**. C'est elle qui génère toute la partie graphique des deux tableaux (celui des étudiants et celui des calculs min, max…). La classe Tableau permet aussi de fixer les **Sorter** du tableau des élèves et toutes les classes custom des JTable : **CustomCellEditor**, **CustomRenderer** et **CustomTableModel**. C'est grâce à eux que l'on peut manipuler les tableaux comme on le désire, comme pouvoir choisir la classe de *l'Object* dans chaque colonne, faire des vérifications au moment de l'ajout de texte dans les cellules, faire des affichages personnalisés, actualiser les valeurs de l'**XMLReader** et du tableau de calcul lors de la modification du tableau des étudiants.

Sorter est la classe qui permet de mettre un tri ASC/DESC sur le tableau pour chaque colonnes grâce à un comparateur custom **NoteComparator** pour les **Note**. Elle permet aussi de faire un tri sur les lignes grâce à un *string* donné depuis le *JTextField* du **Menu**, le filtre s'applique sur les colonnes des numéros étudiants, le nom et le prénom.

NoteComparator implémente *Comparator<Note>* et permet de comparer deux **Note**.

CustomCellEditor étend *DefaultCellEditor* et permet de filtrer les données lors de leur ajout dans les cellules du tableau. Par exemple, une **Note** doit être un float entre 0 et 20, ABI, vide, ou encore que le numéro étudiant doit être un int de 8 numéros.

CustomRenderer étend **DefaultTableCellRenderer** et implémente **TableCellRenderer.** Cette classe permet de faire un affichage personnalisé selon la classe de *l'Object* dans chaque cellule mais aussi de faire une couleur différente une ligne sur deux. Lorsqu'il n'y a qu'un seul étudiant, elle n'affiche que les cours dans lesquels il a une **Note**.

CustomTableModel étend **DefaultTableModel** et permet de forcer les classes des cellules selon les colonnes du tableau, d'actualiser les valeurs de l'**XMLReader** et le tableau des calculs après chaque modification de celui des étudiants.

3) Listeners

Boutons implémente *ActionListener* et est la classe qui redirige tous les boutons vers leurs actions respectives.

Window implémente *WindowListener* et elle empêche de quitter le programme sans passer par le **PopUp** de confirmation lorsque l'on clique sur la croix de la fenêtre.

RechercheField implémente *KeyListener* et permet de modifier le filtre du tri sur les lignes du tableau lorsque l'on écrit dans le *JTextField* du *JMenu* de Menu.

HeaderListener implémente *MouseListener* et affiche le titre de la colonne dans un *JDialog* quand on passe la souris dessus.

HierarchieBouton implémente *MouseListener* et instancie **HierarchieCreation** lorsque l'on clique sur le *JLabel* Hiérarchie dans le *JMenu* de Menu.

Dialog implémente *WindowListener* et instancie **HierarchieCreation** avec une *ArrayList<UE>* contenant les **UE** sélectionnés quand la *JDialog* de la hiérarchie est fermé.

Arbre implémente *TreeExpensionListener* permet de changer la taille de la *JDialog* de la hiérarchie dynamiquement selon le nombre d'éléments affichés.

4) Autres Classes

CustomListModel étend *AbstractListModel<Object>* et permet de faire des *JList* d'*Object*.

ProgrammeSwitch permet de changer la **TabCreation** à l'origine du tableau et le recréer selon un **Programme** ou une *ArrayList<Cours>*.

ProgrammeSelection crée une liste de **TabCreation** de tous les programmes pour facilement passer de l'affichage d'un **Programme** à un autre.

FileChooser configure le *JFileChooser* qui permet de sélectionner et d'enregistrer le fichier XML donné au **XMLReader**. Permet de choisir l'emplacement de l'enregistrement.

HierarchieCreation crée un *JTree* selon l'*ArrayList<Programme>* de l'**XMLReader** contenant tous les *DefaultMutableTreeNode* créés par **HierarchieSelection** pour chaque programme.
Elle crée aussi la *JDialog* et le filtre des **UE** à la fermeture de celle-ci passé à **HierarchieCreation** depuis **Dialog**.

HierarchieSelection crée une *DefaultMutableTreeNode* à partir d'un **Programme** et de ses **Blocs**.

AjoutCours ajoute un nouveau **Cours** dans un nouveau **Bloc** ou un **Bloc** déjà existant d'un **Programme** de l'**XMLReader** si les arguments passés en paramètre sont valides.

AjoutEtudiant ajoute un nouvel **Etudiant** dans au plus un **Programme** de l'**XMLReader** si les arguments passés en paramètre sont valides.

AjoutProgramme ajoute un nouveau **Programme** a l'**XMLReader** si les arguments passés en paramètre sont valides.

SupprimerEtudiant supprime l'**Etudiant** sélectionné de l'**XMLReader** et du tableau.

WriteCSV permet de créer des fichiers CSV de chaque programme présent à partir de **XmlReader**. Ils sont consultables dans le dossier data du projet.

IV) Pistes d'amélioration

a) Difficultés rencontrées

Nous avons rencontré dans la réalisation de ce projet quelques difficultés, plus précisément lors de l'ajout de fonctionnalités supplémentaires.

Non réussite du blocage du déplacement des colonnes du tableaux

Il est possible de déplacer à la souris des colonnes du tableaux. Nous n'avons pas réussi à empêcher cela.

Difficulté lors de l'utilisation d'un JMenu pour la hiérarchie

Lorsque l'on cliquait sur le JMenu, que ceci ouvrait le JDialog de la hiérarchie puis que l'on fermait ce dernier, le JMenu restait sélectionné. Nous avons donc remplacé le JMenu par un JLabel qui pallie ce problème. Or, ceci a eu pour conséquence la difficulté suivante.

Difficulté pour la mise en place de raccourcis pour la hiérarchie

Etant donné qu'il est impossible de mettre des raccourcis clavier sur des JLabel, nous avons placé un JMenu d'une taille égale à 0 derrière celui-ci (ce qui le rend invisible). Ce JMenu s'active lors de l'utilisation d'un certain raccourci ce qui a pour conséquence d'appeler le même ActionListener que le JLabel Hiérarchie.

b) Pistes d'améliorations

Comme dans absolument toute chose de ce monde, il est toujours possible de faire mieux. Alors nous avons exploré diverses pistes d'améliorations pour notre programme.

Base de données en ligne pour sauvegarder et récupérer les données

Un des majeurs défauts de ce programme est qu'il n'a d'effet qu'en local. En effet, il n'est pas possible de transférer les donner dans une base de données en ligne pour pouvoir les récupérer sur un autre ordinateur. Il est donc tout à fait possible d'imaginer pouvoir stocker les données sur un cloud ou une base de données SQL afin de les rendre accessibles de n'importe où.

Possibilité d'envoyer par mail les données

Transmettre les données n'est pas nécessairement chose aisée pour les personnes n'ayant pas de base en informatique. Il serait ainsi pertinent de prévoir une fonctionnalité dans le programme permettant de directement envoyer les données par mail à une adresse souhaitée.

Page d'aide pour les raccourcis

La quasi-totalité des applications de nos jours contiennent des manuels d'utilisation comme par exemple une page d'aide synthétisant tous les raccourcis de l'application. L'application étant pourvue de raccourcis, la création d'une page récapitulant ceux-ci pourrait simplifier la prise en main de l'interface.

Barre de défilement transparente

Après plusieurs jours d'utilisation de notre application, nous nous sommes rendu compte que la barre de défilement sur la droite cache certaines des données. Il pourrait être intéressant de la rendre transparente, pour que cet outil certes très pratique, ne gêne pas la visibilité.

V) Répartition du travail

La répartition du travail fût chose aisée.

Le projet étant découpé en 3 Jalons bien distincts, nous avons pris la décision de nous attribuer un Jalon principal chacun selon le découpage suivant :

- Yann FORMER : Jalon 1
- Quentin BEAUCHER : Jalon 2
- Gillian MASSE : Jalon 3

Malgré cette attribution nous avions bien conscience que les différentes parties de ce projet étaient interconnectées.

Ainsi, nous étions bien plus proches d'un « responsable » pour chaque Jalon que d'un réel découpage, chaque personne ayant directement ou indirectement participé à l'entièreté des jalons.

Soucieux de préserver un réel esprit d'équipe, nous avons réalisé le rendu du projet à trois.

VI) Apports du projet

Après 6 semaines de travail, nous avons terminé ce projet qui nous a finalement beaucoup tenu à cœur. En effet, au fur et à mesure du temps, notre volonté de créer le meilleur outil possible s'est peu à peu dévoilé. Arrivés au terme du projet, nous avons pu constater les nombreux bienfaits que la conception de celui-ci nous a enseignés. Il est donc possible d'en faire un bilan.

a) Les apports techniques

Probablement l'apport le plus important et le plus évident, l'amélioration de nos compétences techniques étant incontestables. De nos jours, la maîtrise du langage JAVA et de Swing est primordiale. S'améliorer dans ce domaine est une évolution très satisfaisante et nous rend le développement bien plus agréable et fluide. En effet, arriver à trouver simple ce que nous trouvions hier complexe est une réelle source de satisfaction. Avoir le sentiment de s'améliorer comme ceci nous pousse à vouloir en savoir toujours plus. L'utilisation de fichiers XML notamment, exercice que nous avions très peu réalisé, fut une découverte très enrichissante. Nous avons bien conscience qu'il nous reste encore une grande marge de progression mais la motivation qu'apporte la création d'un projet comme celui-ci nous conditionne à nous dépasser et à donner le meilleur de nous-même.

b) Les apports organisationnels

Malgré l'aspect technique indéniable de ce projet, un facteur de groupe peut venir en complexifier sa réalisation. La communication est primordiale afin que ce type d'exercice sot un succès. Evidemment, il y aura toujours des difficultés, des désaccords ou des mésententes, et c'est pour ces raisons qu'il faut apprendre à écouter les autres et à pouvoir remettre en question sa propre opinion. Cela peut être parfois délicat mais progressivement, chacun apprend à ne plus seulement penser de façon individuelle mais bel et bien dans l'intérêt du groupe. Chaque réflexion, chaque discussion va permettre de faire avancer l'équipe. D'une situation qui pouvait paraître handicapante au premier abord, nous en avons très vite compris les intérêts.

Ainsi, sans éteindre la fougue individuelle, l'obstination peu objective laisse place aux discussions de groupe bien plus constructives. On ne lisse pas les pensées, on y accorde de l'importance pour former un tout.

Devant éviter tout contact en cette période de crise sanitaire, nous avons été contraints de nous adapter à une nouvelle manière de gérer un projet. Troquer le face à face contre un écran ou l'émulation de la présence contre l'individualité d'une chambre n'est pas chose aisée.

Ainsi, nous avons essentiellement communiqué par le biais de visioconférences et mis en commun nos fichiers sur des plateformes dématérialisées, comme Git-hub, que nous avons découverts pour l'occasion. Maintenir une communication intensive était une des conditions sinéquanones de la réussite de ce projet.

Enfin, nous avons mis l'accent sur la gestion de notre temps. Etant facteur de stress, il était absolument question de ne pas en manquer. Il n'existe rien de plus contreproductif qu'un projet bâclé par une mauvaise organisation. En nous répartissant correctement le travail et en nous donnant des objectifs toutes les semaines, toute pression du « rush » du dernier moment a été évité.

c) Apport professionnel

Enfin, pour l'avenir, la création de ce projet va évidemment nous être utile.

Nous en avons abordé l'aspect technique et organisationnel, mais il ne faut pas en négliger l'aspect professionnel. Les sociétés d'informatique fonctionnent très souvent sous forme de « projet ». Cet exercice nous a donc en quelque sorte permis d'avoir un avant-goût du monde de l'entreprise en nous plongeant directement dans des conditions de travails qui pourraient être réelles. L'exigence de la qualité et de l'organisation sont de réels critères qui, si développés au stade de nos études, pourront être de réels atouts pour notre future vie professionnelle.

VII) Conclusion

Travailler sur ce projet a été une expérience enrichissante pour chacun d'entre nous sur beaucoup d'aspects différents. Il nous a permis de réaliser une application concrète des connaissances que nous acquerrons tout au long de l'année. De même, le plaisir de créer quelque chose par nous-même n'est pas négligeable. Grâce au temps et à l'énergie consacrée, nous nous sommes également découvert un nouveau désir de perfection, matérialisé notamment dans ce projet. Travailler ensemble était très enrichissant et au-delà de l'acquisition de compétences, nous a permis de passer de très bons moments.

Symbole de la réussite de ce projet, aussi bien sur le plan personnel que professionnel, nous avons prévu de reformer la même équipe pour le projet de TERD.

Merci d'avoir pris le temps de lire ce rapport de projet,

Gillian MASSE

Yann FORMER

Quentin BEAUCHET