Faculté des Sciences
Aix*Marseille Université

Année universitaire

Site : ☐ Luminy ☐ St-Charles ☐ St-Jérôme ☐ Cht-Gombert ☐ Aix-Montperrin ☐ Aubagne-SATIS
Sujet de : ☐ 1er semestre ☐ 2ème semestre ☒ Session 2 Durée de l'épreuve : 4h
Examen de : L2 Nom du diplôme : Licence d'informatique
Code du module : SIN3U02 Libellé du module : Programmation 2
Calculatrices autorisées : NON Documents autorisés : OUI

---

1 Consignes

- Créer un projet nommé `examen` avec comme *package* `fr.univ_amu.l2info` grâce à l'option `Create New Project`. Changer la *location* du projet pour qu'elle soit dans le répertoire *exam* de votre espace temporaire.
- Ne passez pas trop de temps sur une question. Si vous restez bloqué (code ne compilant pas ou erreur d'exécution que vous n'arrivez pas à compiler) et n'arrivez pas à corriger le problème, passez à la question suivante. Des points pourront être obtenus même pour du code ne compilant pas.
- Toutes les variables de type `List` seront à initialiser avec des objets de type `ArrayList`.
- Lorsque vous avez fini de composer, assurez-vous d'avoir bien mis votre projet dans le répertoire *exam* puis lancez par double-clic le programme *CLIQUE MOI FORT EN FIN D'EXAM* (situé dans leur espace temporaire), puis déconnectez-vous au plus vite.

2 Gestion de résultats d'étudiants

Le but de ce sujet d'examen 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`).

2.1 Classe `Grade`

Cette classe va permettre de représenter une note obtenue par un étudiant. Une note peut être soit une valeur comprise entre 0 et 20 ou bien correspondre à une absence de résultat. La classe `Grade` contiendra les attributs, méthodes et constructeurs suivants :

- `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`.
- `private final boolean isAbsent` : un booléen indiquant si la note est absente ou pas : égal à `true` si la note est absente et `false` sinon.
- `public Grade(double value, boolean isAbsent)` : constructeur évident.
- `public Grade()` : constructeur créant une note absente de valeur 0.
- `public Grade(double value)` : constructeur créant une note présente (pas absente) de valeur `value`.
- `public boolean isAbsent()` : retourne `true` si la note est absente et `false` sinon.
- `public 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 présente de valeur 12, cette méthode devra retourner `"12.0/20"` et pour une note absente `"ABS"`.

Créer la classe `Grade` avec les méthodes demandées.

2.2 Ajout d'exception

On souhaiterait ajouter la levée d'une exception de type `IllegalArgumentException` lorsque quelqu'un tente de créer une note ayant une valeur strictement inférieur à 0 ou bien strictement supérieur à `MAXIMUM_GRADE`. On souhaiterait que l'instruction `Grade grade = new Grade(22);` provoque un affichage similaire à l'affichage suivant :

```
Exception in thread "main" java.lang.IllegalArgumentException:
  Grade's value (22.0) must be in the range 0-20
    at fr.univ_amu.grades.Grade.<init>(Grade.java:17)
    at fr.univ_amu.grades.Grade.<init>(Grade.java:23)
    at fr.univ_amu.grades.Main.main(Main.java:9)
```

Ajouter le code correspondant au constructeur `public Grade(double value, boolean isAbsent)`.

2.3 Classe `TeachingUnitResult`

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. La classe `TeachingUnitResult` contiendra les attributs, méthodes et constructeurs suivants :

- `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 Grade getGrade()` : retourne la note associée au résultat.
- `public 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 `"Programmation 2 : 20.0/20"`.

Créer la classe `TeachingUnitResult` avec les méthodes demandées.

2.4 Classe `Student`

Cette classe va nous permettre de représenter un étudiant. La classe `Student` contiendra les attributs, méthodes et constructeurs suivants :

- `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 à vide la liste des résultats
- `public void addResult(String teachingUnitName, Grade grade)` : ajoute un nouveau résultat à partir du nom de l'UE et d'une note.

Créer la classe `Student` avec les méthodes demandées.

2.5 Calcul de la moyenne d'un étudiant

On souhaiterait calculer la moyenne d'un étudiant. Vous devez rajouter pour cela les méthodes suivantes :

- `public static Grade averageGrade(List<Grade> grades)` dans la classe `Grade` : calcule la moyenne d'une liste de notes. Si au moins une des notes est absente alors la moyenne est aussi absente. Si toutes les notes sont présentes alors la moyenne est la note ayant pour valeur la moyenne des valeurs.
- `public List<Grade> getGrades()` dans la classe `Student` : renvoie la liste des notes associées aux résultats de l'étudiant.
- `public Grade averageGrade()` dans la classe `Student` : renvoie la moyenne des notes associées aux résultats de l'étudiant.

Ajouter les méthodes demandées pour le calcul de la moyenne.

2.6 Classe `Cohort`

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

- `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
- `public void addStudent(Student student)` : ajoute un étudiant à la promotion.

Créer la classe `Cohort` avec les méthodes demandées.

2.7 Affichage promotion

On souhaite afficher le résultat des étudiants d'une promotion. Pour cela, vous allez rajouter :

- `public void printResults()` dans la classe `Student` : qui affiche le nom de l'étudiant, suivi de ses résultats suivi de la moyenne.
- `public void printStudentResults()` dans la classe `Cohort` : qui affiche le nom de la promotion, suivi de l'affichage des résultats de chaque étudiant.

On souhaite que l'affichage d'une promotion nommée L2 informatique de deux étudiants nommés Arnaud Labourel et Paul Calcul ayant des résultats dans les UE de Programmation 2 (ABS pour arnaud et 12 pour Paul) et Structures Discrètes (18 pour arnaud et 10 pour Paul) donne l'affichage suivant :

```
L2 informatique

Arnaud Labourel
Programmation 2 : ABS
Structures discrètes : 18.0/20
Note moyenne : ABS

Paul Calcul
Programmation 2 : 12.0/20
Structures discrètes : 10.0/20
Note moyenne : 11.0/20
```

Rajouter les méthodes demandées pour l'affichage et créer un `main` donnant l'affichage demandé et utilisant les classes que vous avez créées.

2.8 Compter les étudiants validant leur année

On rappelle que l'interface `Predicate<T>` est déjà définie en java et contient une seule méthode abstraite : `boolean test(T t)` qui renvoie `true` si le prédicat (critère testé) est vrai sur l'élément `t`.
On souhaite compter le nombre d'étudiants ayant validé leur année. Pour cela vous devez :

- Ajouter la méthode `public int countFilteredStudents(Predicate<Student> predicate)` à la classe `Cohort` qui renvoie le nombre d'étudiants satisfaisant le prédicat.
- Créer une classe `MinimalGradeCriterion` implémentant l'interface `Predicate<Student>` :
  - ayant un attribut `private final double minimalGrade`,
  - ayant un constructeur `public MinimalGradeCriterion(double minimalGrade)`,
  - et définissant le prédicat à vrai si la moyenne de l'étudiant n'est pas absente et que sa valeur est supérieure à `minimalGrade`.
- Ajouter la méthode `public int countValidatingStudents()` à la classe `Cohort` qui renvoie le nombre d'étudiants ayant une moyenne supérieure à 10.
- Changer l'affichage produit par la méthode `printStudentResults()` de la classe `Cohort` en ajoutant après les résultats des étudiants un ligne indiquant le nombre d'étudiants ayant validé leur année.

Vous devez obtenir l'affichage suivant :
Arnaud Labourel Programmation 2 : ABS Structures discrètes : 18.0/20 Note moyenne : ABS
Paul Calcul Programmation 2 : 12.0/20 Structures discrètes : 10.0/20 Note moyenne : 11.0/20
Nombre d'étudiants ayant validé : 1
Rajouter les méthodes demandées et changer la méthode `printStudentResults()` de la classe `Cohort` pour obtenir le nouvel affichage.

2.9 Amélioration de l'application

Ajouter de nouvelles classes et/ou de nouvelles méthodes afin de rajouter les fonctionnalités suivantes :

- Calcul du nombre d'absent d'une promotion
- Calcul de la note maximum et minimum (hors absence) d'une promotion
- Calcul pondéré de la moyenne en fonction du nombre de crédits de l'UE d'une promotion
- Calcul de la moyenne (hors absence) d'une promotion