Adham Berrakane
Nathan Dorny
Quentin Moutté

Rapport POO

2022/2023

Sommaire

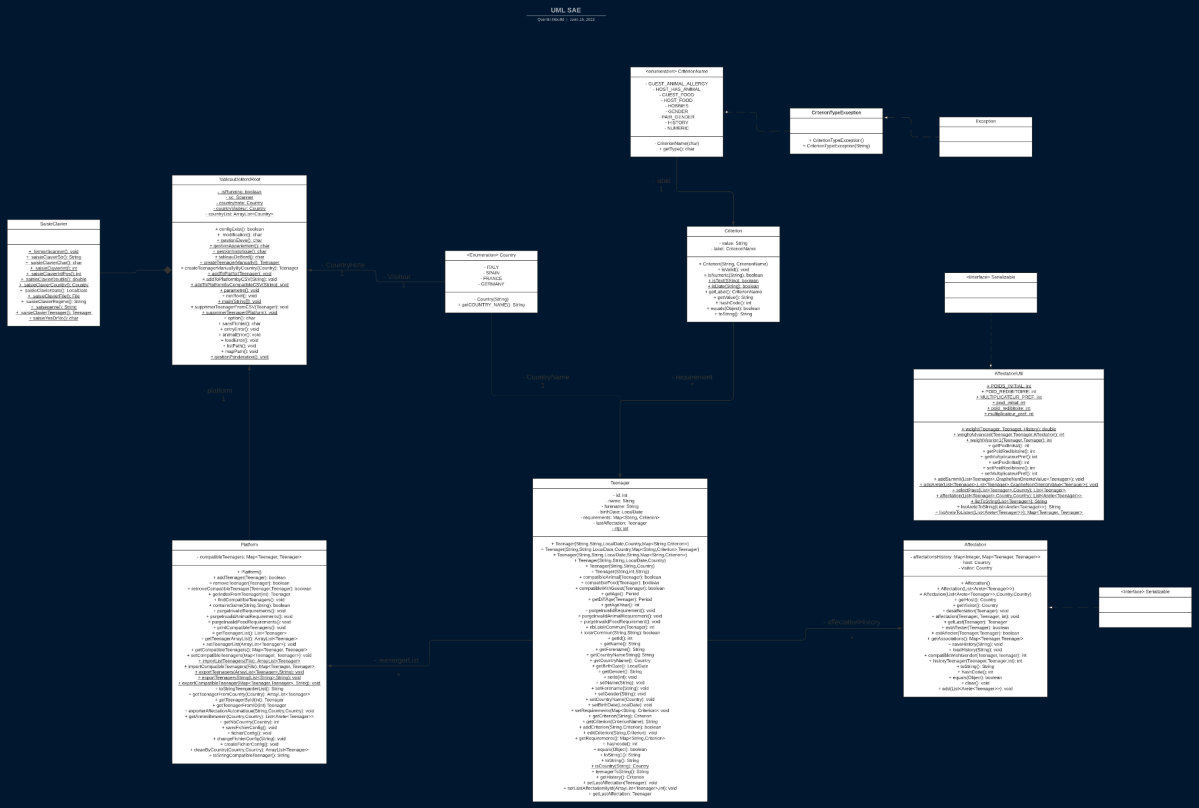
Lancement de l’application :

Pour lancer l’application, vous avez besoin de de vous trouver à la racine du projet, puis d’exécuter l’archive jar (nom du jar) avec la commande :

```
java --module-path lib --add-modules
javafx.controls,javafx.fxml -jar (nom du jar)
```

Si il n’y a pas de jar, alors ouvrir la classe TableauDeBord.java et cliquer sur run dans l’IDE.

Diagramme UML :



Voici le lien : UML

Réflexion sur les mécanismes objets vus en cours que vous avez mis en œuvre et leurs intérêts le cas échéant :

Dans cette UML nous remarquons que un peut tous les principes java on été utiliser les voici :

- L'usage d'interface
- Usage d'héritage
- Méthode délégué
- Des énumérations enrichie
- Usage de class qui hérite de la classe Exception
- L'usage d'interface Serializable qui va nous permettre d'enregistrer nos données dans des fichiers facilement transportables.
- Classe interne pour la gestion des saisie

Analyse Technique :

Voici une analyse technique et critique de l'implémentation des fonctionnalités demandées :

Compatibilité entre adolescents (contraintes) :

Notre méthode qui se nomme compatibleWithGuest() va prendre en paramètre une Teenager et va le comparer au Teenager courant ici on le voit grâce au 'this'. Notre méthode vérifie dès le début si le Teenager

passer en paramètre n'est pas le même que celui qui est courant ce qui permettra de ne pas comparer les mêmes Teenager entre eux.

Ensuite notre méthode va faire appel à 3 autre méthode ce qui est bien car cela décompose bien le code :

- compatibleAnimal qui en paramètre un Teenager et renvoie un boolean, cette méthode va permettre de savoir si ce qui permet de savoir si un adolescent est compatible avec un autre adolescent en termes de critères animal.

- compatibleFood qui en paramètre un Teenager et renvoie un boolean, cette méthode va permettre de savoir si un adolescent est compatible avec un autre adolescent en termes de critères nourriture.

- nbLoisirWithGuest qui en paramètre un Teenager et renvoie un int, cette méthode va nous permettre de savoir combien les 2 Teenager ont de Loisirs en commun cela va surtout nous servir pour la règle spécifique de la France avec les autre pays mais aussi dans le poids de l'arête lors du calcule de compatibilité.

Cette méthode va donc être très efficace algorithmiquement et pourra répondre à la demande de compatibilité.

Gestion de la validité des critères par un mécanisme d'exception :

Pour gérer la validité des critères par un mécanisme d'exception notre travaille c'est décomposer en 2 partie :

1er partie :

- Réalisation d'un classe qui hérite de Exception qui va renvoyer une CriterionTypeException et de la cela va renvoyer un print qui qu'il y a une erreur Criterion.
-

2eme partie :

- Vient maintenant la réalisation de la méthode isValid dans la class Criterion , cette méthode va permettre de vérifier tout les valeur des criterion entrée et va propager l'erreur quand cette méthode sera utilisé autre part.

Cette méthode va permettre de gérer tous les cas que ce soit le null ou même des mauvaise valeur et va permettre donc de gérer comme il le faut la validation des critères.

Développement des règles spécifiques de compatibilité pour certains pays :

Pour gérer cela plusieurs méthodes vont être écrites pour gérer cela :

- Dans la méthode compatibleWithGuest si le pays du Teenager courant est le même que celui entré en paramètre alors il sont incompatibles.
- Dans la méthode compatibleWithGuest pour la France si celle n'a pas au moins un loisir en commun avec l'autre Teenager alors il sont incompatibles.

Cette implémentation ici va nous permettre de répondre aux exigences demandées pour certains pays.

Import par fichier de format csv, répondant à la structure donnée :

Nous pouvons importer depuis un fichier csv des listes d'étudiants et des listes d'appariement d'étudiants, avec respectivement importListeTeenager et importCompatibleTeenager, afin de les ajouter aux attributs de la classe Platform.
Nous avons fait en sorte que l'importation réussisse quelles que soient l'ordre des critères des étudiants.
Pour pouvoir importer, il y a seulement besoin de donner le chemin vers le fichier demandé.

Export d’un résultat sous format csv :

Nous pouvons également exporter des listes d'adolescents grâce à exportTeenager, qui prend en compte une liste d’étudiants, et des listes d’appariements grâce à exportCompatibleTeenager, qui prend en compte une liste d’appariement.
Ces méthodes exporte les informations aux formats csv et peuvent être importées pour être utilisées grâce aux méthodes d’importation vues précédemment.

Gestion de l’historique par sérialisation binaire :

Afin de faire une gestion de l’historique alors nous avons créé la classe AffectationUtil celle-ci va nous permettre plusieurs actions sur celle-ci tels que la modifier , supprimer des appariements et en ajouter. Mais primordialement stocker un historique dans un une HashMap avec les 2 Teenagers associer précédemment , chaque fois qu’un appariement de 2 Teenager sera fait alors obligatoirement cette association sera enregistrée dans l’historique.
Dans cette classe va aussi se trouver une méthode qui vérifiera les choix concernant l’historique et retourne un entier qui sera utile pour le calcul des arete lors de la création des appariements.

Prétraitement des critères des adolescents sur demande :

Nous n’avons pas réussi à implémenter le prétraitement manuellement.
On avait une possible solution, mais qui n’est pas fonctionnelle, cela aurait été de modifier le fichier de configuration manuellement à l’aide de la méthode changeFichierConfig, mais elle n’a pas été implémenté dans le main, bien qu’elles soient existantes.

Prétraitement automatique des critères en cas d’existence d’un fichier de configuration à un emplacement prédéfini

Nous avons ajouté un fichier de configuration csv servant à faire certaines tâches dès le lancement de l'application.
Il y a 5 critère dans ce fichier:

CRITERION_TYPE: lorsque sa valeur est à true, il traite les critere mal rempli,
ANIMAL_ALLERGY: lorsque sa valeur est à true, il traite les erreurs et les incohérence liée au animaux,
FOOD_ALLERGY: lorsque sa valeur est à true,il traite les erreurs liées à la nourriture,
TEENAGER_LIST: lorsque sa valeur correspond a un chemin vers un fichier d'adolescent csv, il l'importe automatiquement,
COMPATIBLE_TEENAGER_LIST: lorsque sa valeur correspond a un chemin vers un fichier d'appariement csv, il l'importe automatiquement.

Analyse Quantitative :

Voici une analyse quantitative/qualitative des tests que vous avez réalisés, en rapport aux notions vues en cours :

Nous avons de nombreux tests dans les classes les plus importantes, comme platform, teenager et affectation, avec environ 4 tests dans chaque classe, sauf Teenager qui en compte 9.
Les classes de test ne visent que les méthodes les plus importantes, qui regroupent elles même des d'autre méthode, comme par exemple la méthode weight dans la classe affectation ou purgeInvalidRequirement de Teenager.

La répartition du travail :

Dans cette SAE chacun a pu participer dans le code on a sut se répartir comme il fallait le travail , travailler dans des classe différente à certain moment pour ne pas se marcher sur les pieds mais en sois tout le monde y a mis du sien et ce projet n'aura pas pu aboutir si chacun n'y avais pas mis du sien comme maintenant. Ce projet a été réparti en quelque sorte ou tout le monde a pu mettre à disposition ces connaissance afin de permettre à la réussite du projet chacun de nous a su trouver des innovations dans le code afin de permettre de répondre au attente demander dans le sujet.
Enfin je finirai sur le faite que tout le monde est mis la main à la patte même si certaine méthode n'ont pas été taper pas quelqu'un qu'elle ne vient pas de lui , notre but n'a pas été la course a celui qui aller avoir le plus de code mais notre but a été de faire un code optimisé qui reprend tous les principe vus en cours.