Domaine Sciences et Technologies
Master Bio-informatique : DLAD

Aix*Marseille université
Socialement engagée

Année 2023-24

Programmation avancée : TP bonus 1
Code UE : SBICU05

Gestion des notes des étudiants

Gestion des notes des étudiants

Introduction

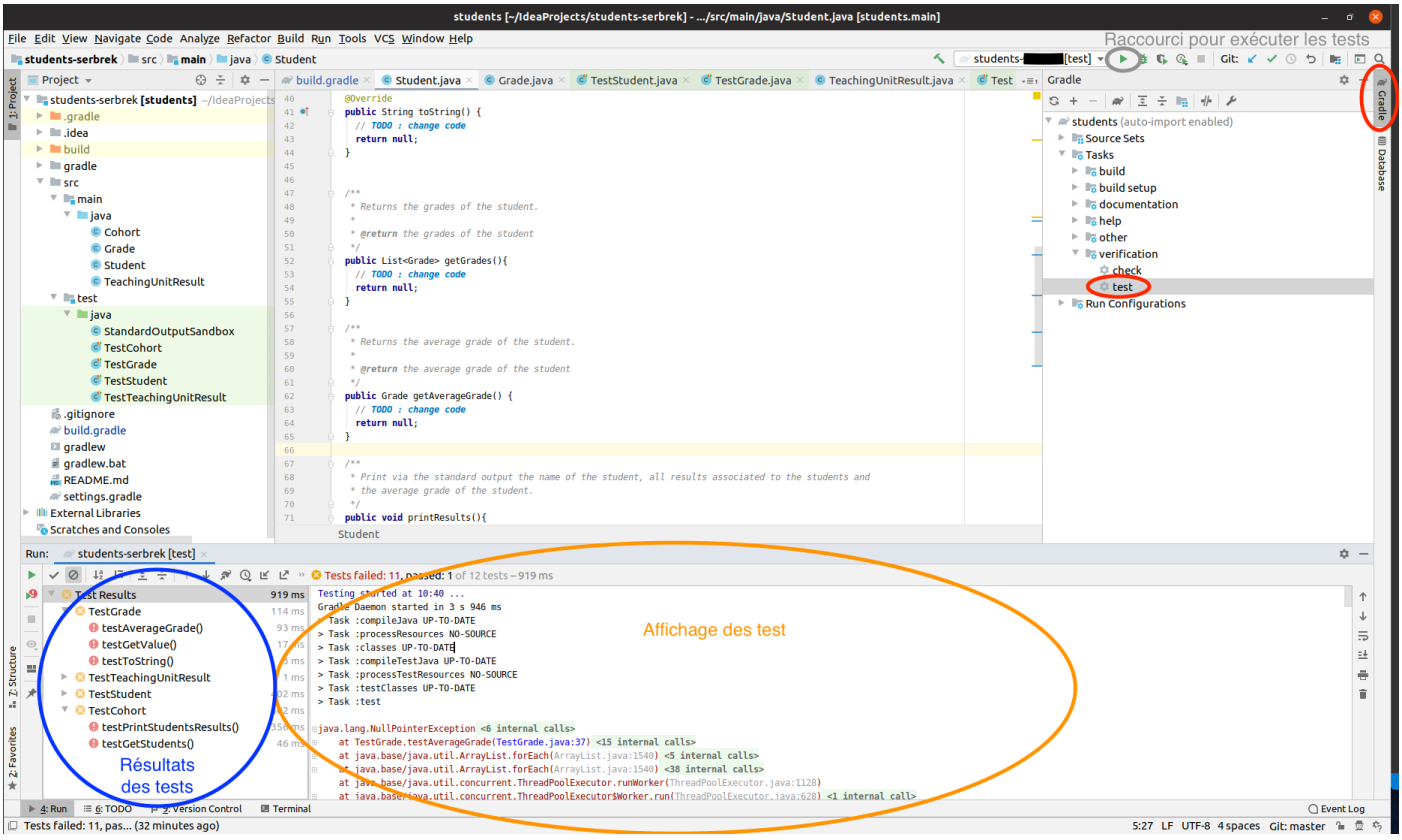
Le but de ce TP est de créer des classes permettant de représenter des étudiants (classe `Student`), des notes (classe `Grade`), des résultats à une unité d'enseignement (classe `TeachingUnitResult`) et des promotions d'étudiants (classe `Cohort`).

Récupérer le dépôt

Comme pour le TP 2, on va utiliser git pour la gestion de versions. Il vous faut donc vous reporter aux consignes du précédent TP. Le lien vers le projet à forker est le suivant : lien.

Exécuter le projet du dépôt

Pour compiler et exécuter le programme, il faut passer par l'onglet *gradle* à droite et cliquer deux fois sur `students` -> `Tasks` -> `verification` -> `test`. Cela va compiler et exécuter les tests. Pour le moment, les tests ne passeront pas, car certaines classes sont incomplètes.



Classes à coder

Classe `Grade`

Lien documentation de la classe

Cette classe va permettre de représenter une note obtenue par un étudiant. Une note est une valeur flottante comprise entre 0 et 20.

Cette classe contient les éléments suivants qui sont corrects :

- `private static final int MAXIMUM_GRADE` : un attribut statique représentant la valeur de la note maximale qui est égal à 20.
- `private final double value` : la valeur de la note comprise entre 0 et `MAXIMUM_GRADE`.
- `public Grade(double value)` : constructeur évident.
- `public boolean equals(Object o)` : méthode permettant de tester l'égalité de deux notes.

Tâche 1 : Votre but est de **compléter** les instructions des méthodes suivantes :

- `double getValue()` : retourne la valeur (`value`) de la note.
- `String toString()` : retourne une représentation de la note sous forme de chaîne de caractères. Pour une note ayant une valeur 12, cette méthode devra retourner la chaîne de caractères : `"12.0/20"`.
- `static Grade averageGrade(List<Grade> grades)` : calcule et renvoie la moyenne d'une liste de notes.

Assurez-vous que votre classe est correcte en exécutant les tests et en vérifiant que votre classe passe les tests `testToString`, `testGetValue` et `testAverageGrade` de `TestGrade` avec succès. Pour que les tests se lancent, il vous faut enlever les lignes `@Disabled("Disabled until Grade is coded")` du code de la classe `GradeTest`.

Classe `TeachingUnitResult`

Lien documentation de la classe

Cette classe va nous permettre de représenter un résultat obtenu par un étudiant, c'est-à-dire une note associée à une Unité d'Enseignement (UE).

Cette classe contient les éléments suivants qui sont corrects :

- `private final String teachingUnitName` : le nom de l'unité d'enseignement du résultat.
- `private final Grade grade` : la note du résultat.
- `public TeachingUnitResult(String teachingUnitName, Grade grade)` : constructeur évident.
- `public boolean equals(Object o)` : méthode permettant de tester l'égalité de deux résultats.

Tâche 2 : Votre but est de **compléter** les instructions des méthodes suivantes :

— `Grade getGrade()` : retourne la note associée au résultat.

— `String toString()` : renvoie le nom de l'unité d'enseignement suivi de `" : "` suivi de la représentation en chaîne de caractère de la note. Par exemple, un résultat d'une UE de Programmation 2 avec une note de 20 devra renvoyer la chaîne de caractères suivante : `"Programmation 2 : 20.0/20"`.

Assurez-vous que votre classe est correcte en exécutant les tests et en vérifiant que votre classe passe les tests `testToString` et `testGetGrade` de `TestTeachingUnitResult` avec succès. Pour que les tests se lancent, il vous faut enlever les lignes `@Disabled("Disabled until TeachingUnitResult is coded")` du code de la classe `TeachingUnitResultTest`.

Classe `Student`

Lien documentation de la classe

Cette classe va nous permettre de représenter un étudiant.

Cette classe contient les éléments suivants qui sont corrects :

— `private final String firstName` : le prénom de l'étudiant.

— `private final String lastName` : le nom de famille de l'étudiant.

— `private final List<TeachingUnitResult> results` : les résultats de l'étudiant.

— `public Student(String firstName, String lastName)` : constructeur initialisant le nom et prénom de l'étudiant avec les valeurs données et créant une liste vide pour les résultats.

— `public boolean equals(Object o)` : méthode permettant de tester l'égalité de deux étudiants.

Tâche 3 : Votre but est de compléter les instructions des méthodes suivantes :

— `void addResult(String teachingUnitName, Grade grade)` : ajoute un nouveau résultat à partir du nom de l'UE et d'une note.

— `List<Grade> getGrades()` : renvoie la liste des notes associées aux résultats de l'étudiant.

— `String toString()` : renvoie le nom de l'étudiant, c'est-à-dire son prénom, suivi d'un espace, suivi de son nom.

— `Grade averageGrade()` : renvoie la moyenne des notes associées aux résultats de l'étudiant.

— `void printResults()` : affiche les résultats de l'étudiant en sortie standard.

Un étudiant nommé `"Arnaud Labourel"` et ayant 20 en `"Programmation avancée"` et 20 en `"Génomique comparative et évolutive"` devra produire l'affichage suivant avec un appel à `printResults()` (avec un saut de ligne à la fin) :

```
1 Arnaud Labourel
```

```
Programmation avancée : 20.0/20
Génomique comparative et évolutive : 20.0/20
Note moyenne : 20.0/20
```

Assurez-vous que votre classe est correcte en exécutant les tests et en vérifiant que votre classe passe les tests `testToString`, `testGetGrades`, `testGetAverageGrade` et `testPrintResults` de `TestStudent` avec succès. Pour que les tests se lancent, il vous faut enlever les lignes `@Disabled("Disabled until Student is coded")` du code de la classe `StudentTest`.

Classe `Cohort`

Lien documentation de la classe

Cette classe va nous permettre de représenter une promotion d'étudiants. La classe `Cohort` contiendra les attributs, méthodes et constructeurs suivants :

Cette classe contient les éléments suivants qui sont corrects :

- `private final String name` : le nom de la promotion
- `private final List<Student> students` : les étudiants de la promotion
- `public Cohort(String name)` : constructeur à partir du nom de la promotion et initialisant à vide la liste des étudiants

Tâche 4 : Votre but est de compléter les instructions des méthodes suivantes :

- `public void addStudent(Student student)` : ajoute un étudiant à la promotion.
- `public List<Student> getStudents()` : renvoie la liste des étudiants de la promotion.
- `String toString()` : retourne une représentation de la promotion correspondant à son nom.
- `public void printStudentsResults()` : affiche les résultats de l'étudiant en sortie standard.

Appeler `printStudentsResults` sur une promotion ayant pour nom `"M2 Bio-informatique (DLAD)"` et deux étudiants devra produire un affichage similaire à l'affichage suivant (avec un saut de ligne à la fin) :

```
M2 Bio-informatique (DLAD)

Jean-Michel Bruitage
Programmation avancée : 20.0/20
Génomique comparative et évolutive : 20.0/20
Note moyenne : 20.0/20

David Goodenough
Programmation avancée : 0.0/20
Génomique comparative et évolutive : 0.0/20
Note moyenne : 0.0/20
```

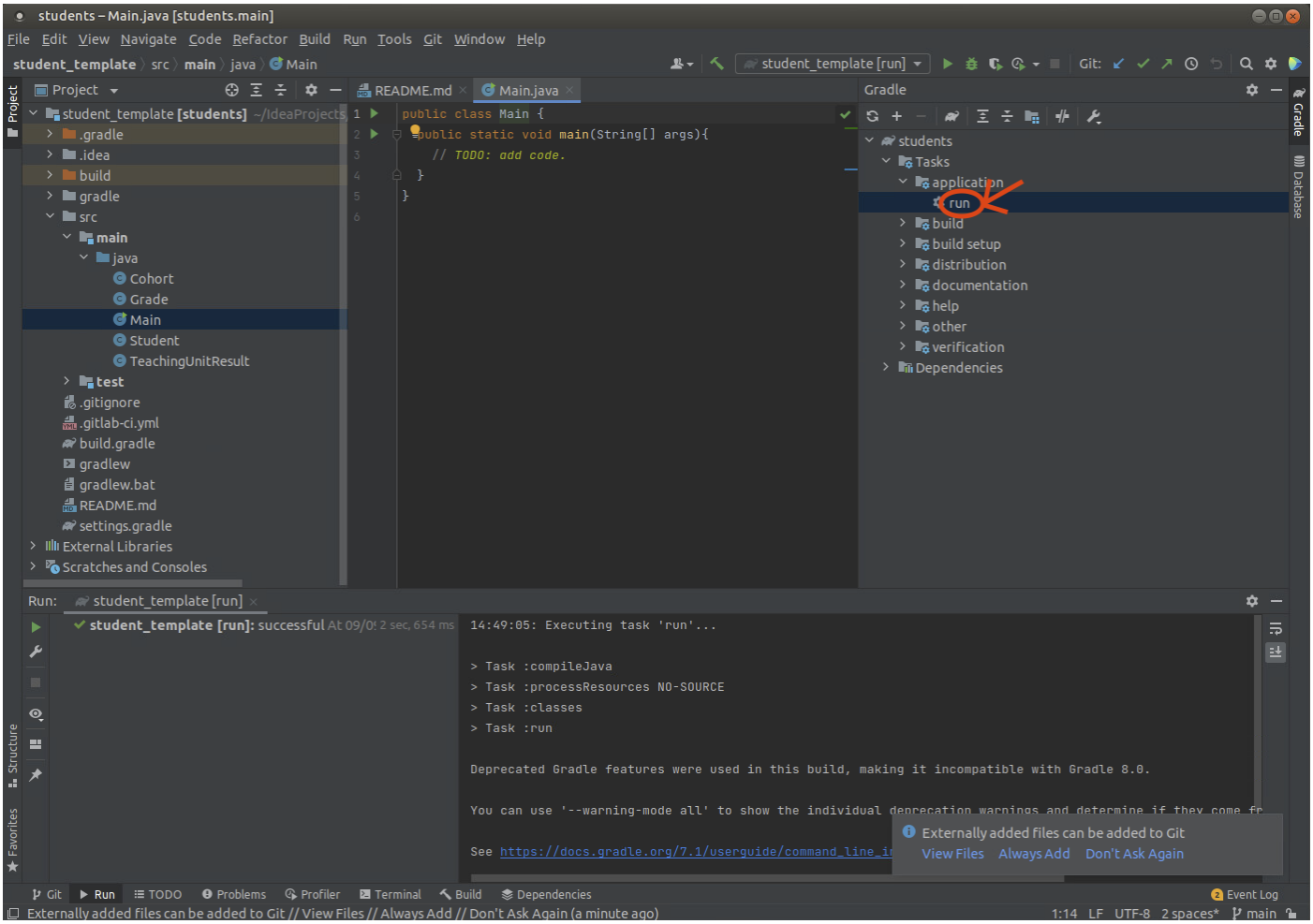
Assurez-vous que votre classe est correcte en exécutant les tests. Pour que les tests se lancent, il vous faut enlever les lignes `@Disabled("Disabled until Cohort is coded")` du code de la classe `CohortTest`.

Classe `App`

Tâche 5 : Ajoutez dans la classe `App` (fichier `src` -> `main` -> `java` -> `App.java`) le code d'une méthode `public static void main(String[] args)` qui (il vous faudra supprimer le commentaire `// TODO` après avoir changé le code) :

1. crée une instance de `Student` ayant votre nom et prénom et une autre avec des nom et prénom de votre choix,
2. ajoute à ces étudiants les notes en `"Programmation avancée"` et `"Génomique comparative et évolutive"` que vous aimeriez avoir,
3. crée une promotion (instance de `Cohort`) ayant nommée `"M2 Bio-informatique (DLAD)"`,
4. ajoute les étudiants créés à la promotion et
5. affiche les résultats de la promotion.

Pour compiler et exécuter le `main`, il faut passer par l'onglet *gradle* à droite et cliquer deux fois sur `students` -> `Application` -> `run`. Cela va compiler et exécuter votre méthode `main`.



Tâches optionnelles

Si vous avez fini les tâches précédentes, vous pouvez améliorer votre projet en rajoutant les fonctionnalités suivantes :

- Ajout d'une méthode comptant le nombre d'étudiants ayant validé leur année (moyenne supérieure ou égale à 10) dans une promotion.
- Changement de la classe `Grade` pour qu'elle permette de stocker des notes correspondant à une absence du résultat (affiché `ABS`).
- Calcul du nombre d'absents d'une promotion.
- Calcul de la note maximum et minimum (hors absence) d'une promotion.
- Création d'une classe `TeachingUnit` qui permet d'associer des crédits à une UE.
- Calcul pondéré de la moyenne de résultats en fonction du nombre de crédits des UE.
- Calcul de la moyenne (hors absence) d'une promotion.