Programmation Fonctionnelle (2024-2025)

TD 2

Exercice 1

Soit le modèle suivant :

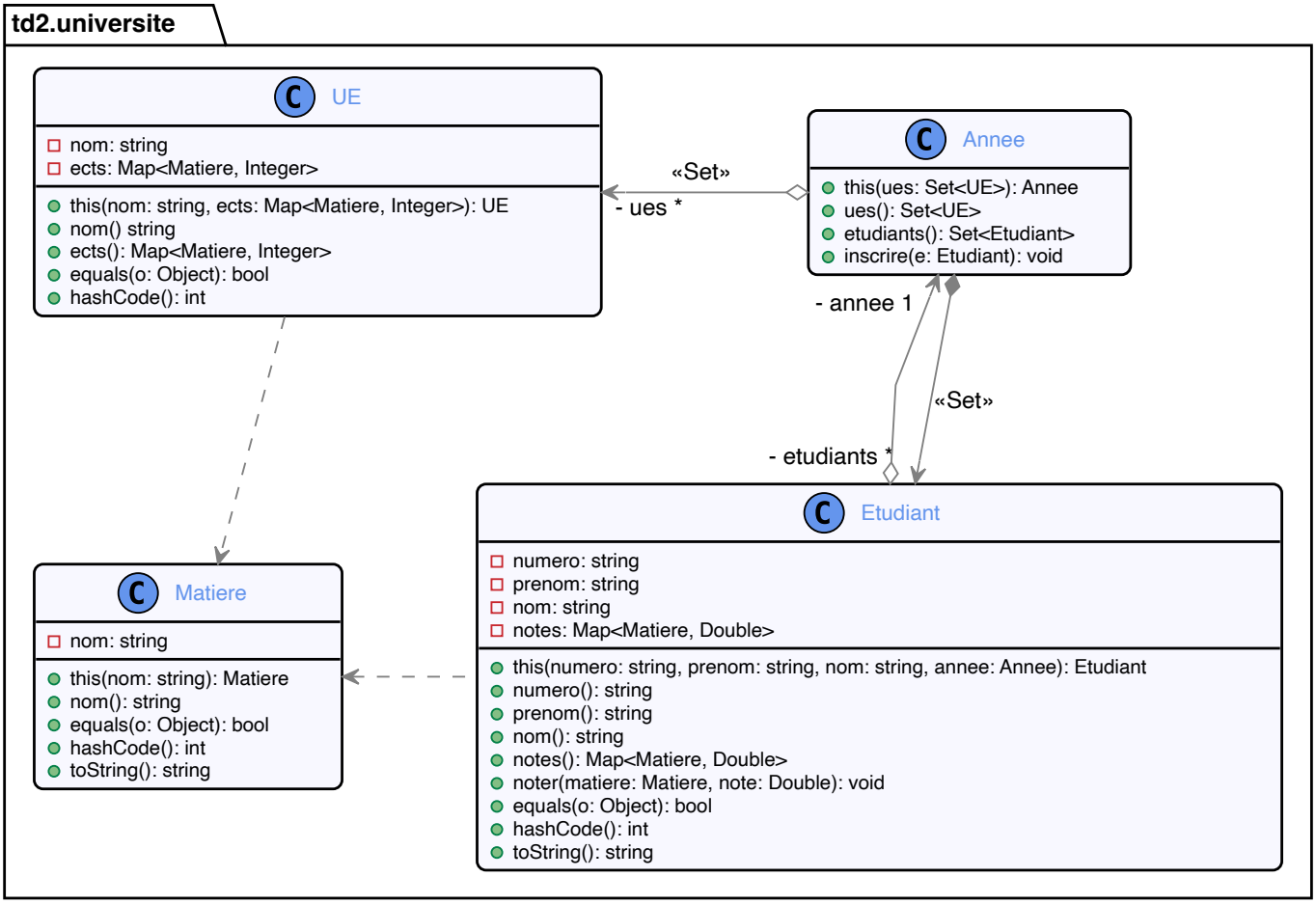

Figure 1: diagramme de classes

ainsi que les données suivantes :

```
Matiere m1 = new Matiere("MAT1");
Matiere m2 = new Matiere("MAT2");
UE ue1 = new UE("UE1", Map.of(m1, 2, m2, 2));
Matiere m3 = new Matiere("MAT3");
UE ue2 = new UE("UE2", Map.of(m3, 1));
Annee a1 = new Annee(Set.of(ue1, ue2));
Etudiant e1 = new Etudiant("39001", "Alice", "Merveille", a1);
e1.noter(m1, 12.0);
e1.noter(m2, 14.0);
e1.noter(m3, 10.0);
System.out.println(e1);
Etudiant e2 = new Etudiant("39002", "Bob", "Eponge", a1);
e2.noter(m1, 14.0);
e2.noter(m3, 14.0);
Etudiant e3 = new Etudiant("39003", "Charles", "Chaplin", a1);
e3.noter(m1, 18.0);
e3.noter(m2, 5.0);
e3.noter(m3, 14.0);
```

Question 1.

Implémentez le modèle en Java. Dans la suite vous utiliserez les API fonctionnelles et `Stream`.

Note: certaines méthodes ne sont pas fonctionnelles (par exemple `Annee` :: `inscrire`) et d'autres risquent des fuites de données

(par exemple `UE::ects`). On va passer sur cela ici.

Question 2.

Implémentez les méthodes suivantes.

2.1. écrire une méthode `afficheSi` qui prend en paramètre une chaîne de caractère en-tête, un prédicat portant sur un étudiant et une année et qui affiche l'en-tête suivi de tous les étudiants pour lesquels le prédicat est vrai.

```
**TOUS LES ETUDIANTS
39001 Alice Merveille
UE2
MAT3 (1) : 10.0
UE1
MAT1 (2) : 12.0
MAT2 (2) : 14.0
39003 Charles Chaplin
UE2
MAT3 (1) : 14.0
UE1
MAT1 (2) : 18.0
MAT2 (2) : 5.0
39002 Bob Eponge
UE2
MAT3 (1) : 14.0
UE1
MAT1 (2) : 14.0
MAT2 (2) : DEF
```

2.2. écrire un prédicat `aDEF` qui permet de savoir si un étudiant est DEFaillant (pas de note pour une matière ou +). Utiliser `afficheSi` pour afficher les étudiants concernés.

```
39002 Bob Eponge
UE2
MAT3 (1) : 14.0
UE1
MAT1 (2) : 14.0
MAT2 (2) : DEF
```

2.3. écrire un prédicat `aNoteEliminatoire` qui permet de savoir si un étudiant a une note éliminatoire (sous un plancher de 6/20). Utiliser `afficheSi` pour afficher les étudiants concernés.

```
**ETUDIANTS AVEC NOTE ELIMINATOIRE
39003 Charles Chaplin
UE2
MAT3 (1) : 14.0
UE1
MAT1 (2) : 18.0
MAT2 (2) : 5.0
```

2.4. écrire une méthode `moyenne` qui calcule la moyenne d'un étudiant.

$$moyenne(e) = \frac{\sum_{u \in ues(annee(e))} \sum_{(m,k) \in ects(u)} note(e,m) \times k}{\sum_{u \in ues(annee(e))} \sum_{(\_,k) \in ects(u)} k}$$

On ne peut pas calculer de moyenne si l'étudiant est défaillant. Utiliser `aDEF` et retourner `null` dans ce cas.

2.5. définir un prédicat `naPasLaMoyennev1` qui permet de savoir si un étudiant n'a pas la moyenne. Se contenter de la comparer à 10. Utiliser `afficheSi` pour afficher les étudiants concernés. Que se passe-t-il quand on utilise ce prédicat sur un étudiant défaillant ?

2.6. définir une seconde version de ce prédicat, `naPasLaMoyennev2` qui prennent en compte le cas des étudiants défaillants. Utiliser `afficheSi` pour afficher les étudiants n'ayant pas la moyenne.

```
**ETUDIANTS SOUS LA MOYENNE (v2)
39002 Bob Eponge
UE2
MAT3 (1) : 14.0
UE1
MAT1 (2) : 14.0
```

```
MAT2 (2) : DEF
```

2.7. définir un prédicat `session2v1` composé à partir des prédicats précédents permettant de savoir si un étudiant va en session 2. Un étudiant va en session 2 s'il n'a pas la moyenne (utiliser `naPasLaMoyennev1` pas `naPasLaMoyennev2`), s'il a une note éliminatoire, ou s'il est défaillant. Qu'observe-t-on pour différents ordres dans la disjonction logique des prédicats ?

```
**ETUDIANTS EN SESSION 2 (v2)
39003 Charles Chaplin
UE2
MAT3 (1) : 14.0
UE1
MAT1 (2) : 18.0
MAT2 (2) : 5.0
39002 Bob Eponge
UE2
MAT3 (1) : 14.0
UE1
MAT1 (2) : 14.0
MAT2 (2) : DEF
```

2.8. écrire une méthode `afficheSiv2` qui améliore `afficheSi` en permettant de passer en plus une fonction de représentation d'étudiant qui est utilisée par `afficheSiv2` pour afficher chaque étudiant. Utiliser cette nouvelle méthode `afficheSiv2` pour arriver au même résultat qu'`afficheSi` (utiliser la référence à la méthode qui permet d'afficher un étudiant). Utiliser ensuite à nouveau `afficheSiv2` pour afficher l'ensemble des étudiants avec leur moyenne (définir une fonction ad-hoc anonyme qui pour un étudiant donne son prénom, nom et moyenne, et la passer à `afficheSiv2`).

```
**TOUS LES ETUDIANTS
Alice Merveille : 12,40
Charles Chaplin : 12,00
Bob Eponge : défaillant
```

2.9. écrire une méthode `moyenneIndicative` où les notes non indiquées (DEF) sont traitées comme des 0/20. Utiliser cette méthode avec `afficheSiv2`.

```
Alice Merveille : 12,40
Charles Chaplin : 12,00
Bob Eponge : 8,40
```

2.10. généraliser `naPasLaMoyennev2` en une méthode `naPasLaMoyenneGeneralise` qui permet de choisir la fonction de moyenne à utiliser. Utiliser cette nouvelle fonction avec `afficheSiv2`.

```
**TOUS LES ETUDIANTS SOUS LA MOYENNE INDICATIVE
Bob Eponge : 8,40
```

En remplaçant les deux 14/20 de Bob par des 20/20, il a plus que la moyenne et il n'apparaît plus.

```
**TOUS LES ETUDIANTS SOUS LA MOYENNE INDICATIVE
(rien)
```

Question 3.

Complétez les définitions de fonctions suivantes.

```java
// matières d'une année
public static final Function<Annee, Stream<Matiere>> matieresA = ???

// matières d'un étudiant
public static final Function<Etudiant, Stream<Matiere>> matieresE = ???

// matières coefficientées d'un étudiant (version Entry)
public static final Function<Etudiant, Stream<Entry<Matiere, Integer>>> matieresCoefE_ = ???

// transformation d'une Entry en une Paire
public static final Function<Entry<Matiere, Integer>, Paire<Matiere, Integer>> entry2paire = ???

// matières coefficientées d'un étudiant (version Paire)
public static final Function<Etudiant, Stream<Paire<Matiere, Integer>>> matieresCoefE = ???

// accumulateur pour calcul de la moyenne
// ((asomme, acoefs), (note, coef)) -> (asomme+note*coef, acoef+coef)
public static final BinaryOperator<Paire<Double, Integer>> accumulateurMoyenne = ???
```

```java
// zero (valeur initiale pour l'accumulateur)
public static final Paire<Double, Integer> zero = ???

// obtention de la liste de (note, coef) pour les matières d'un étudiant
// 1. obtenir les (matière, coef)s
// 2. mapper pour obtenir les (note, coef)s, 0.0 pour la note si l'étudiant est DEF dans cette matière
public static final Function<Etudiant, List<Paire<Double, Integer>>> notesPondereesIndicatives = ???

// obtention de la liste de (note, coef) pour les matières d'un étudiant
// 1. obtenir les (matière, coef)s
// 2. mapper pour obtenir les (note, coef)s, 0.0 pour la note si l'étudiant est DEF dans cette matière
public static final Function<Etudiant, List<Paire<Double, Integer>>> notesPondereesIndicatives = ???

// replie avec l'accumulateur spécifique
public static final Function<List<Paire<Double, Integer>>, Paire<Double, Integer>> reduit = ???

// calcule la moyenne à partir d'un couple (somme pondérée, somme coefs)
public static final Function<Paire<Double, Integer>, Double> divise = ???

// calcul de moyenne fonctionnel
// composer notesPonderees, reduit et divise
// exception en cas de matière DEF
public static final Function<Etudiant, Double> computeMoyenne = ???

// calcul de moyenne fonctionnel
// composer notesPondereesIndicatives, reduit et divise
// pas d'exception en cas de matière DEF
public static final Function<Etudiant, Double> computeMoyenneIndicative = ???

// calcul de moyenne (sert juste de précondition à computeMoyenne)
public static final Function<Etudiant, Double> moyenne = e -> (e == null || aDEF .test(e)) ? null : computeMoyenne.appl
```

Exercice 2

Idem mais en Scala.