Rapport POO

Bouin Julien, Milleville Paul, Leprêtre Mathys

Contents

1 Rapport POO

1.1 Pré-requis

1.1.1 1.Position des ressources nécessaires

Dans notre projet nous avons classé les différents fichiers dans plusieurs dossiers. - src : pour les fichier .java concernant la partie back du projet - JAVAFX : pour les fichiers .java concernant la partie front du projet et .fxml (IHM) - bin : pour les fichiers binaires - doc : les rapports et rendus demandées mais également pour le fichier de configuration - img : pour les photos des graphes générées - test : pour les fichiers .java permettant d'effectuer les tests

1.1.2 2.Commande de lancement

pour lancer le projet il faut utiliser la commande :

```
javac Interface.java
/usr/bin/env /usr/lib/jvm/jdk-19.0.2/bin/java @/tmp/cp_aoqppbnv4mlmh380nmjjn2xtv.
argfile Interface
```

1.2 Diagramme UML

1.3 Analyse de l'implémentation des fonctionnalités demandées

- La vérification de la validité des critères exprimés

Pour la validité des critères nous avons mis en oeuvre un mécanisme d'exception (try/catch) via la méthode "isValid" qui permet de savoir si un critère est valide ou non. La validité des critères passent aussi par la méthode "purgeInvalidRequirements" qui permet à partir d'une ArrayList de retirer les étudiants dont un des critères n'est pas bon.

- La sélection des adolescents à retirer Pour la sélection d'étudiant à retirer nous avons deux options :
  - soit les informations sur l'étudiant ne sont pas suffisant, dans ce cas, l'étudiant ne sera pas créé. Donc ceci est effectuer dès l'import d'un fichier CSV avec la méthode "importTeenagerFromCSV" qui effectue ceci.
  - soit avec une solution, qui est plus explicite que celle ci-dessus, l'utilisation de la méthode "suppEtu", qui utilise implicitement la méthode "suppEtuCSV", qui permet de supprimer un étudiant d'une arrayList mais aussi du CSV utilisé pour l'import (pour notre exemple le fichier csv "AdosAleatoires").
- L'implémentation de règles de compatibilité propres à certaines nationalités

Comme dit précédemment, on sait que deux nationalités ne peuvent pas être apparier et que pour les français, ils doivent avoir au moin un loisir en com-

mun avec l'autre étudiant. Nous avons donc fait en sorte avec la méthode "isCompatibleWithCountry" puisse gérer ces contraintes.

- La gestion des imports/exports

Pour la gestion des imports et des exports, l'utilisation du fichier Plateform.java, permet d'importer un fichier csv, exporter les resultats d'apariement au format csv. Il permet aussi de faire une serialisation mais aussi lire un fichier serialisé, ici pour ce qui a été demander cela concerne l'historique.

Pour l'import des fichiers, il y a une contrainte, car en effet, les données sur l'étudiant doivent être mis dans un ordre bien spécifique, pour eviter les erreurs dans les données. En effet, si un étudiant a une critère qui est invalide ou incohérent celui-ci ne sera pas importé, ce qui peut sembler logique. Donc le fichier pour l'import doit etre correctement contstitué.

Pour l'export des fichiers, nous avons créer une méthode "exportResults" permettant de le faire. Cette méthode permet à partir d'un résultat, de type "List<Arete>" de créer un fichier et de mettre sur chacune des lignes les informations suivantes : ID;FIRSTNAME;NAME;COUNTRY;ID2;FIRSTNAME2;NAME2;COUNTRY2. Un exemple de fichier d'exportation est présent dans le dossier doc de notre arborésence avec pour nom "affectations.csv".

- La gestion de l'historique des affectations

La gestion de l'historique utilise la classe History. On utilise une HashMap pour stocker des Teenager (un en tant que clé et un autre en tant que valeur). Pour inclure l'historique dans l'appel de la fonction de calcul de poids (weight), il est nécessaire de fournir un objet de type History. Ainsi, on peut appeler la méthode history sur l'objet History.

- La prise en compte de l'historique dans les affectations

L'historique est pris en compte exclusivement lors du calcul des poids dans les affectations. La fonction history renvoie un entier qui représente la présence d'antécédents positifs ou négatifs. Cet entier doit être ajouté au poids total de l'affinité. En effet, cette historique a été tester dans des scénarios bien précis mais n'a pas pu etre utilisé dans notre cas d'exemple (notre application) car ceci dans notre cas est le premier séjour des étudiants.

- Le traitement des données chargées

Pour le traitement des données chargées, cela a été fait dans le fichier "Interface.java" avec une méthode bien spécifique qui est "suppEtuWithFileConfiguration" qui permet donc à partir d'une chaine de caractères, qui est un chemin d'un fichier, de supprimer via les données contenus dans le fichier de configuration de supprimer les étudiants qui ne correspondent pas . Ce fichier de configuration est présent dans le dossier "doc" de notre arboréscence, et contient deux lignes. La première ligne est l'entete qui est donc critère et value. La deuxième ligne est la ligne où l'utilisateur mettra le critère et la value qu'il veut pour qu'ensuite à partir de cela les étudiants fuent supprimer. A noter, que l'utilisateur doit

noter correctement le critère et la value, donc comme indiqués dans les fichiers csv d'exemple sinon cela ne marchera pas. Mais aussi que l'utilisateur peut noter seulement sur la deuxième ligne le critère et la value et donc qu'il est possible seulement d'indiquer un critère et non pas plusieurs. A noter que l'exemple final est lancé de manière automatique avec un fichier de configuration (configuration.csv) contenant des données.

1.4 Analyse des tests réalisés

En ce qui concerne les tests du projet, nous avons fait en sorte de vérifier tous les scénarios possibles afin de repérer d'éventuelles erreurs dans notre code et les prévenir. Nous avons effectué une batterie de tests complets pour s'assurer que chaque fonctionnalité fonctionne correctement et que toutes les conditions sont prises en compte comme il se doit. Cela implique d'effectuer des tests avec différentes valeurs d'entrée, des situations limites, des valeurs extrêmes et des scénarios spécifiques qui pourraient mettre à l'épreuve la solidité de notre code. En menant ces tests approfondis, notre objectif était de garantir la fiabilité et la stabilité de notre application, en nous assurant qu'elle se comporte conformément à nos attentes et aux exigences du projet.