GROUPE E3 : Fourmaintraux Camille – Desrumeaux Julien

Rapport pour la Saé 2.01/2.02 partie Programmation Orientée Objet

Lien vers le répertoire GitLab : https://gitlab.univ-lille.fr/sae2.01-2.02/2023/E3/

Démarrer l'application :
Une fois le fichier décompressé, pour démarrer l'application, vous devez ouvrir un terminal vous rendre dans le répertoire « Application » afin d'entrer cette commande :
→ ***java -jar E3.jar***

Utiliser l'application :
Les répertoires CSV et historique jouent un rôle crucial dans le fonctionnement de l'application, permettant l'importation et l'exportation de données. Pour pouvoir utiliser un fichier CSV, il faut obligatoirement qu'il soit placer dans ce répertoire CSV.
Lire le readme pour plus de détails.

Analyse Technique

L'implémentation des fonctionnalités demandées à partir des classes fournies a été réalisée avec une approche orientée objet cohérente. Nous avons réussi à intégrer les fonctionnalités requises en utilisant les classes existantes et en en créant d'autres au besoin, telles que Sejour et History pour la gestion de l'historique. Cependant, de nombreuses améliorations pourraient être apportées à l'implémentation. Tout d'abord, la gestion des erreurs et des exceptions pourrait être renforcée. Celle-ci est encore très maladroite et n'est pas entièrement bien exploitée. De nombreuses parties du code ne sont pas protégées et une erreur à ces endroits pourrait faire planter l'application. Améliorer la gestion d'erreur améliorerait la clarté, car les messages d'erreur seraient plus précis, et cela renforcerait la robustesse globale de l'application en diminuant les risques de plantage. De plus, l'implémentation pourrait bénéficier d'une meilleure modularité et d'une plus grande réutilisabilité du code. Les classes fournies couvrent globalement les fonctionnalités requises, mais on retrouve des bouts de code très similaires éparpillés dans le code. Il faudrait identifier les parties du code qui pourraient être réutilisées dans d'autres contextes et les isoler dans des classes distinctes. Cela facilitera les mises à jour et permettra une plus grande flexibilité dans l'ajout des nouvelles fonctionnalités. Enfin, nous nous sommes rendu compte d'erreurs et de maladresses dans les parties fondamentales du code, mais seulement après les avoir développées, et donc bien trop tard. Avant que le projet ne démarre, il est difficile d'avoir une vision globale de ce à quoi doit ressembler le résultat. Maintenant que le projet est terminé, on se rend compte de nombreux problèmes et de comment on aurait pu faire mieux. Il y a aussi de nombreux morceaux de code qui sont devenus inutiles, mais qui restent présents dans le projet, car nous avons changé notre manière de faire entre-temps, mais nous ne sommes plus sûrs de ce qui est essentiel ou non. Nous avons aussi mal compris l'historique, car nous l'avons commencé à coder en prenant en compte plusieurs années, alors qu'il était clairement écrit dans le sujet que l'historique ne se basait que sur une. L'historique est donc un hybride étrange entre un historique qui prend plusieurs années et un qui n'en prend qu'une. Il y a aussi beaucoup de fonctions qui ne sont pas du tout optimisées, notamment les fonctions weight qui ont tendance à recalculer une valeur à chaque appel, plutôt que de stocker cette valeur quelque part et de la récupérer ensuite dans la fonction.

Analyse des Tests

Au niveau quantitatif, nous avons testé toute les classes, mais nous n'avons pas testé toute les méthodes. Ces test s'assurent que la classe fonctionnent correctement mais ne vérifie pas forcément chaque méthodes, surtout dans les plus grandes fonctions, ce n'est pas le cas dans les plus petite classe où toute les fonctions sont testées même les méthodes d'une ligne qui ne mérite pas forcément de tests. Cela aurait dû donc être l'inverse, où les classes les plus grande et les plus complexe, celle qui sont les plus susceptibles de causer des erreurs soit les plus testés. Au final, pour la partie quantitative, le minimum est fait, mais c'est clairement grandement améliorable.
Au niveau de la partie qualitative, comme dit précédemment, toutes les fonctions des plus grandes classes ne sont pas testés, mais les méthodes testés sont les principales, celles qui utile toute les autres. Si ces méthodes fonctionnent alors tout le reste des méthodes devraient fonctionner correctement, pour autant cela manque de rigueur car ce n'est pas optimal, il y devrait y avoir au mieux qu'une méthode testé par test. En créant des tests testant plusieurs méthodes à la fois, si une erreur apparaît il est alors beaucoup plus difficile de déterminer quelle est la méthode défectueuse.

Diagramme UML et réflexion sur les mécanismes objets

Voir figure 1.

En plus de des classes de base à implémentés, nous avons ajouté que deux nouvelles classes : Sejour et History, qui permettent la gestion de l'historique. Ces classes ont beaucoup changé par rapport au moment où nous avons décider de les implémenter, et donc les évolutions les ont fait accumuler beaucoup de défaut, notamment History. Nous avons peu réutiliser la classe CriterionName, pendant un long moment Criterion n'utilisait même pas CriterionName mais un String pour son attribut valeur dû un une erreur d'inattention de notre part. Contrairement à tout ce que nous avons vus en cours, notre code en réutilise peu, il n'y a pas d'héritage, de classes abstraites ou d'interfaces. Cela nous laisse supposer que des bien meilleures implémentation qui exploite toutes les notions sont possibles. Enfin, nous avons beaucoup utiliser les ArrayList, car c'était l'objet que nous maitrisions le mieux, quitte à délaisser des structure de données qui aurait été probablement plus approprié.

Conclusion

L'implémentation actuelle des fonctionnalités demandées nécessite des améliorations en termes de gestion des erreurs, de modularité du code et de couverture des tests, de plus, l'application pourrait bénéficier d'une meilleure exploitation des concepts objets vus en cours. Le résultat final en est presque décevant, mais en tirerons des leçons de ces erreurs et de ces maladresses constatées pour améliorer nos futures projets.

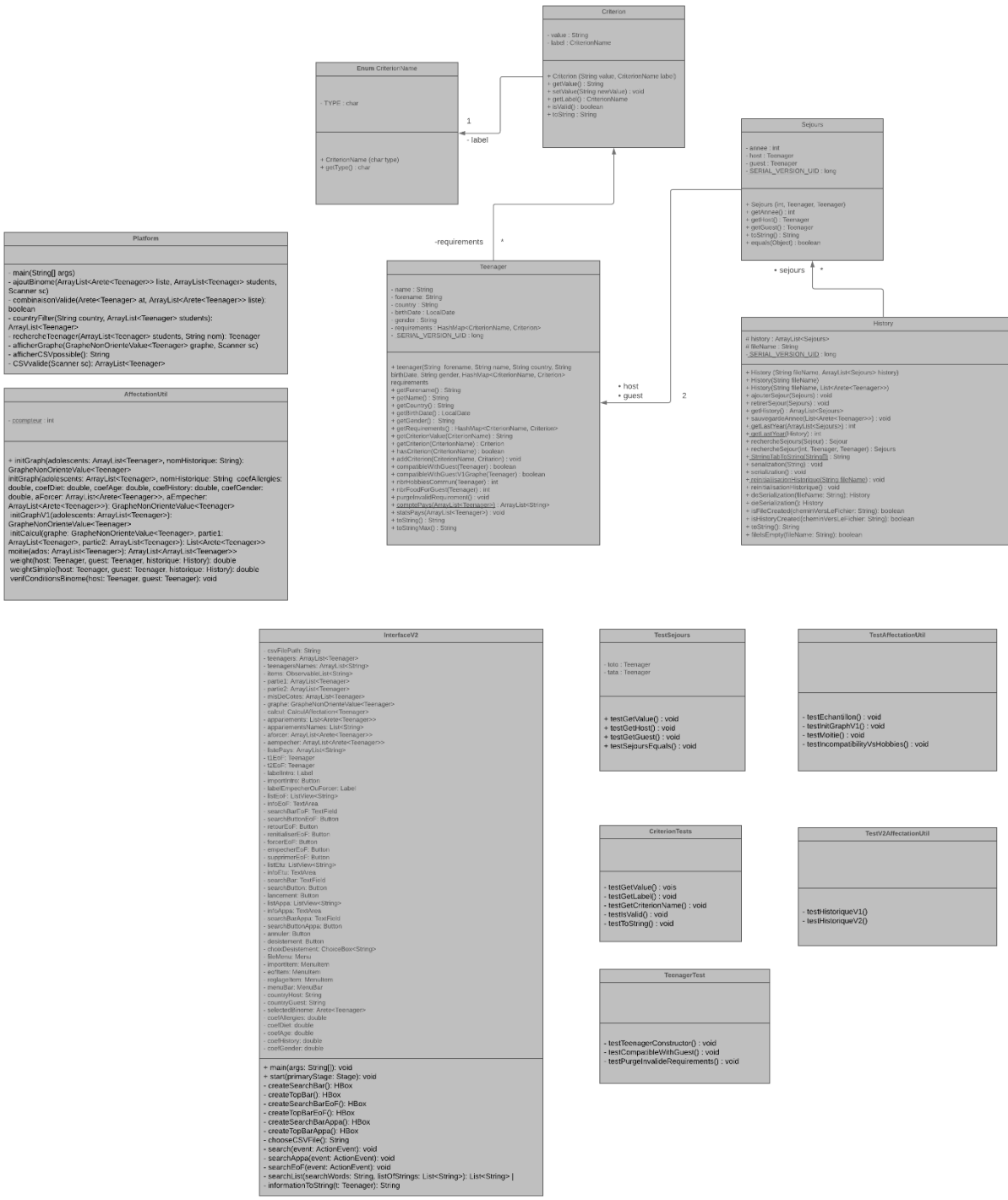

Figure 1 - Diagramme UML