SAÉ S2.01/02

Ethan Robert, Jonas Facon, Antonin Marouzè

Mai 2025

Contents

1 Rendu des Semaines 1 et 2

Pour cette première semaine.

1.1 Pourquoi ce diagramme UML ?

Nous avons décidé d'intégrer ce diagramme pour deux principales raisons : - **Plannifier** correctement le projet et les classes à développer, afin de **fournir une bonne base** pour la suite du développement, et notamment l'interface utilisateur, qui doit s'appuyer sur une codebase solide - **Repérer et corriger les éventuelles erreurs** de conception du code, que ce soit dans les **structures de données utilisées** ou leur **manière d'interagir**, afin d'éviter les ralentissements dûs à de lourds changements plus tard (il vaut mieux prévenir que guérir).

1.2 Structure principale du code

- La classe abstraite `DataType`, mère de toutes les autres classes, prévoit l'importation et exportation de ses données au format CSV (via deux méthodes abstraites `boolean importCSV(String)` et `boolean exportCSV(String)`), utilisées pour gérer les fichiers externes.
- La classe `TeenagerInventory` encapsule plusieurs objets de type `Teenager` au sein d'un `ArrayList<Teenager>`, permettant ainsi de gérer de manière centralisée tous les adolescents de la plateforme d'échanges. En dehors de cela, son attribut principal est `CriteriaMap<String, String>`, qui

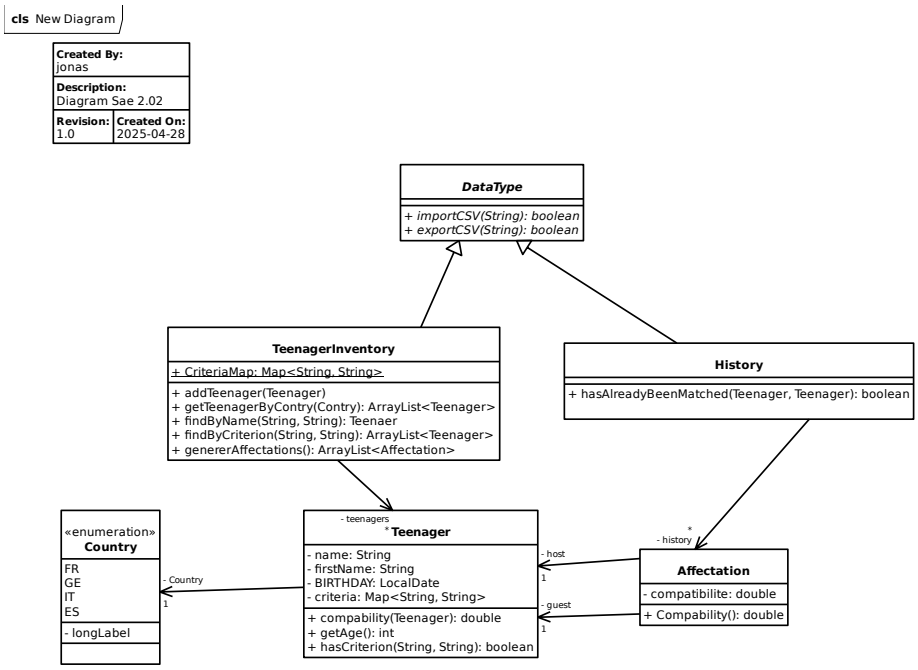

Figure 1: Diagramme UML du projet

renseigne tous les critères existants, ainsi que leurs types (`T` pour du texte, `B` pour des booléens, etc). Elle possède également différentes **méthodes** :

1. `addTeenager(Teenager)`
   Ajoute un objet **Teenager** à la `ArrayList`.
2. `getTeenagersByCountry(Country)`
   Retourne la liste des **Teenagers** dont le pays correspond à celui passé en paramètre.
3. `Teenager findByName(String, String)`
   Retrouve un **Teenager** à partir de son prénom et de son nom.
4. `findByCriterion(String, String)`
   Renvoie tous les **Teenagers** correspondant à un même critère (ex. : “ville”, “école”, etc.).
5. `genererAffectations()`
   Génère et retourne une `ArrayList` d’**Affectation**.

- La classe `Teenager` qui modélise chaque adolescent, à travers des **attributs** :
  1. `private String name` : le nom de l’adolescent

2. `private String firstName` : le prénom de l'étudiant
3. `private final LocalDate BIRTH` : une constante pour la date de naissance de l'adolescent
4. `private Map<String, String> criteria` : une Map permettant d'inscrire les critères de compatibilité (rhédibitoires ou non), en suivant le modèle établi par `TeenagerInventory.CriteriaMap`

- Mais également des **méthodes** :

  1. `double compatibility(Teenager)` qui donne le score de compatibilité probable avec un autre adolescent, compris entre 0 et 1.
  2. `int getAge()` qui calcule l'âge d'un adolescent
  3. `boolean hasCriterion(String, String)` qui retourne un booléan suivant si un adolescent possède un critère ou non.

- La classe `Affectation` qui permet d'encapsuler 2 objets `Teenager` (1 hôte et 1 visiteur) avec une méthode `double compatibility()`, donnant un score (compris entre 0 et 1) de compatibilité entre ces deux adolescents.

- La classe `History` permet d'ordonner le tout, en fournissant une `ArrayList<Affectation>` d'affectations déjà effectuées.

1.3 Enum et objets supplémentaires

- L'enumération `Country` contient les différents pays sous format abbréger elle contient comme méthode :
  1. `longLabel`Renvoie le nom complet d'un Pays.