TD Collections.

Formations.
La classe `Curriculum` permet de représenter les formations. Une formation (*curriculum*) est définie par :

- le nom de la formation,
- les matières qui sont enseignées dans la formation avec leurs coefficients.

Les coefficients d'une matière peuvent changer d'une formation à l'autre.

Une matière (*subject*) est définie par une valeur du type énuméré `university.Subject`. Ce type est défini par exemple comme ceci :

```
package university;
/** the subjects that appear in curricula */
public enum Subject {
  Logique, ALR, POO, BDD1, TW2;
}
```

Q 1. Pour une formation, on veut pouvoir :

- ajouter (`addSubject`) ou supprimer (`removeSubject`) une matière,
- savoir si une matière existe (`subjectExist`),
- obtenir toutes les matières de la formation (`allSubjects`).
- connaître le coefficient d'une matière dans la formation (`getCoeff`).
  Chercher à connaître le coefficient d'une matière alors qu'elle n'existe pas pour la formation doit déclencher une `IllegalStateException`.

Q 1.1. Quelle structure de données la plus pertinente proposez-vous pour gérer les matières d'une formation et leurs coefficients ?

Q 1.2. Donnez le code de la classe `Curriculum` correspondant au diagramme :

Curriculum
...
+ Curriculum(name : String) + getName() : String + getCoeff(subject : Subject) : int + allSubjects() : Set<Subject> + subjectExist(subject : Subject) : boolean + addSubject(subject : Subject, coeff : int) + removeSubject(subject : Subject)

Q 1.3. Écrivez les lignes codes qui permettent d'afficher, pour une formation `curri`, ses matières avec leur coefficient.

Q1.1 Une hashmap Subject -> int

Q1.2.

```
public class Curriculum {
    private String name;
    private Map<Subject, int> subjects;

    public Curriculum(String name){
        this.name = name;
        this.subjects = new HashMap<>();
    }

    public String getName(){
        Return this.name;
    }
```

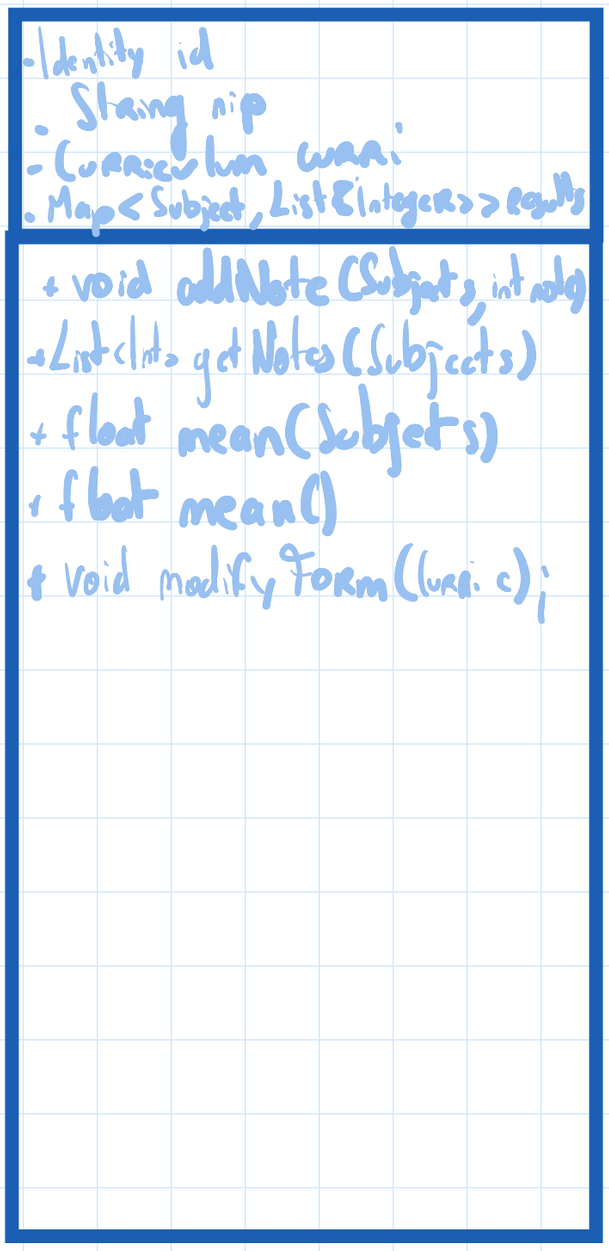
```
public int getCoeff(Subject s) throws ISE {
    if (!this.subjectExist(s)) {
        throw new ISE();
    }
    Return this.subjects.get(s);
}
public Set<Subject> allSubjects() {
    return this.subjects.keySet();
}
public boolean subjectExist(Subject s) {
    Return this.subjects.containsKey(s);
}
    public void addSubject(Subject s, int coeff) {
        this.subjects.put(s, coeff);
    }
    public int RemoveSubject(Subject s) {
        Return this.subject.Remove(s);
    }
}
Curriculum curri;
for (Map.Entry<Subject, int> sub : curri.entrySet()) {
    S.o.p(sub.getKey(), sub.getValue());
}
```

Q 2 . La classe Student doit permettre :

- d'ajouter une note à un étudiant,
- de connaître la liste des notes de l'étudiant pour une matière,
- de calculer la moyenne de l'étudiant pour une matière et sa moyenne générale.
  Pour une matière, toutes les notes ont le même coefficient alors que la moyenne générale tient compte des coefficients des matières dans la formation de l'étudiant. Une moyenne est 0 lorsqu'il n'y a pas de note.
- de modifier la formation d'un étudiant. Lorsque l'on modifie la formation d'un étudiant, les notes existantes sont supprimées et une liste vide de notes est associée à chaque matière de la formation.

Q 2.1. Donnez le diagramme UML de la classe Student.

Q 2.2. Écrivez le code de la classe Student.




```
public void addNote(Subject s, int note){
    this.getNotes(s).add(note);
}

public float mean(Subject S){          ← if (pas de notes ...) return 0;
    float res = 0;
    for(int note : this.getNotes(s)){
        res += note;
    return res / this.getNote(s).size();
}
```

```
public float mean (){                                  ← if...
    float res = 0;
    Set<Subject> subjects = this.subjects.getkey();
    for (Subject subject : subjects){
        res += this.mean(subject);
    }
    return res / subjects.size();
}
```