SAE S2.02 - Rapport pour la ressource Dev-OO

De Gaël DIERYNCK, Romain HARLAUT, Amaury VANHOUTTE

Groupe A - A1-2025

Version 1

A. Structuration du projet

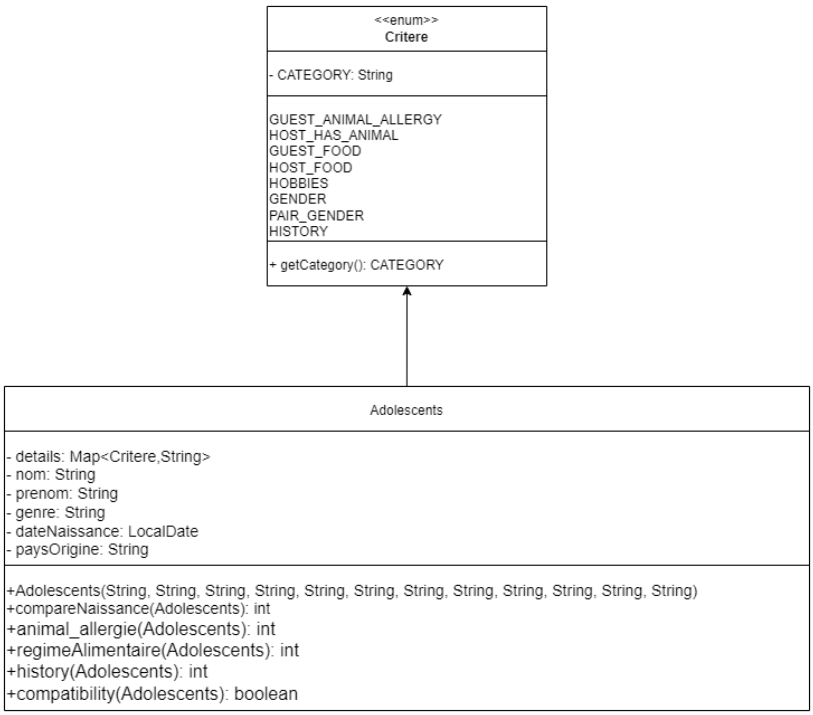
Le projet est structuré sous la forme d'une arborescence de fichiers comme suit :

- Les librairies nécessaires au bon fonctionnement du programme se trouvent dans le repertoire `lib/` - Les ressources du programme dans le repertoire `src/` tandis que les classes de tests se trouvent dans le repertoire tests. - Le repertoire `graphes` contient le rendu pour la partie graphe du projet. - Les repertoires

`doc/` et `bin/` servent respectivement à stocker la documentation des différentes classes du projet, ainsi que les fichiers compilés de ces mêmes classes.

B. Diagramme UML

Notre code se décompose actuellement pour la v1 en 2 classes puis une classe de test représentés par ce diagramme UML :



C. Implémentation des fonctionnalités

Dans cette première version, nous devions effectuer la base du programme de cette SAE : Créer des adolescents et tester leur compatibilité selon certains critères.

1 - L'énumération `Critere` L'énumération Critere.java permet d'énumerer par le type Critere les contraintes du **GUEST_ANIMAL_ALLERGY**, du **HOST_HAS_ANIMAL**, du **GUEST_FOOD**, du **HOST_FOOD**, des **HOBBIES**, du **GENRER**, du **PAIR_GENDER** et du **HISTORY**.

2 - La classe `Adolescents` La Classe Adolescents.java permet de créer un adolescent pour ensuite le comparer aux autres adolescents.

La classe fournit le constructeur Adolescents qui crée un adolescent avec toutes les caractéristiques liées aux allergies envers les animaux, aux régimes alimentaires, aux hobbies, à l'historique , aux genres et à l'identité.

La classe fournit une fonction `int compareNaissance(Adolescents)` qui renvoie `true` si la préférence liée à un écart d'âge inferieur à 1 an et demi est respectée, `false` sinon.

Une fonction `int animal_allergie(Adolescents)` permet de savoir si l'hôte peut accueillir un visiteur malgré la présence d'un animal, ou si une allergie aux animaux vient du visiteur.

La fonction `int regimeAlimentaire(Adolescents)` permet de savoir si l'hôte et le visiteur ont le même régime alimentaire.

Ensuite, la fonction `int history(Adolescents)` permet de savoir si l'hôte et le visiteur veulent réitérer leur voyage avec la même personne que l'année précédente.

Enfin, une fonction `boolean compatibility(Adolescents)` qui renvoie l'affirmation qu'un hôte et un visiteur valident les contraintes imposées par les allergies animaliaires, le régime alimentaire et l'historique.

D. Tests réalisés

Nous avons réalisé une classe de tests destinée à vérifier le bon fonctionnement des méthodes principales. Elle permet actuellement de tester 6 méthodes différentes. Chacune d'entre elle teste différents cas et contrôle le résultat renvoyé par la fonction.

Version 2

A. Diagramme UML

Notre code se décompose actuellement pour la v2 en 5 classes puis deux classe de test représentés par ce diagramme UML :

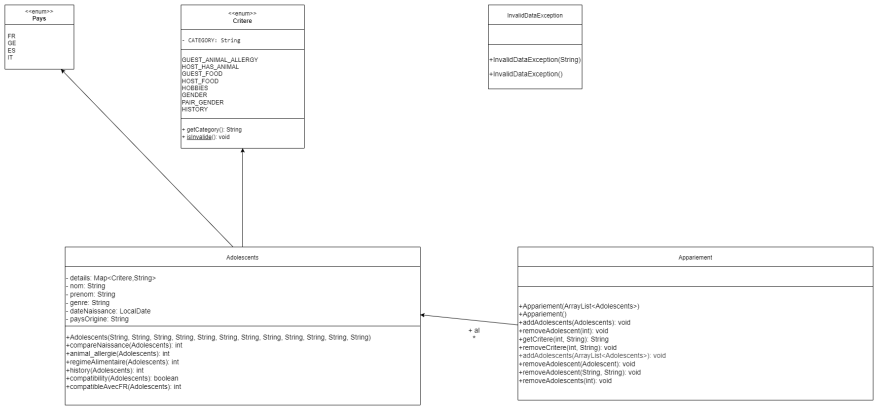


Figure 1: UML V2

B. Implémentation des fonctionnalités

Dans cette deuxième version, nous devions effectuer l'extention du programme de cette SAE : Crée des appairements d'adolescents en prenant en compte leur compatibilité

1 - L'énumération `Pays` L'énumération Pays.java permet d'énumerer tout les pays d'origine des Adolescents.

2 - L'énumération `Critere` ajout d'une fonction qui permet de vérifier que le critere saisie est un critere qui existe

3- La classe `Adolescents` Une fonction `int compatibleAvecFR(Adolescents)` a été ajouté qui permet de savoir si l'hôte peut accueillir un visiteur même si l'un des deux est français.

Ensuite, la fonction `int history(Adolescents)` a été modifié pour prendre en compte la compatibilité avec un français.

4 - La classe `Appariement` La Classe Appariement.java permet de créer une liste adolescent.

La classe fournit le constructeur Appariement qui crée une liste d'adolescent ou une liste d'adolescent vide.

La classe fournit une fonction `void addAdolescent(Adolescents a)` qui permet de rajouter un adolescent a la liste.

La classe fournit une fonction `void addAdolescent(Arraylist<Adolescents> a)` qui permet de rajouter des adolescents a la liste.

Des fonctions `void removeAdolescent(int id)`, `void removeAdolescent(Adolescents a)` et `removeAdolescent(String nom, String prenom)` permet de retirer un adolescent de la liste en fonction de son id , d'un adolescent particulier ou du nom et du prenom.

Des fonctions `void removeAdolescents(int nb)` permet de retirer les nb premier adolescents de la liste.

Une fonction `int size()` qui permet de connaitre le nombre d'adolescent de la liste.

5 - La classe `InvalidDataException` Permet de s'occuper des erreurs de saisie.

C. Tests réalisés

Nous avons réalisé une classe de tests destinée à vérifier le bon fonctionnement des méthodes principales pour l'appariement ainsi que pour les critères.

`AppariementTest` permet actuellement de tester 4 méthodes différentes, `CritereTest` s'assure quant à elle que l'exception `InvalidDataException` soit levée au moment propice.

Chacune d'entre elles teste différents cas et contrôle le résultat renvoyé par la fonction.

La classe de test pour les adolescents a été mise à jour pour prendre en compte les nouvelles fonctions et les mises à jour des fonctions existantes.

Version 3

A. Diagramme UML

Notre code se décompose actuellement pour la v3 en 11 classes puis 5 classes de test représentés par ce diagramme UML :

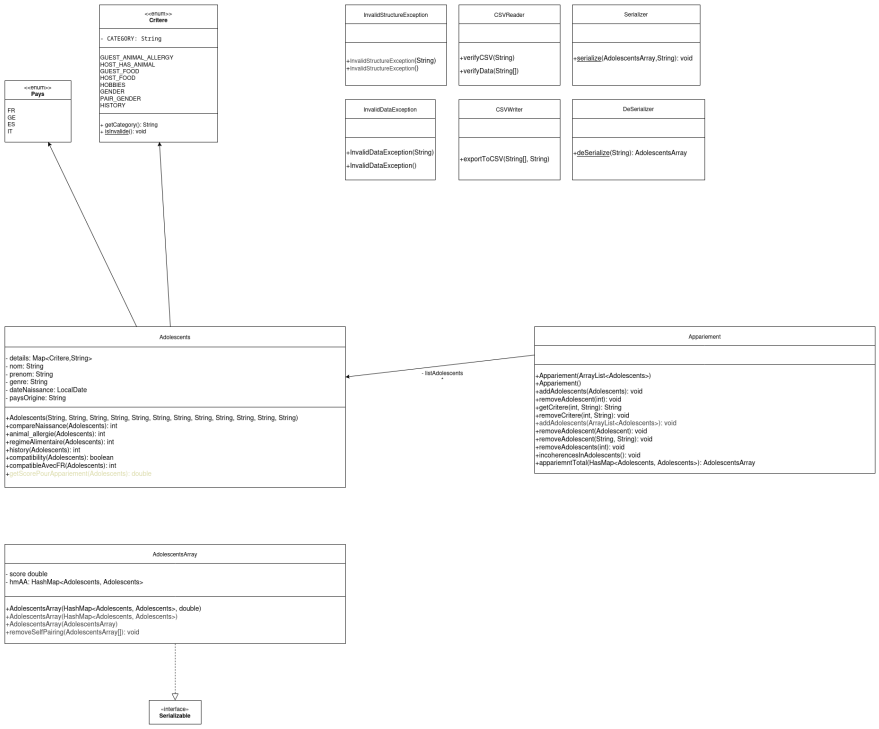


Figure 2: UML V3

B. Implémentation des fonctionnalités

Dans cette troisième version, nous devions effectuer l'extention du programme de cette SAE : Crée des appairements d'adolescents en prenant en compte leur compatibilité à partir de fichier

2 - L'énumération `Critere` ajout d'une fonction qui permet de vérifier que le critere saisie est un critere qui existe

1 - La classe `InvalidStructureException` Permet de s'occuper des erreurs de structure des fichiers qui contiennent adolescents.

2 - La classe `CSVReader` Permet vérifier le contenue d'un fichier csv contenant des adolescents

La classe fournit une fonction `boolean verifyCSV(String file)` qui permet de vérifier que chaque ligne du fichier correspont bien a un adolescent

Elle fournit aussi une fonction `boolean verifyData(String[] data)` qui permet vérifier qu'une ligne correspont bien a un adolescent

3 - La classe `CSVWriter` Permet vérifier le contenue d'un fichier csv contenant des adolescents

La classe fournit une fonction `void exportToCSV(String[] data,String path)` qui permet d'exporter des données dans un fichier csv

4- La classe `Adolescents` Une fonction `double getScorePourAppariement(Adolescents other)` a été ajouté pour permettre de connaître le score d'affinité entre deux adolescents.

5- La classe `Appariement` Une fonction `AdolescentsArray[] appariementTotal(HashMap<Adolescents, Adolescents> paireAdo)` a été ajouté pour connaître le score d'affinité pour chaque paire.

Une fonction `void incoherencesInAdolescents()` a été ajouté pour permettre de gardé que les adolescents qui des données cohérentes.

6- La classe `AdolescentsArray` La Classe AdolescentsArray.java permet de créer une HashMap qui contient des pairs d'adolescents.

La classe fournit le constructeur AdolescentsArray qui crée une HashMap de pair d'adolescents.

Une fonction `void removeSelfPairing(AdolescentsArray o)` a été ajouté pour permettre la suppresion d'une paire d'adolescents.

7- La classe `Main` La Classe Main.java permet de monter les incohérences dans une possible nouvelle instance d'Adolescents.

8- La classe `Serializer` La Classe Serializer.java permet de **sérialiser** une classe.

9- La classe `DeSerializer` La Classe DeSerializer.java permet permet de **désérialiser** une classe.

C. Tests réalisés

Nous avons réalisé deux classe de tests destinée à vérifier le bon fonctionnement des méthodes principales pour la lecture et l'écriture d'un fichier csv.

`CsvWriterTest` permet actuellement de tester l'export de données, `CSVReaderTest` permet actuellement de testés 4 méthodes différentes.

Nous avons aussi réalisé une classe de tests destinée à vérifier le bon fonctionnement de la sérialisation et la désérialisation.

`SerializerDeserializer` permet actuellement de tester la sérialisation et la désérialisation.

Chacune d'entre elles teste différents cas et contrôle le résultat renvoyé par la fonction.

La classe de test pour les adolescents a été mise à jour pour prendre en compte les nouvelles fonctions et les mises à jour des fonctions existantes.

La classe de test pour les appariements a été mise à jour pour prendre en compte les nouvelles fonctions et les mises à jour des fonctions existantes.

Version 4

Le code comporte **11 classes et 6 classes de tests**.

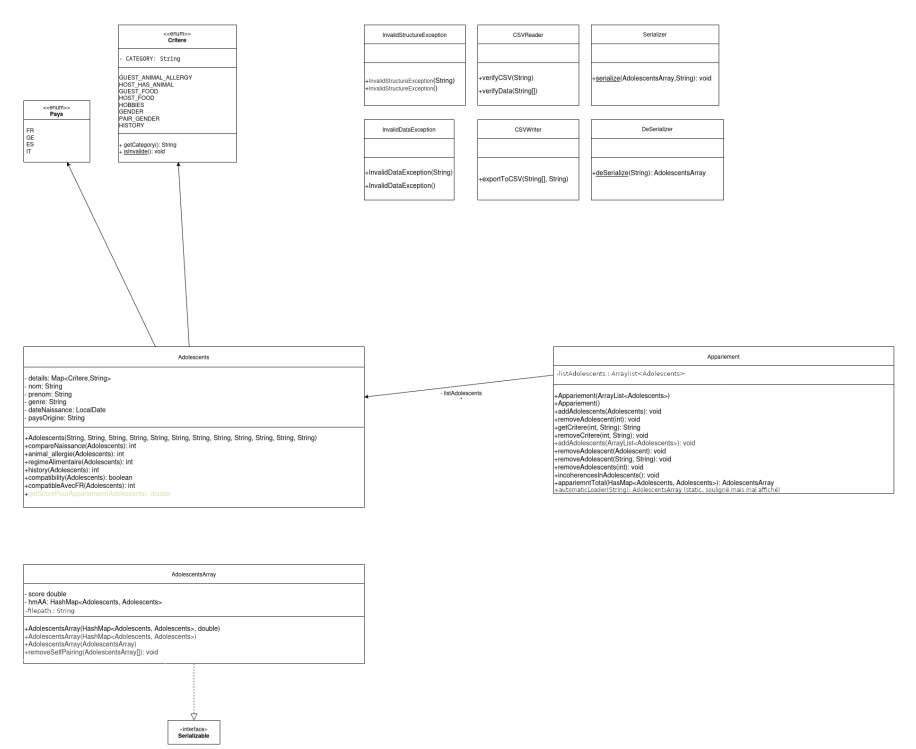


Figure 3: UML V4

A. Diagramme UML

Pour cette version finale, le code a été modifié pour être plus lisible et plus cohérent.

B. Implémentation des fonctionnalités

Pour cette version finale, il nous était demandé de réaliser une méthode permettant le prétraitement des adolescents à partir d'un fichier sérialisé sur demande. Néanmoins, la classe Appariement accueille une nouvelle méthode statique `AdolescentsArray automaticLoader(String path)` qui permet de charger automatiquement les adolescents à partir d'un fichier class sérialisé, s'il existe, sinon la configuration se comporte comme précédemment.

La signature de la méthode est la suivante :

```
public static AdolescentsArray automaticLoader(String path)
```

Anciennes et nouvelles fonctionnalités :

2 - L'énumération `Critere` Ajout d'une fonction qui permet de vérifier que le critère saisi est un critère qui existe.

1 - La classe `InvalidStructureException` Permet de s'occuper des erreurs de structure des fichiers qui contiennent des adolescents.

2 - La classe `CSVReader` Permet vérifier le contenue d'un fichier csv contenant des adolescents

La classe fournit une fonction `boolean verifyCSV(String file)` qui permet de vérifier que chaque ligne du fichier correspont bien à un adolescent

Elle fournit aussi une fonction `boolean verifyData(String[] data)` qui permet vérifier qu'une ligne correspont bien a un adolescent

3 - La classe `CSVWriter` Permet vérifier le contenue d'un fichier csv contenant des adolescents

La classe fournit une fonction `void exportToCSV(String[] data,String path)` qui permet d'exporter des données dans un fichier csv

4- La classe `Adolescents` Une fonction `double getScorePourAppariement(Adolescents other)` a été ajouté pour permettre de connaître le score d'affinité entre deux adolescents.

5- La classe `Appariement` Une fonction `AdolescentsArray[] appariementTotal(HashMap<Adolescents, Adolescents> paireAdo)` a été ajouté pour connaître le score d'affinité pour chaque paire.

Une fonction `void incoherencesInAdolescents()` a été ajouté pour permettre de gardé que les adolescents qui des données cohérentes.

6- La classe `AdolescentsArray` La Classe AdolescentsArray.java permet de créer une HashMap qui contient des pairs d'adolescents.

La classe fournit le constructeur AdolescentsArray qui crée une HashMap de pair d'adolescents.

Une fonction `void removeSelfPairing(AdolescentsArray o)` a été ajouté pour permettre la suppresion d'une paire d'adolescents.

7- La classe `Main` La Classe Main.java permet de monter les incohérences dans une possible nouvelle instance d'Adolescents.

8- La classe `Serializer` La Classe Serializer.java permet de **sérialiser** une classe.

9- La classe `DeSerializer` La Classe DeSerializer.java permet permet de **désérialiser** une classe.

11 - La classe MainV2 La classe MainV2.java permet de montrer les incohérences dans une possible nouvelle instance d'AdolscentsArray.

12 -La classe Appariement Dans la classe Appariement.java, la méthode AdolescentsArray automaticLoader(String path) a été ajoutée pour permettre le chargement automatique des adolescents à partir d'un fichier sérialisé ou de poursuivre la configuration manuelle.

C. Tests réalisés

Nous avons réalisé une classe de tests destinée à vérifier le bon fonctionnement de la nouvelle méthode `automaticLoader`.

Une nouvelle classe de test `SerializerDeserializerTest` permet actuellement de tester la sérialisation et la désérialisation des adolescents. Les autres tests ont été mis à jour pour prendre en compte le nouveau fonctionnement des classes.

Pour finir, la stucture du projet se résume comme suit :

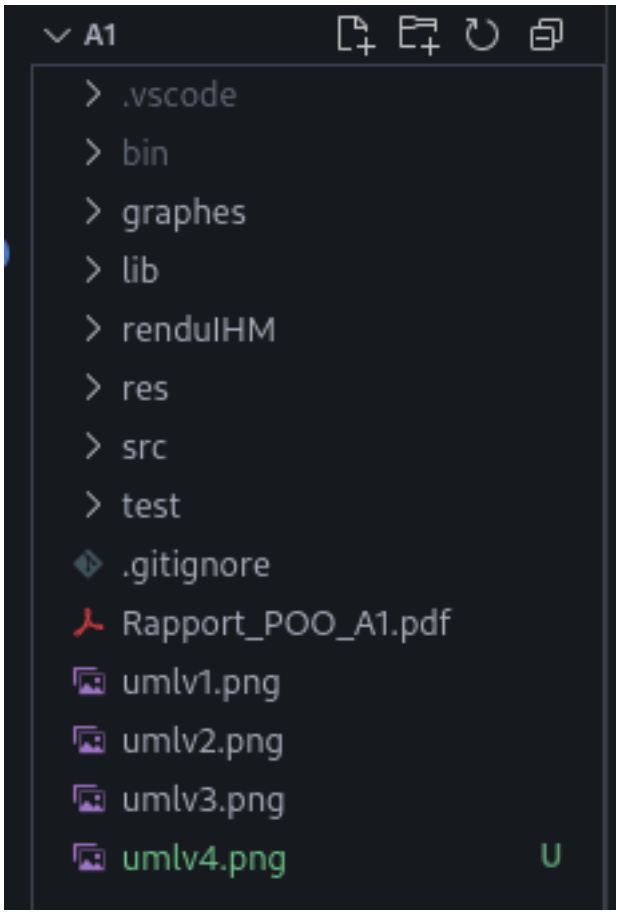


Figure 4: Arborescence finale

Merci de votre lecture et de votre attention.

Nous espérons que ce rapport vous a permis de mieux comprendre notre projet et les différentes fonctionnalités implémentées.

Merci pour l'encadrement de cette SAE et pour les retours constructifs que nous avons reçus tout au long de son développement.