PARTIE 2 – Relations entre classes

I. Des variables dont le type est une classe

Leçon

En POO quasiment tous les types sont des classes. Seuls les types primitifs ne sont pas des classes, par exemples : *int, float, double, char, boolean.*

Si vous voulez en savoir plus sur les classes **Java existantes** allez consulter l'API Java : https://docs.oracle.com/en/java/javase/17/docs/api/
En particulier vous avez déjà vu la classe **String** dont vous avez utilisé différentes méthodes telles que : *charAt*.

Vous pouvez, vous aussi, définir l'API des classes que vous créez en écrivant la javadoc (cf. https://www.oracle.com/fr/technical-resources/articles/java/javadoc-tool.html).

Reprenons l'énoncé et la définition de Cours :

Un cours est caractérisé par son intitulé (une chaine de caractères), les étudiants inscrits, un niveau de difficulté noté entre 1 (facile) et 5 (difficile) et un enseignant responsable.

Dans cette définition vous pouvez noter que les étudiants et l'enseignant sont définis par des classes. Voici une implémentation en java possible de la classe ***Cours*** et sa représentation graphique.

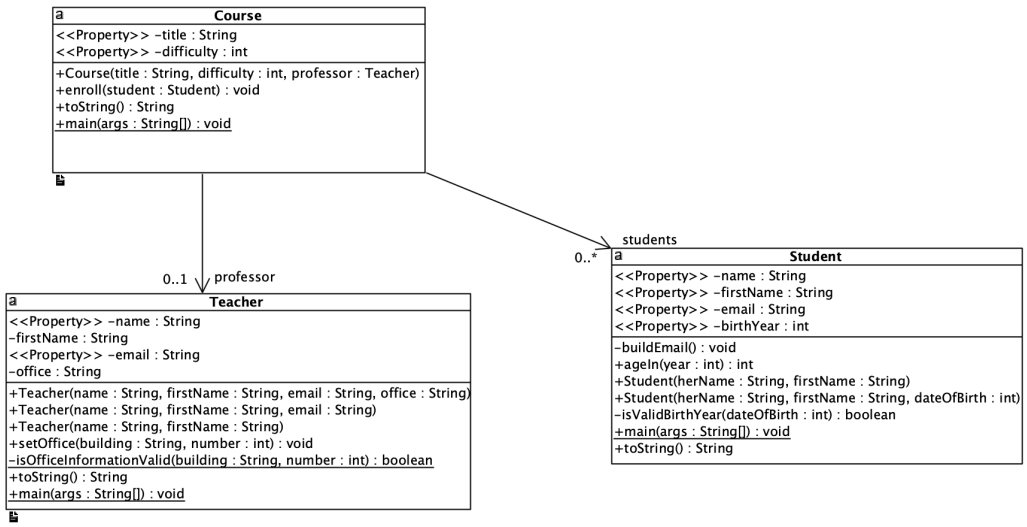
Dans le code qui suit nous utilisons la notion de liste (List et ArrayList) des étudiants associés à un cours et nous utilisons la méthode **add** pour ajouter un élément à la liste. Regardez la spécification de ces classes et remarquez en particulier que nous ajoutons les éléments en fin de la liste.

```
import java.util.ArrayList;
import java.util.List;

public class Course {
    private String title;
    private List<Student> students;
    private int difficulty;
    private Teacher professor;

    public Course(String title, int difficulty, Teacher professor) {
        this.title = title;
        this.difficulty = difficulty;
        this.professor = professor;
        this.students = new ArrayList<>();
    }

    public void enroll(Student student) {
        students.add(student);
    }
    public List<Student> getStudents() {
        return students;
    }
```

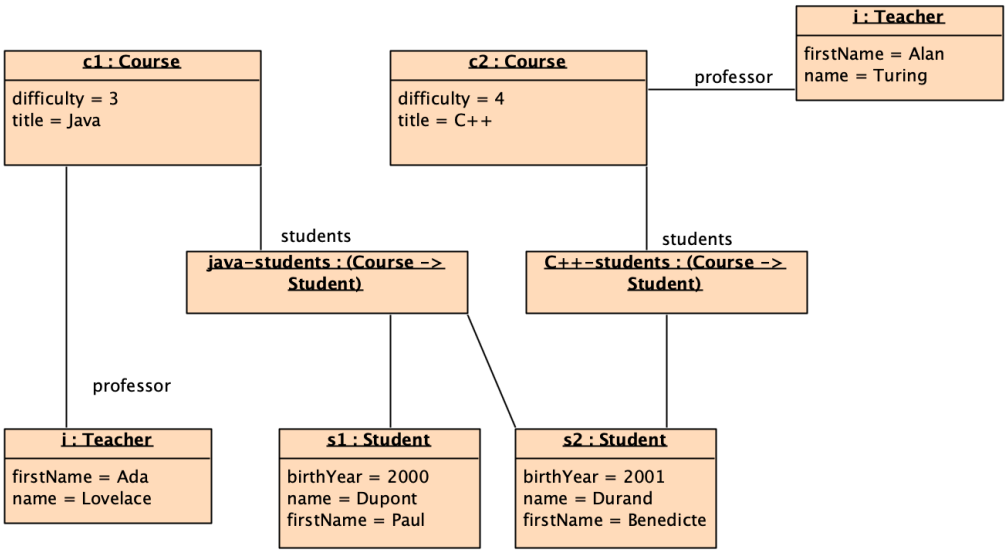
• • •

```
    public static void main(String[] args){
        Course c1 = new Course("Java", 3, new Teacher("Lovelace", "Ada"));
        Course c2 = new Course("C++", 4, new Teacher("Turing", "Alan"));
        Student s1 = new Student("Dupont", "Paul", 2000);
        Student s2 = new Student("Durand", "Benedicte", 2001);
        Student s3 = new Student("Smith", "John");
        c1.enroll(s1);
        c1.enroll(s2);
        c2.enroll(s2);
        c2.enroll(s3);
        System.out.println(c1);
        System.out.println(c2);
        System.out.println("Students in java course : ------- \n" +
c1.getStudents());
        System.out.println("Students in C++ course : ------- \n" +
c2.getStudents());
    }

}
```

Ci-dessous nous donnons une représentation graphique de ce code. Lorsque le type d'une variable d'instance est un type défini par votre application nous le représentons ci-dessous par une flèche entre les classes dont la fin vous donne le nom de la variable d'instance.



Le diagramme suivant visualise partiellement (c'est long à faire) les objets qui ont été créés dans le « main » et leurs relations.



Variable d'instance versus Attribut, Champs, fields : Une variable d'instance est une variable déclarée au niveau de la classe et qui conserve son état propre à chaque instance (objet) créée à partir de cette classe. Chaque objet a sa propre copie de cette variable, ce qui lui permet de stocker des données uniques. C'est ainsi qu'elles ont été introduites au début de cet énoncé. **Attribut** est un terme plus général qui fait référence aux propriétés ou aux caractéristiques d'une entité. Dans le contexte de la programmation orientée objet, un attribut est souvent implémenté à l'aide de variables d'instance. C'est une manière de modéliser les données et les propriétés des objets. On utilise donc souvent le terme d'attribut à la place de variable d'instance mais dans un contexte objet, il s'agit de la même chose. Certains parlent aussi de champs et IJ utilise le terme de **field**.

Questions en groupe & Codage individuel

1. Comprenez l'ensemble des codes donnés et en particulier
    i. le code d'initialisation de la liste des étudiants
    ii. le code d'ajout un étudiant dans la liste
2. Implémentez en comprenant la classe *Course* et en ajoutant les accesseurs manquants.
3. Modifiez votre code pour vérifier que le coefficient de difficulté est compris en 1 et 5.
4. Est-ce que le *System.out.println* d'un cours affiche ce que vous souhaitez ou pas ? Corrigez votre code si besoin.

Question individuelle & Codage

5. L'harmonisation fait appel à des enseignants et des étudiants. Elle est composée de cours.
Implémentez la classe *Harmo* pour tenir compte de cette spécification.

Vous pouvez choisir de représenter l'ensemble des cours délivrés en harmonisation, non par une liste comme nous l'avons fait précédemment mais par un ensemble (**Set** et **HashSet**). Analysez la spécification de ces deux classes et remarquez qu'elle ne préserve pas l'ordre d'ajouts des éléments dans l'ensemble, c'est juste un « ensemble » 😉

Voici des exemples de code :

```
private Set<Course> courses = new HashSet<>();

public void addCourse(Course c) {
    courses.add(c);
}
public Set<Course> getCourses() {
    return courses;
}
```

II. Interactions entre classes

Leçon

Nous avons abordé pour l'instant la construction des relations entre classes dans la déclaration des variables d'instances et avons permis d'ajouter des instances dans un ensemble et une liste, par exemple, des étudiants à un cours, des cours à l'harmonisation, etc.

Nous nous intéressons ici à la complexité de ces relations en gérant les relations inverses, un étudiant est inscrit à une liste de cours, donc on ajoute à l'étudiant la liste des cours auxquels il est inscrit. Cela suppose donc que non seulement nous ajoutions la variable *courses* à l'étudiant mais qu'également nous décidions de quand l'affecter.
Faisons le choix suivant :
un étudiant s1 choisit un cours c1 revient à appeler s1.enrollsIn(c1) qui déclenche un appel à c1.enroll(s1). *On peut donc considérer que l'étudiant choisit un cours et que le cours enregistre l'étudiant.*
Nous aurions pu faire un choix inverse ; si pour qu'un étudiant puisse s'inscrire à un cours il faut d'abord que sa demande soit acceptée nous aurions pu avoir : c1.enroll(s1) qui n'appelle s1.enrollsIn(c1) que si la demande est valide.

Questions en groupe & Codage individuel

6. Implémenter dans la classe *Student* la méthode *enrollsIn(Course c)* avec tout ce qui est nécessaire, et testez là.
7. Analysez le code donné ci-dessus qui retourne le dernier étudiant inscrit à un cours. Ce code se trouve dans la classe *Course*

```
public Student getLastStudentEnrolled() {
    return students.get(students.size() - 1);
}
```

8. Pourriez-vous définir une méthode dans la classe *Student* qui renvoie le dernier cours auquel un étudiant s'est inscrit ? Regardez la spécification associée aux classes utilisées. **Vous voyez qu'elles ne préservent pas l'ordre. Le choix de List ou de Set a donc un impact important.**

Question individuelle & Codage

Un enseignant connait la liste des cours dont il est responsable. Évidemment un cours dont il est responsable doit avoir pour responsable lui-même.

9. Implémenter cette spécification en tenant bien compte de la bi-directionnalité entre le responsable de cours et la liste des cours dont un enseignant est responsable : un enseignant choisit à un cours c1 dont il prend la responsabilité.

III. Boucles

Leçon

Nous allons à présent parcourir les ensembles ou les listes que vous avez définis.

Dans la classe **Course**

```
public boolean isEnrolled(String  name, String firstName) {

    for (Student s : students) {
        if (s.getName().equals(name) && s.getFirstName().equals(firstName))
{
            return true;
        }
    }
    return false;
}
```

Dans la classe **Harmo**

```
/**
 * Permet de retrouver un étudiant inscrit dans nos listes à partir de son
email
 * @param email
 * @return l'étudiant dont l'émail correspond au paramétre, retourne null
sinon.
 */
public Student findStudent(String email) {
    for (Student student : students) {
        if (student.getEmail().equals(email)) {
            return student;
        }
    }
    return null;
}
```

Questions en groupe & Codage individuel

10. Comprendre les codes
11. Écrire une méthode qui renvoie la position d'un étudiant de nom donné dans un cours. Si aucun étudiant avec ce nom n'est inscrit, elle renvoie -1.
    a) Dans quelle classe devez-vous écrire cette méthode ?
12. Écrire la méthode qui calcule la moyenne d'âge des étudiants inscrits à un cours.
    a) Dans quelle classe définissez-vous cette méthode ?
    b) Que se passe-t-il si aucun étudiant n'est inscrit au cours ?

Question individuelle & Codage

13. Écrire la méthode qui calcule la moyenne du niveau de difficulté des cours choisis par un étudiant. Dans quelle classe définissez-vous cette méthode ?
14. Écrire la méthode qui calcule la moyenne du niveau de difficulté des cours dont il est responsable. Dans quelle classe définissez-vous cette méthode ?
15. Écrire la méthode qui calcule la moyenne d'âge des étudiants de l'harmonisation. Dans quelle classe définissez-vous cette méthode ?
16. Supposons que nous voulions ajouter le fait que les étudiants évaluent leur progression à la fin du cours sur les notions abordées avec un pourcentage. Comment proposez- vous de définir cette donnée et de faire évoluer le code en conséquence ? N'hésitez pas à regarder les classes de l'API Java.