BARRY Saïkou Ahmadou
KABEMBA Christelle
28 octobre 2016

Rapport Evaluator2.0

Présentation du sujet:

Il s'agit de développer une application console permettant de gérer les évaluations des étudiants dans différents cours. Nous avons choisi de développer les fonctionnalités suivantes:

- Afficher la liste de toutes les activités, professeurs et étudiants
- Afficher les informations relatives à une activité, un professeur ou un étudiant en particulier
- Ajouter une activité, un professeur ou un étudiant

Mode d'emploi:

Prérequis: Il faut installer le package json.Net (Newtonsoft.json) via package manager Nuget

Lors du lancement de l'application, nous nous retrouvons devant un menu proposant 3 sous-menu et quitter le programme. L'utilisateur doit faire un choix et en fonction de son choix il sera amener au menu suivant, arrêtera le programme ou vera un message d'erreur. En fonction du choix de l'utilisateur, dans les sous menu, il lui sera demandé d'entrer des informations. Si l'information est erroné un message d'erreur apparaitra.

Developpement:

1) Menu:

Pour faciliter la navigation entre les différentes fonctionnalités, il faut un menu. Pour la structure du menu, il y avait 2 possibilités:

- Ecrire le menu dans la fonction main ou une autre fonction et avoir grand nombre de switch case et condition. Pour une toute petite application c'est une option viable mais pour

une plus grosse application c'est difficilement maintenable et en plus il n'y pas de mise en pratique des concepts de POO.

- Ecrire le menu en POO, ce que nous avons opté.

La structure simplifié du menu:

Main Menu

- Menu 1
  - SubMenu 1
- Menu 2
  - SubMenu 1
  - SubMenu 2

Nous avons divisé la structure de notre menu en 3 classes (+ d'info cf Menu.cs).

- MenuItem qui représente une ligne d'un menu. Il renvoie soit vers un autre menu soit une action
- Menu qui est une collection de MenuItem. Il représente un menu de façon générale.
- MenuCollection qui représente un ensemble de menu et gère les events liés à ceux-ci.

2) **Fonctionnalités:**

Ensuite les différentes fonctionnalités de l'application ont été divisées dans des fonctions pour une meilleur lisibilité et maintenabilité.

Pour naviguer dans le menu, l'utilisateur est invité à entrer une valeur. Cette valeur est vérifiée et un message d'erreur est envoyé si elle n'est pas correcte. Pour les fonctionnalités d'ajout, des vérifications sont faites pour empêcher la duplication d'activités, de professeurs et d'étudiants dans « la base de donnée ».

3) **Import/Export to Json:**

Pour « la base de donnée » nous avons choisi un fichier json. On a utilisé la librairie JSON.Net pour la gestion de la sérialisation et la désérialisation de nos objets dans le fichier json. Nous avons crée une classe Parser qui va contenir tous les objets à sérialiser et désérialiser.

La méthode la plus rapide de convertir entre le texte JSON et un objet .NET utilise le JsonSerializer. Le JsonSerializer convertit objets .NET en leur équivalent JSON et retour en mappant les noms de propriétés d'objet .NET aux noms de propriété JSON et copie les valeurs pour vous.

Le soucis est qu'avec les classes du labo 2 ce n'était pas possible à cause de la classe abstraite et ces filles. Donc plutôt que de réécrire les 3 classes, nous avons opté pour la création d'une classe de « liaison » Bulletin. Elle stocke comme valeurs les informations nécessaire à la création d'une évaluation pour un étudiant. C'est cette classe qui est sérialiser et désérialiser. Les objets Evaluations ne sont pas stocker dans le fichier json. Ils sont générés à chaque démarrage de l'application. Pour une application de cette envergure ce n'est pas un problème mais si le json contient énormément d'objets Bulletin cette option n'est plus viable. En plus de cette classe, dans la classe Activity on ignore la propriété Teacher pour éviter une représentation imbriquée de l'objet Teacher dans le json.

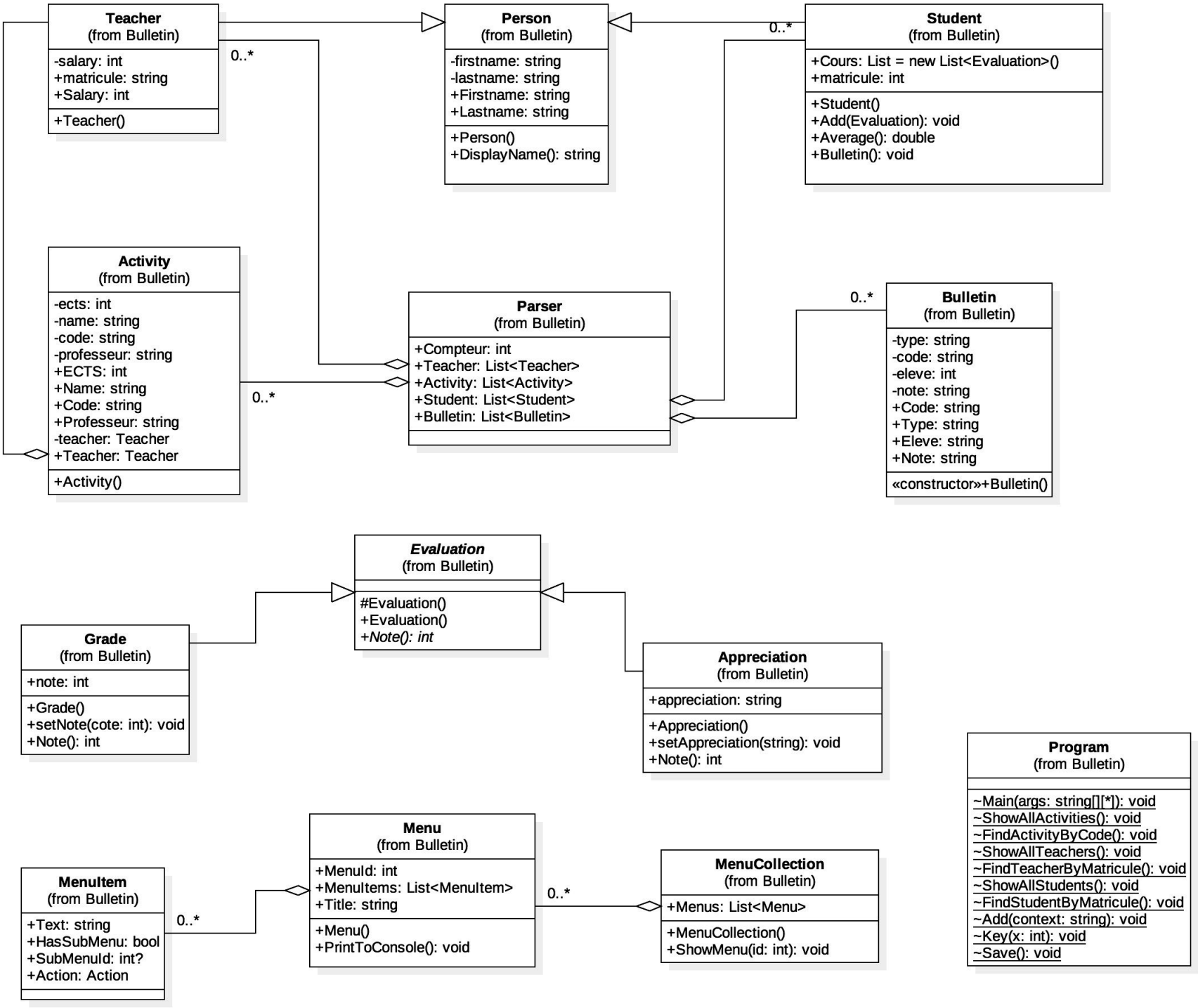
4) **Autres:**

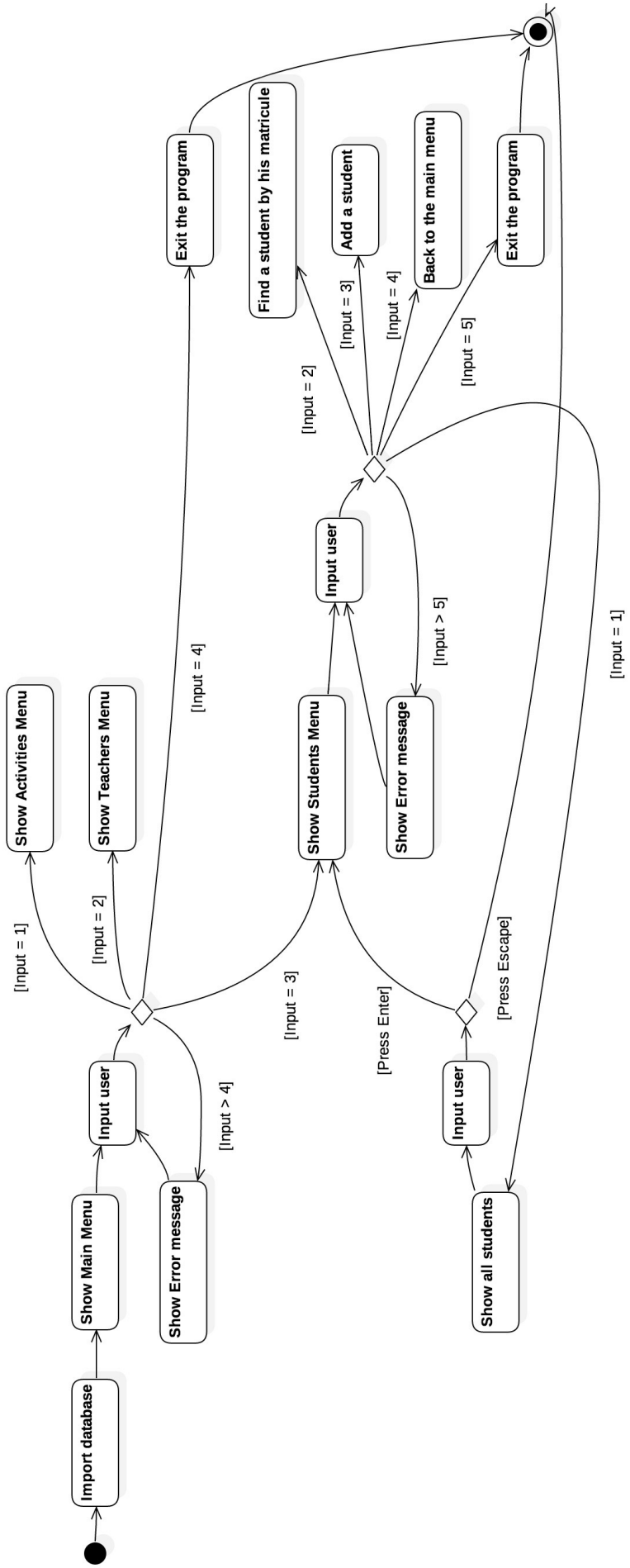
Il n'y a pas eu de grandes modifications dans les autres classes de l'application. On a seulement ajouté des propriétés pour simplifier la rechercher d'un element dans les différentes listes grâce à LinQ

Conclusion:

Ce projet nous a permis d'approfondir nos connaissances sur C# (notamment la découverte de LinQ et d'autres petits subtilités propre au langage) et d'appliquer les concepts vu en cours.

Annexes:


Teacher
(from Bulletin)
-salary: int
+matricule: string
+Salary: int
+Teacher()
0..*
Person
(from Bulletin)
-firstname: string
-lastname: string
+Firstname: string
+Lastname: string
+Person()
+DisplayName(): string
0..*
Student
(from Bulletin)
+Cours: List = new List<Evaluation>()
+matricule: int
+Student()
+Add(Evaluation): void
+Average(): double
+Bulletin(): void
Activity
(from Bulletin)
-ects: int
-name: string
-code: string
-professeur: string
+ECTS: int
+Name: string
+Code: string
+Professeur: string
-teacher: Teacher
+Teacher: Teacher
+Activity()
0..*
Parser
(from Bulletin)
+Compteur: int
+Teacher: List<Teacher>
+Activity: List<Activity>
+Student: List<Student>
+Bulletin: List<Bulletin>
0..*
Bulletin
(from Bulletin)
-type: string
-code: string
-eleve: int
-note: string
+Code: string
+Type: string
+Eleve: string
+Note: string
«constructor»+Bulletin()
Evaluation
(from Bulletin)
#Evaluation()
+Evaluation()
+Note(): int
Grade
(from Bulletin)
+note: int
+Grade()
+setNote(cote: int): void
+Note(): int
Appreciation
(from Bulletin)
+appreciation: string
+Appreciation()
+setAppreciation(string): void
+Note(): int
Program
(from Bulletin)
~Main(args: string[][*]): void
~ShowAllActivities(): void
~FindActivityByCode(): void
~ShowAllTeachers(): void
~FindTeacherByMatricule(): void
~ShowAllStudents(): void
~FindStudentByMatricule(): void
~Add(context: string): void
~Key(x: int): void
~Save(): void
MenuItem
(from Bulletin)
+Text: string
+HasSubMenu: bool
+SubMenuId: int?
+Action: Action
0..*
Menu
(from Bulletin)
+MenuId: int
+MenuItems: List<MenuItem>
+Title: string
+Menu()
+PrintToConsole(): void
0..*
MenuCollection
(from Bulletin)
+Menus: List<Menu>
+MenuCollection()
+ShowMenu(id: int): void


Import database
Show Main Menu
Input user
Show Error message
[Input > 4]
[Input = 1]
Show Activities Menu
[Input = 2]
Show Teachers Menu
[Input = 4]
Exit the program
[Input = 3]
Show Students Menu
Input user
[Input = 2]
Find a student by his matricule
[Input = 3]
Add a student
[Input = 4]
Back to the main menu
[Input = 5]
Exit the program
[Input > 5]
Show Error message
[Press Enter]
Input user
[Press Escape]
Show all students
[Input = 1]