

Rapport de Dév. SAE 2.01-2.02

☰ Groupe E-G1

Renan Declercq

Florian Etrillard

Felix Pereira

1. Table des matières

- 1. Table des matières
- 2. Guide de démarrage
 - 2.1. Fonctionnalités
 - 2.2. Utilisation de l'application/pré-requis
 - 2.2.1. Fichier de configuration générale
 - 2.2.2. Fichier des étudiants
 - 2.2.3. Fichiers des filtres d'approbation
 - 2.3. Lancement de l'application
- 3. Diagramme UML
- 4. Analyse de la conception objet
 - 4.1. Packages `input` et `output`
 - 4.2. Package `tutoring`
 - 4.2.1. Classes représentant les individus
 - 4.2.2. Système d'approbation
 - 4.2.3. Utilisation des exceptions
 - 4.3. Package `calculation`
 - 4.3.1. Affectations
 - 4.3.2. Calcul du score
 - 4.3.3. Mise à profit des méthodes `hashCode` et `equals`
 - 4.4. Récapitulatif
- 5. Qualité de développement
 - 5.1. Testing
 - 5.2. Qualité du code
 - 5.3. Utilisation de Git

- 6. Bilan et réflexion individuelle
 - 6.1. Contributions personnelles
 - 6.1.1. Renan Declercq
 - 6.1.2. Florian Etrillard
 - 6.1.3. Felix Pereira
 - 6.2. Retours d'expérience
 - 6.2.1. Renan Declercq
 - 6.2.2. Florian Etrillard
 - 6.2.3. Felix Pereira
 - 6.2.3.1. Travail d'équipe - Apprentissage
 - 6.2.3.2. Participation - Détail des problèmes retenus
 - 6.2.3.3. Compréhension
 - 6.2.3.4. Solution

2. Guide de démarrage

2.1. Fonctionnalités

Le programme permet de calculer et d'obtenir des affectations d'élèves tuteur et tutoré par matière dans des fichiers JSON (un fichier JSON par matière). Tous les paramétrages et toutes les fonctionnalités sont générés à partir de fichiers JSON.

Liste exhaustive des fonctionnalités :

- Importer des étudiants via un fichier JSON
- Importer des matières et leur nombre d'affectations maximum via un fichier JSON
- Forcer affectations (à travers un fichier JSON)
- Interdire des affectations (à travers un fichier JSON)
- Calculer des affectations via l'algorithme d'affectations
- Intervenir sur l'importance des critères d'affectations via un fichier JSON
- Filtrer les étudiants pré-affectations grâce à des critères spécifiques via un fichier JSON

Toutes ces fonctionnalités seront expliquées plus en détails ci-dessous.

2.2. Utilisation de l'application/pré-requis

Il faut préparer des *fichiers JSON* qui seront importés lors de l'exécution du programme, ils sont tous situés dans le dossier `res`.

2.2.1. Fichier de configuration générale

Un fichier `config.json` est déjà disponible pour tester le programme, cependant il est possible de modifier ce fichier de configurations :

- Vous pouvez y spécifier les coefficients (= importance) des différents critères pris en compte dans le calcul d'affectation : Veuillez ne changer que la valeur, vous ne pouvez pas ajouter de critères.
- Vous pouvez aussi y spécifier toutes les matières disponibles en tutorat en clé, ainsi que le nombre de places maximum (nombre maximum d'affectations) dans les valeurs.

Prenez exemple sur ce fichier JSON ci-dessous :

```

{
  "coefficients": {
    "OVERALL_GRADE": {
      "coefficient": 3
    },
    "RESSOURCE_OVERALL_GRADE": {
      "coefficient": 4
    },
    "TUTOR_YEAR": {
      "coefficient": 4
    },
    "TUTOR_MOTIVATION": {
      "coefficient": 1
    },
    "STUDENT_IN_DIFFICULTY_MOTIVATION": {
      "coefficient": 1
    }
  },
  "ressources": {
    "NOM_MATIERE_1": 15,
    "NOM_MATIERE_2": 20
  }
}

```



Signification des nombres

3 , 4 , 4 , 1 , 1 représentent la valeur du coefficient, c'est-à-dire l'importance donnée au critère : plus la valeur est haute, plus le critère aura d'importance dans le calcul d'affectation. Dans la partie coefficients, il n'y a que ces valeurs que vous pouvez changer.

15 , 20 représentent le nombre d'affectations maximum pour la matière, vous pouvez ajouter autant de matière que vous le souhaitez (a la place de NOM_MATIERE).

2.2.2. Fichier des étudiants

Un fichier `students.json` est déjà disponible, nous avons généré aléatoirement quelques étudiants tuteur et étudiants en difficulté, mais dans un cas réel, ce serait à l'enseignant d'entrer chaque candidat comme le modèle ci-dessous

```

{
  "students_tutor": [
    {
      "first": "first_name_1",
      "last": "last_name_2",
      "enrollment": "NOM_MATIERE",
      "moy_ressource": 17.15,
      "moy_generale": 13.61,
      "motivation": "MEDIUM_MOTIVATION",
      "nb_absences": 1,
      "year": 2
    },
    {
      "first": "first_name_2",
      "last": "last_name_2",
      "enrollment": "NOM_MATIERE",
      "moy_ressource": 18.46,
      "moy_generale": 15.14,
      "motivation": "HIGH_MOTIVATION",
      "nb_absences": 3,
      "year": 3
    }
  ],
  "students_in_difficulty": [
    {
      "first": "first_name_3",
      "last": "last_name_3",
      "enrollments": [
        "NOM_MATIERE_1",
        "NOM_MATIERE_2"
      ],
      "moy_generale": 10.25,
      "grades": {
        "NOM_MATIERE_1": 10.57,
        "NOM_MATIERE_2": 7.87
      },
      "motivation": "LOW_MOTIVATION",
      "nb_absences": 8
    },
    {
      "first": "first_name_4",
      "last": "last_name_4",
      "enrollments": [
        "NOM_MATIERE"
      ],
      "moy_generale": 7.5,
      "grades": {
        "NOM_MATIERE": 6.59
      },
      "motivation": "MEDIUM_MOTIVATION",
    }
  ]
}

```

```
    "nb_absences": 5
  }
]
```



Motivations disponibles

LOW_MOTIVATION

MEDIUM_MOTIVATION

HIGH_MOTIVATION

2.2.3. Fichiers des filtres d'approbation

Pour chaque matière disponible en tutorat, vous pouvez configurer les filtres d'approbation. Pour cela, il vous faut créer un dossier `<NOM_MATIERE>` (toujours dans `res`) et ajouter un fichier JSON dans ce dossier `<NOM_MATIERE>_filters.json` . Si vous ne voulez pas créer ce dossier, vous pouvez exécuter une première fois l'application et il le fera pour vous.

Par exemple, si vous voulez créer des filtres pour les matières POO et IHM, il faudra créer ces deux fichiers :

- `res/POO/POO_filters.json`
- `res/IHM/IHM_filters.json`

Vous pouvez suivre l'exemple ci-dessous, il n'y a pas de minimum de filtres requis, vous pouvez très bien ne pas définir de filtres du tout dans **reject** et/ou **approve**.

Par exemple si vous ne spécifiez aucun filtre pour les étudiants tuteurs ils seront tous approuvés par défaut et seront pris en compte dans le calcul d'affectation, cependant cela ne veut pas forcément dire qu'ils seront tous automatiquement affectés, l'algorithme d'affectation en décidera !

```
{
  "students_tutor": {
    "reject": {
      "OVERALL_GRADE_INFERIOR_TO": 13.5,
      "NB_ABSENCES_SUPERIOR_TO": 10
    },
    "approve": {
      "RESSOURCE_OVERALL_GRADE_SUPERIOR_TO": 17
    }
  },
  "students_in_difficulty": {
    "reject": {
      "OVERALL_GRADE_SUPERIOR_TO": 13
    },
    "approve": {
      "RESSOURCE_OVERALL_GRADE_INFERIOR_TO": 7.5,
      "NB_ABSENCES_INFERIOR_TO": 1
    }
  }
}
```

Filtres disponibles

RESSOURCE_OVERALL_GRADE_INFERIOR_TO
 RESSOURCE_OVERALL_GRADE_SUPERIOR_TO
 OVERALL_GRADE_INFERIOR_TO
 OVERALL_GRADE_SUPERIOR_TO
 NB_ABSENCES_SUPERIOR_TO
 NB_ABSENCES_INFERIOR_TO

Filtres par défaut

Vous n'êtes pas obligé de configurer les filtres pour chaque matière, il existe un fichier (res/DEFAULT_filters.json) contenant les filtres par défaut. Si vous n'avez pas créé de fichier filtres pour une matière existante, ce fichier sera pris à la place. Vous pouvez tout à fait modifier les filtres par défaut. Vous pouvez aussi vous contenter des filtres par défaut pour la majorité ou même toutes vos matières.

2.3. Lancement de l'application

Pour lancer l'application : Sur un terminal, allez dans le répertoire qui contient l'archive jar et entrez la commande suivante : `java -jar ./E-G1.jar`

Cela exécutera automatiquement notre classe `main` nommée `UseAssignmentPlatform.java`.

Entrez les matières dans lesquelles vous intervenez (en tant qu'enseignant) une à une, puis entrez "OK" lorsque vous avez fini d'énumérer toutes vos matières.



Il est nécessaire de respecter vos conventions !

C'est-à-dire qu'il faut entrer les matières comme elles sont écrites dans les fichiers JSON importés (étudiants, filtres et config).

Par exemple, si vous décidez de nommer une matière POO, il faut que cette matière soit nommé de cette manière dans tous les fichiers d'import utilisés, ainsi que lorsque vous l'entrez dans le terminal au lancement du programme pour spécifier vos matières.

Vos affectations sont automatiquement générées dans des fichiers JSON, chaque matière aura son propre dossier (dans le dossier `res`) ainsi que son fichier JSON à l'intérieur du dossier de la matière. Les dossiers et les fichiers seront automatiquement créé s'il n'existe pas, s'ils existent déjà, le fichier `<NOM_MATIERE>_assignments.json` sera chargé avant l'affectation et mis à jour après l'affectation. Les affectations déjà existantes sont sauvegardés et ne changent pas après une nouvelle affectation sauf si vous changez vous-même le fichier d'affectations.

Les affectations forcées et interdites :

Vous pouvez aussi désigner des affectations forcées et interdites au préalable ou a posteriori. Les fichiers d'affectations se présentent comme ci-dessous :


```
{
  "computed": [
    {
      "score": 2.740234801437361,
      "student_tutor": "Hermione.Granger",
      "student_in_difficulty": "Drago.Malfoy"
    }
  ],
  "forced": [
    {
      "student_tutor": "Harry.Potter",
      "student_in_difficulty": "Ron.Weasley"
    }
  ],
  "forbidden": [
    {
      "student_tutor": "Harry.Potter",
      "student_in_difficulty": "Drago.Malfoy"
    }
  ]
}
```



Affectations interdites

Si vous spécifiez une affectation interdite entre deux élèves qui ont été affecté par calcul au préalable, leur affectation calculée sera enlevée au prochain lancement de l'algorithme d'affectations.



Affectations forcées

Les affectations forcées ajoutées a posteriori ne changent pas les affectations calculées même si l'on relance l'algorithme, c'est-à-dire que le tuteur garde toutes ses affectations calculées en plus de son affectation forcée et il en est de même pour le tutoré (qui peut alors avoir plusieurs tuteurs). Il est toujours possible d'enlever l'affectation calculée directement dans le fichier JSON ou de spécifier une affectation interdite pour retirer les affectations calculées au préalable.



D'autres exemples

Un fichier `res/EXAMPLE_assignments.json` est disponible pour voir d'autres exemples d'affectations



3. Diagramme UML

Par soucis de lisibilité, le diagramme UML est disponible en formats `svg` et `png` dans le dossier `UML` joint avec rapport.

PlantUML

Pour réaliser ce diagramme, nous avons utilisé l'outil `PlantUML`. Les flèches d'association sont mal représentées et apparaissent comme des flèches pleines, mais nous n'avons pas trouvé comme corriger cela. De plus, nous ne pouvons pas contrôler le positionnement sur le diagramme, mais cela reste globalement plus pratique d'utiliser cet outil pour le réaliser.

Info

Dans le diagramme, nous avons omis nos classes d'exceptions pour ne pas brouiller la lecture de ce dernier.

4. Analyse de la conception objet

La classe nommée `AssignmentPlatform` est la classe principale de l'application, il s'agit de la cheffe d'orchestre de l'application. C'est cette classe qui nous permet d'agir sur les autres, elle permet de créer un niveau d'*abstraction* et une *encapsulation* sur les autres classes. Elle contient donc peu de logique et peut être présentée comme une *API/interface* (au sens général du terme) avec les autres classes. Cela nous permet de cacher toute la logique de l'application de la future partie IHM.

Décomposition en packages

Notre projet est décomposé en 4 packages de manière à séparer les classes en fonction de leur utilité.

4.1. Packages `input` et `output`

Tout d'abord, le package `input` regroupe toutes les classes permettant la lecture des différents fichiers de configuration, de données, etc.

Nous avons séparé la lecture de chaque fichier différent dans les classes suivantes

`AssignmentsJSONReader`, `ConfigJSONReader`, `FiltersJSONReader` et `StudentsJSONReader`. Elles permettent respectivement la lecture des affectations calculées ou définies par l'utilisateur, de la configuration (coefficients, matières et places disponibles), des filtres d'approbation et des étudiants.

Elles héritent toutes de la classe `JSONReader`, déclarée *abstraite*, car elle ne contient pas de "*code concret*", mais seulement des méthodes communes aux 4 classes de lecture. Ce principe nous permet de rajouter de nouvelles classes de lecture de fichiers sans modifier celles existantes, tout en respectant le principe de *responsabilité unique*.

Lecture de CSV

Nous avons codé la fonctionnalité de lecture des étudiants avec CSV (`StudentsCSVReader`), avant d'apprendre qu'il était mieux (et plus pratique) de le faire au format JSON, mais nous l'avons laissée au cas où.

Ensuite, le package `output` regroupe les classes permettant l'écriture des différents fichiers de configuration, de données etc. Nous avons appliqué le même principe d'héritage qu'avec la lecture de

fichiers JSON pour les mêmes raisons.

Pour le moment, il n'y a qu'une seule classe d'écriture, car il n'était pas vraiment nécessaire d'en ajouter d'autres, mais il sera toujours possible d'en rajouter, si nécessaire, grâce à la bonne modularité du code.

Pour finir, nous avons choisi d'écrire la logique pour convertir les objets en JSON et inversement, en dehors des classes des objets, eux même, pour ne pas les surcharger et isoler la logique de lecture/écriture.

4.2. Package `tutoring`

Ce package contient toutes les classes spécifiques au tutorat, autrement dit toutes les classes qui sont spécifiques au contexte du problème donné.

La classe `Ressource` encapsule les étudiants en difficulté et les tuteurs concernés par cette ressource.

4.2.1. Classes représentant les individus

Pour répondre à la problématique posée, il nous est naturellement venu l'idée de créer différentes classes pour représenter les étudiants tuteurs (`StudentTutor`), les étudiants en difficulté (`StudentInDifficulty`) et les professeurs (`Teacher`).

Afin de factoriser le code, nous avons créé la classe *abstraite* `Student` qui contient les méthodes communes aux 2 classes d'étudiant, il s'agit une *classe abstraite*, car elle n'a pas pour but d'être instanciée et permet d'éviter la duplication de code.

Toujours dans le but de factoriser le code, nous avons créé une classe *abstraite* `Person` dont les classes `Student` et `Teacher` dérivent.

Pour finir, chaque étudiant possédant une certaine motivation, nous avons créé une *énumération* (`MotivationScale`) regroupant les différents degrés de motivation. Les énumérations étant très adaptées pour stocker *"un ensemble fini et constant de valeurs"*.

4.2.2. Système d'approbation

Le système d'approbation permet d'approuver ou non les étudiants en fonction de filtres définis par l'utilisateur. Un filtre est décrit par la classe `CriteriaFilter` dont les attributs sont déclarés `final` car au cours *"du cycle de vie de l'objet"*, ils n'ont pas pour vocation de changer. Ces attributs sont le type du critère, sa valeur limite et si c'est un filtre d'approbation ou de refus.

Les types de critère des filtres sont définis par une *énumération* (`FilterCriteriaType`) pour les mêmes raisons évoquées que pour l'énumération des différents degrés de motivation.

Les classes implémentant ce mécanisme de filtre par critère doivent implémenter l'interface `ICriteriaFilterable` .

4.2.3. Utilisation des exceptions

Les classes d'exception peuvent s'avérer très utiles, car nous pouvons profiter du mécanisme de gestion des erreurs pour éviter de faire une multitude de tests à l'intérieur des fonctions. De plus nous pouvons effectuer un traitement spécifique au déclenchement de l'erreur sans faire planter tout le programme. Cela permet aussi d'empêcher certaines erreurs de se produire : par exemple, si l'utilisateur ajoute un étudiant qui n'existe pas dans une ressource, l'erreur permet de l'empêcher et d'éviter d'obtenir une erreur "plus grave" (par exemple, une exception de type `NullPointerException`).

- `NotAllowedException` : Cette exception se déclenche lorsque l'enseignant connecté essaye d'intervenir sur une matière dans laquelle il n'est pas concerné. Nous avons jugé nécessaire de limiter l'utilisateur uniquement aux matières dans lesquels il est référent.
- `RessourceNotFoundException` : Cette exception se déclenche lorsqu'on essaye d'obtenir une ressource et qu'elle n'est pas trouvée. Nous avons jugé important d'inclure cette exception, car cela peut être une erreur assez fréquente, lorsque la ressource n'a pas été ajoutée au programme ou qu'il y a une erreur dans le nom de la ressource (dans les fichiers importés par exemple)
- `StudentNotFoundException` : Cette exception se déclenche lorsqu'on essaye d'obtenir un étudiant qui n'est pas trouvé. Cette erreur peut aussi être assez fréquente, elle peut se produire quand un étudiant n'existe tout simplement pas ou lorsqu'il n'est pas trouvé dans la ressource concernée.
- `IncorrectFormatException` : Lorsqu'une erreur dans le "*parsing*" du fichier à lieu, nous renvoyons notre propre exception afin de pouvoir par la suite adresser un message d'erreur à l'utilisateur.

4.3. Package `calculation`

Ce package contient toutes les classes permettant de réaliser les calculs d'affectations, de score, etc.

Nous avons décidé de les rendre plus dépendantes des classes concernant le tutorat en lui-même, en effet elles peuvent être utilisées plus génériquement et éviter un trop fort *couplage* entre ces classes permet une meilleure maintenabilité du code. Afin de faire cela, nous avons utilisé les *interfaces* et la *généricité*.

4.3.1. Affectations

Pour commencer, nous avons une *interface* `IAssignable` représentant le fait qu'une entité peut être affecté. Il s'agit de l'interface que les classes qui veulent utiliser le calcul d'affectations doivent implémenter (comme notre classe `Student` par exemple).

Deux autres classes implémentent cette *interface*:

- `FictiveAssignable` qui est une classe permettant de "*combler les trous*" dans le graphe d'affectations
- `UniqueAssignableWrapper` qui est une classe encapsulant elle-même un objet implémentant l'*interface* `IAssignable`. Elle nous permet d'ajouter plusieurs fois un même *objet assignable* dans le graphe d'affectations, grâce à l'encapsulation (et la redéfinition des méthodes `hashCode` et `equals`), le graphe et notre classe `AssignmentCalculator` n'ont pas besoin de se préoccuper de la manière dont est implémentée `IAssignable`.

La classe `AssignmentCalculator` s'occupe de créer le graphe et le remplir avec les *assignables* (dont les fictifs et doublons si nécessaires) Les *assignables* se présentent sous forme de deux listes, une pour la partie gauche du graphe et une autre pour la partie droite.

Puis, elle se charge de renvoyer les affectations sous la forme d'une liste d' `Assignment` (composé d'un élément gauche, le tuteur en l'occurrence et d'un élément droit, l'étudiant en difficulté).

Il nous a été nécessaire d'utiliser des *types génériques* pour pouvoir passer n'importe quels *types* (implémentant l'interface `IAssignable`) à notre classe `AssignmentCalculator`, tout en récupérant des objets `Assignment` avec les mêmes *types* à la sortie.

4.3.2. Calcul du score

Afin de maintenir une bonne décomposition du code, nous avons décidé de séparer le calcul du score des objets concernés et du calcul d'affectations (`AssignmentCalculator`).

Pour ce faire, nous avons créé une classe `ScoreCalculator` qui accepte deux objets implémentant l'interface `ICriteriaRatable`, ils représentent les deux entités pour lesquelles nous voulons calculer un score commun, en l'occurrence, il s'agit du `StudentTutor` et du `StudentInDifficulty` (même si nous avons utilisé une interface afin de réduire le couplage entre les classes). L'utilisation de cette interface nous permet d'utiliser la notion de polymorphisme et ainsi d'isoler la logique de calcul en laissant la classe qui implémente cette interface renvoyer seulement les valeurs pour les critères qu'elle souhaite utiliser pour calculer son score (définis dans l'énumération `ScoreCriteria`).

Afin de nous assurer qu'un objet *assignable* possède bien la possibilité d'être *noté*, nous avons fait dériver l'interface `IAssignable` de l'interface `ICriteriaRatable`.

4.3.3. Mise à profit des méthodes `hashCode` et `equals`

Dans la classe `Assignment` (comme pour d'autres classes), nous avons défini stratégiquement nos méthodes `hashCode` et `equals` afin de prendre en compte seulement les attributs contenant la partie droite/gauche de l'affectation. Ainsi, en ajoutant nos `Assignments` dans un `Set`, il est impossible d'avoir à la fois une affectation forcée et une affectation interdite. Cette redéfinition réfléchie des deux méthodes `hashCode` et `equals` permet de se libérer d'une implémentation "manuelle" de ce même mécanisme.

4.4. Récapitulatif

Pour résumer, nous avons essayé au maximum dans tout le projet d'appliquer le principe de *responsabilité unique*, chaque classe/méthode à un seul objectif. Nous avons également fait en sorte d'éviter au maximum la répétition/duplication de code (voir [l'analyse](#)) en appliquant le principe *DRY* (*Don't Repeat Yourself*). Pour cela, nous avons utilisé les concepts objets comme l'héritage, les interfaces...

Nous avons également utilisé les mécanismes objets comme la visibilité des attributs, l'encapsulation, les packages, afin de créer des niveaux d'abstraction sur nos classes, de rendre le code plus lisible et utilisable plus facilement sans avoir à connaître la logique interne de chaque classe.

5. Qualité de développement

5.1. Testing

Nous avons 2 classes de test :

- `AssignmentAlgoTest` : Cette classe de test a été implémenté pour la partie *graphe*, elle permet de tester les affectations obtenues selon diverses configurations
- `AssignmentProgramTest` : Cette classe de test a été implémenté pour la partie *POO*, elle est décomposée en 12 parties :
 - `getStudent` : Test sur les méthodes qui recherchent un étudiant par son nom
 - `getStudentNotFound` : Test sur l'exception `StudentNotFound` lorsque l'on recherche un étudiant qui n'a pas été ajouté
 - `addStudent` : Test sur les ajouts d'étudiants
 - `removeStudents` : Test sur les suppressions d'étudiants
 - `setConnectedTeacher` : Test sur le changement de l'enseignant connecté (fonctionnalité pour IHM)
 - `handleCriteriaCoefficients` : Test sur les mises à jour et les réinitialisations de l'importance (coefficients) des différents critères pris en compte dans le calcul d'affectation
 - `forcedAssignments` : Test des affectations forcées
 - `forbiddenAssignments` : Test des affectations interdites
 - `removeAssignment` : Test sur la suppression d'une affectation (qu'elle soit calculée, forcée ou interdite)
 - `resetAssignments` : Test sur la réinitialisation des affectations selon leur type (calculée, forcée ou interdite)
 - `setStudentsApprovalStatusByCriteria` : Test sur filtres d'approbation
 - `errorException` : Test des différentes exceptions implémentées

Choix des tests

Nous avons privilégié les tests pour les packages `assignment`, `calculation` et `tutoring`. Nous n'avons pas fait de test concernant les `input` et les `output` car nous n'avons pas eu le temps et nous voulions tester la logique de notre application en priorité. Bien évidemment, nous n'avons pas testé le package `examples` contenant nos différentes classes de lancement. Nous avons donc 50% de couverture de code sur l'ensemble du projet.

MEASURES

New Code

Since May 7, 2022
Started 1 month ago

Overall Code

0

Bugs

Reliability

A

0

Vulnerabilities

Security

A

0

Security Hotspots

Reviewed

Security Review

A

3h 10min

Debt

8

Code Smells

Maintainability

A



50.0%

Coverage on 1.2k Lines to cover

-

Unit Tests



0.0%

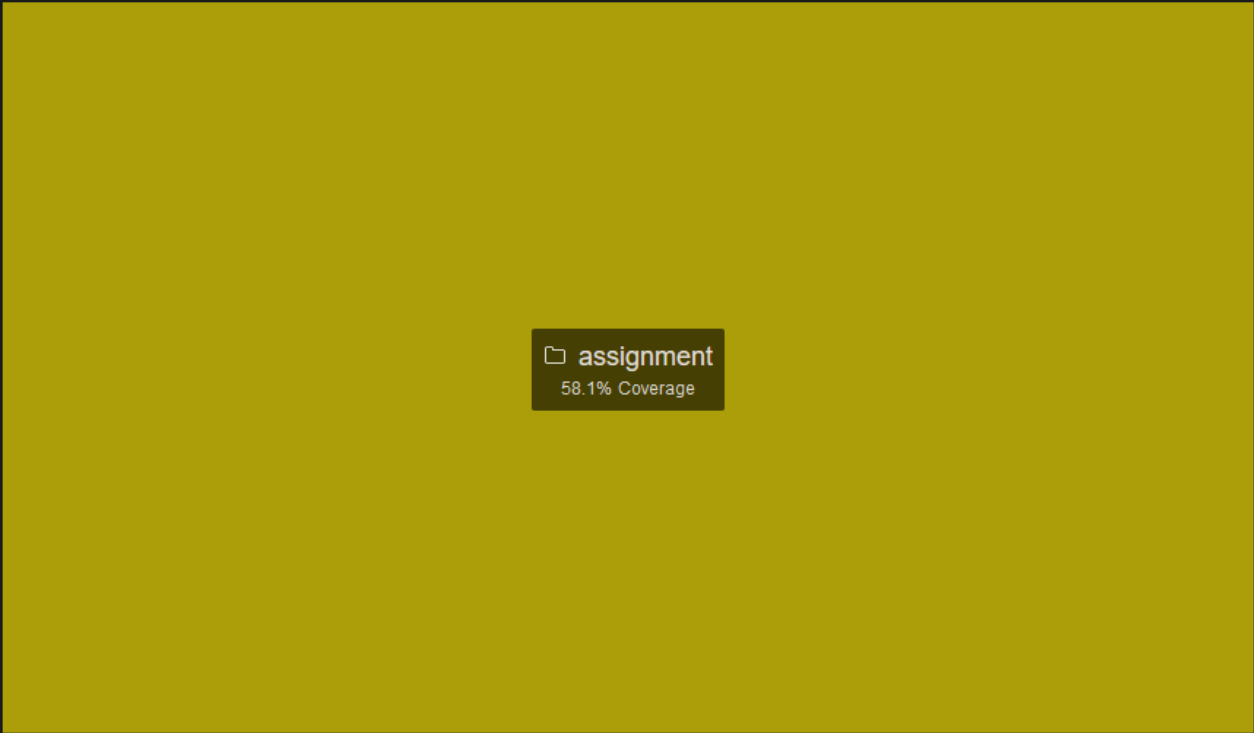
Duplications on 1.7k Lines

0

Duplicated Blocks

Coverage 50.0%

Color: Coverage Size: Lines of Code



assignment

58.1% Coverage

examples

0.0% Coverage

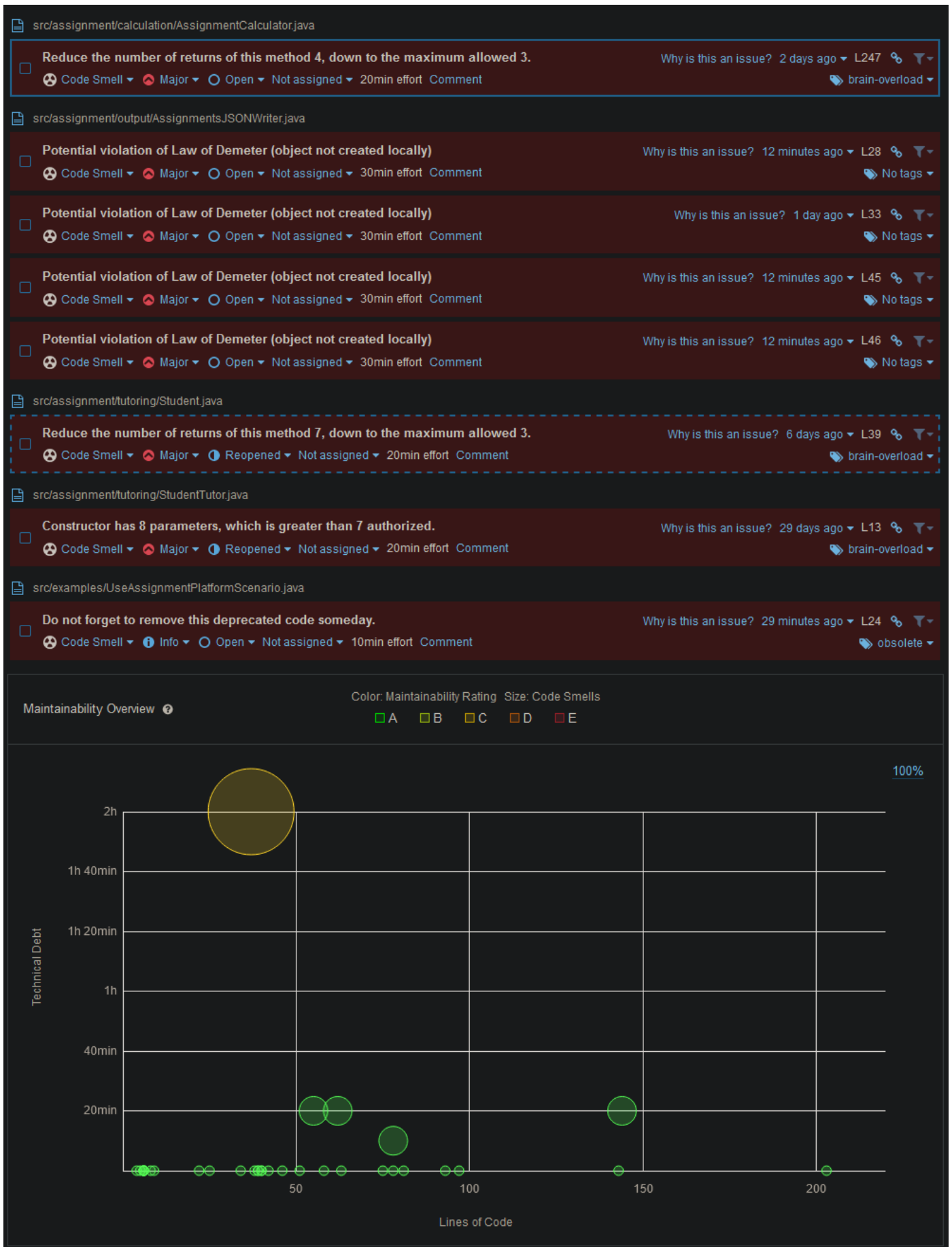


Info

Pour calculer la couverture de code nous avons dû utiliser l'utilitaire `jacoco` compatible avec SonarQube

5.2. Qualité du code

- Rédaction du code en anglais : c'est une bonne pratique, car c'est la langue internationale (et c'est plus cohérent/propres).
- Nous avons utilisé `SonarQube` pour mesurer la qualité de notre code avec les règles intégrées de bases, nous avons aussi pu rajouter la plupart des règles de `PMD` comme la *Loi de Demeter* par exemple (grâce à une extension de `SonarQube`).
- Nous avons investi beaucoup de temps pour régler le maximum de *code smells*, cependant certains n'ont pas été réglés par manque de temps et d'autres, car nous ne savions pas comment les régler, comme avec notre constructeur qui prend 8 paramètres dans la classe `StudentTutor`. Le gros cercle jaune représente les *code smells* de la classe `assignmentJSONWriter` : 4 violations de la *Loi de Demeter* que nous n'avons pas eu le temps de régler.



- Rédaction de la documentation JAVA dans les classes que nous avons jugées être les plus pertinentes :

- `ScoreCalculator` et `AssignmentCalculator` car ce sont deux classes qui dépendent beaucoup de la résolution du problème, les commentaires peuvent aider à la compréhension des différentes méthodes.
- `AssignmentPlatform` car c'est la classe maîtresse de notre projet, cette classe permet l'abstraction de toutes nos autres classes. Nous passons exclusivement par les méthodes de cette classe dans notre programme principal. C'est donc la classe la plus importante à comprendre dans notre projet.


Comments (%) 13.6%	
calculation	18.2%
input	0.0%
output	0.0%
tutoring	1.1%
AssignmentPlatform.java	38.7%

Info

La documentation JAVA est disponible dans `/doc/index.html` .
 Nous avons créé la *javadoc* grâce à la commande ci-dessous :

```
javadoc -classpath ".\lib\sae2_02.jar;.\lib\json.jar" -sourcepath .\src\ assignment -d Doc
```

- Nous n'avons aucune duplication de code

Duplications DUPLICATED LINES (%)	
Overview 	
On new code	
Density	0.0%
Duplicated Lines	0
Duplicated Blocks	0
Overall	
Density	0.0%
Duplicated Lines	0
Duplicated Blocks	0
Duplicated Files	0

5.3. Utilisation de Git

Plus de 200 commits : Un travail assez régulier et quantitatif du groupe sur ce projet

- Nous avons fait notre maximum pour faire des commits "atomiques" (même si vraiment pas toujours évident)
- Répartition du travail par fonctionnalité et thématique puis répartition du travail avec les autres SAE (car nous avons le même groupe pour toutes les SAE).
- Cependant, nous nous tenions très régulièrement au courant de nos avancées respectives + travail en groupe régulier (Autonomie + Travail à domicile en groupe via discord) -> Beaucoup de réflexion commune et de mini-réunions / concertations
- 17 branches (dont master) : Création de branches par thématique de travail
- Merges assez fréquents sur master pour la mise en commun de nos codes

6. Bilan et réflexion individuelle

6.1. Contributions personnelles

6.1.1. Renan Declercq

Les packages dans lesquels j'ai le plus travaillé sont le package `tutoring` (notamment avec les différentes *classes students et les filtres*) et le package `input` (avec *les classes concernant tous les fichiers importés JSON et CSV* même si on n'utilise pas les imports CSV, ils restent fonctionnels).

J'ai aussi beaucoup travaillé sur la classe `AssignmentPlatform`, sur les tests `AssignmentProgramTest` et sur les *main classes* comme `UseAssignmentPlatform` qui est le programme utilisateur en ligne de commande et `UseAssignmentPlatformScenario` qui est un scénario préparé pour avoir un aperçu d'un résultat d'affectations dans le terminal à travers un petit scénario.

6.1.2. Florian Etrillard

Personnellement, mon travail s'est surtout concentré sur la réalisation de la structure/architecture du programme dans sa globalité (avec l'UML) et sur une grande partie de la réalisation des classes située dans le package `calculation` et un peu moins dans le package `tutoring`.

Tout au long du projet, j'ai essayé de diriger mon équipe le plus possible et au maximum de mes compétences vers une approche *Clean Code*, *DRY* et une bonne décomposition du code. Lorsque nécessaire, je me suis également assuré de corriger des incohérences, problèmes dans le code.

Pour finir, je me suis occupé de la partie analyse de la qualité du code (maintenabilité, coverage...) avec SonarQube, car j'avais déjà une certaine connaissance de cet outil plus compliquée à mettre en place, cependant nous avons beaucoup communiqué pour que tout le monde puisse comprendre comment cela fonctionne.

6.1.3. Felix Pereira

Objectif	Détails
UML - Logique	Réorganisation de l'UML / Ajout(s) : Suppression(s) d'élément(s)
UML - Graphique	Changement graphique concernant l'UML et son rendu

Objectif	Détails
Classes	Participation à la réalisation de classes (Au travers de réalisation mise en ligne / Au travers de mises en commun)
IHM - Basse fidélité	Participation à la réalisation du séquentiel et de la maquette Basse fidélité (via l'outil <i>Excalidraw</i>)
IHM - Haute fidélité	Participation à la réalisation de la maquette et des séquences d'interactions (via l'outil <i>Figma</i>)

6.2. Retours d'expérience

6.2.1. Renan Declercq

J'ai beaucoup apprécié travailler sur ce projet, car j'estime avoir appris et progressé plus que jamais en programmation orientée objet et en qualité de développement, notamment en "propreté" du code et Git, car rien ne vaut la pratique via un projet de groupe pour comprendre Git et développer des automatismes (comme les conventions Git que j'ai appris au fur et à mesure du projet par exemple).

Cependant, le projet était loin d'être simple et nous avons dû énormément travaillé (aussi à cause de toutes les SAE qui se chevauchent, c'est toujours difficile de travailler sur beaucoup de projet en même temps.). Pour ma part, j'ai rarement autant travaillé en termes de temps et d'investissement que ces deux derniers mois avec les différentes SAE dont celle-ci.

J'ai aussi beaucoup apprécié travailler avec mon groupe, je pense qu'ils me tiraient vers le haut et c'est aussi grâce à mes camarades de groupe que j'ai pu autant apprendre notamment sur la qualité de développement.

6.2.2. Florian Etrillard

J'ai adoré cette SAé, car elle m'a fait beaucoup progressé, j'ai appris et compris de très nombreux concepts (Git, UML, utilisation des interfaces en POO etc.). C'était très enrichissant, même si vraiment compliqué surtout à cause de la quantité de travail que nous avons à côté et en même temps, mais je pense qu'on est tous fiers du résultat final.

J'ai beaucoup apprécié travailler avec mon équipe ! Même avec le partage des tâches que nous avons effectué, nous avons maintenu une bonne communication au sein du groupe, pour veiller en permanence à ce que tout le monde ait bien compris ce qui a été fait, les concepts utilisés, etc. Excellent climat d'apprentissage et d'échange d'idées ! Personnellement, j'ai beaucoup développé mon sens du partage du savoir pendant cette SAé.

6.2.3. Felix Pereira

6.2.3.1. Travail d'équipe - Apprentissage

Le principal avantage que j'ai retenu à l'issue de cette SAE est le travail d'équipe entre moi et mes camarades. En effet, constater les différentes façons de raisonner de chaque membre en fonction d'un problème était intéressant dans la mesure où cela nous permet d'aborder différemment un problème que de la façon initiale à laquelle on aurait pu penser.

6.2.3.2. Participation - Détail des problèmes retenus

Le plus important dans cette partie est d'aborder d'un aspect critique ma participation au projet et mon utilité.

On peut tout d'abord noter une bien plus petite présence de ma part que de celles de mes camarades sur le *repository E-G1* en termes de commits. Il est arrivé que ma participation au projet ait été effectuée au travers de nos échanges durant les séances de travail en commun, ce qui résultait donc parfois en un `git push` commun pour l'ensemble du groupe sous le compte de l'un d'entre nous. Cependant, d'une manière générale, la différence de commits entre moi et mes camarades s'est créée de par la différence de temps personnel accordé à cette SAE. En effet, Florian et Renan sont parvenus tous deux à utiliser leur temps personnel d'après-cours en temps de travail pour la SAE, ce que je ne suis pas parvenu à faire. Mon absence à ces heures de travail, à eu comme résultat une différence de participation au projet de ma part et de cette différence, une distanciation du groupe.

6.2.3.3. Compréhension

Le problème principal retenu dans cette SAE est que mon manque d'investissement dans un contexte extérieur aux séances de l'université a créé un manque de participation au projet, ce qui n'était pas le cas dans le contexte des séances d'autonomies.

6.2.3.4. Solution

Une meilleure organisation personnelle et meilleure conciliation du temps personnel / productif est à envisager, car le problème que j'ai constaté dans cette SAE venait pleinement de moi-même.