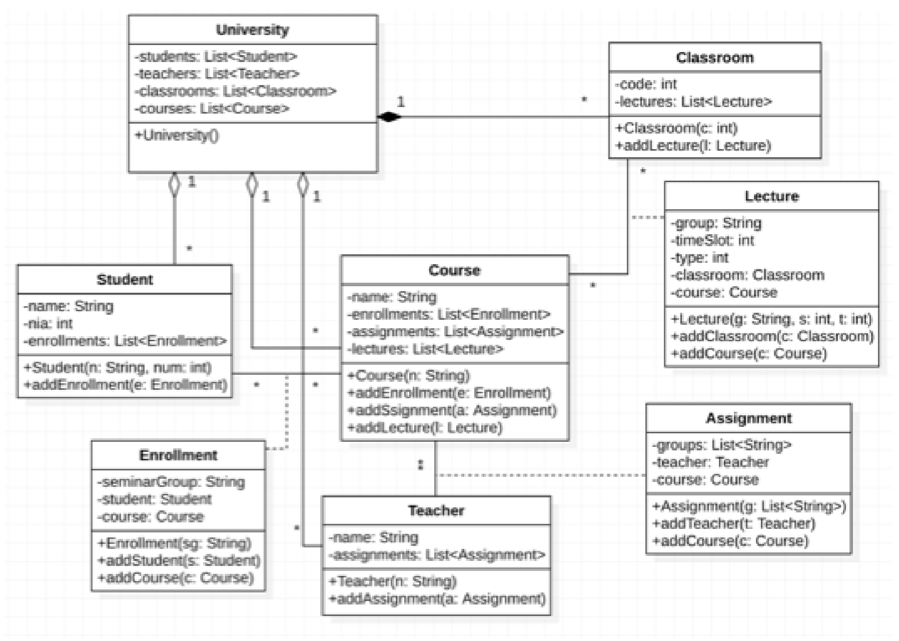
Lab Session 2: Implementing class relations

Introduction

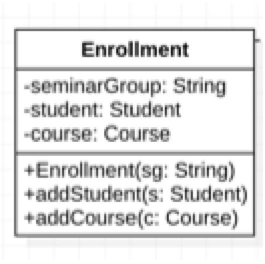
In this lab we put in practice the relations between classes and classes themselves, in order to choose wisely their relation through the concept learned in seminars and theory, so our challenge is to understand and relate this correctly in VS Code.We have to construct relationships that all have in common the ‘’University Class’’ and co-depend with each other for existing as a solid base, thanks to some files provided from the teachers to help us and include it in our code. Let’s start.

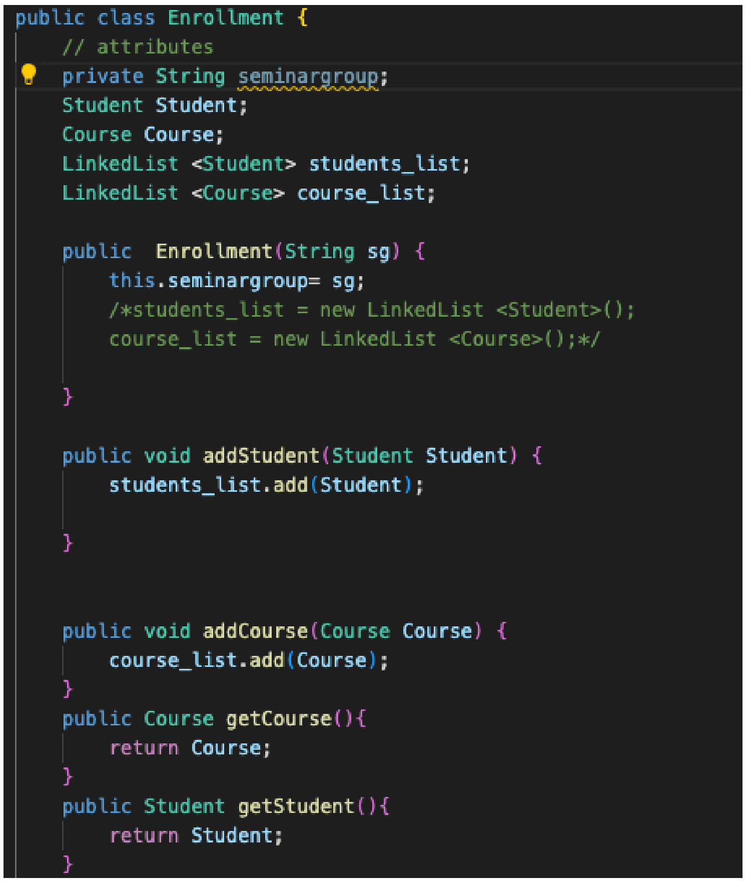


Implementing the design

The first step is to create the definition of our classes, attributes and methods. Also, we highlight that mostly all the relations given are aggregation, composition and dependency and we went along with the class diagram in our code. In such a manner…

ENROLLMENT CLASS

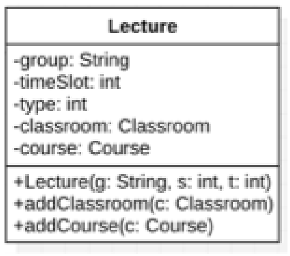


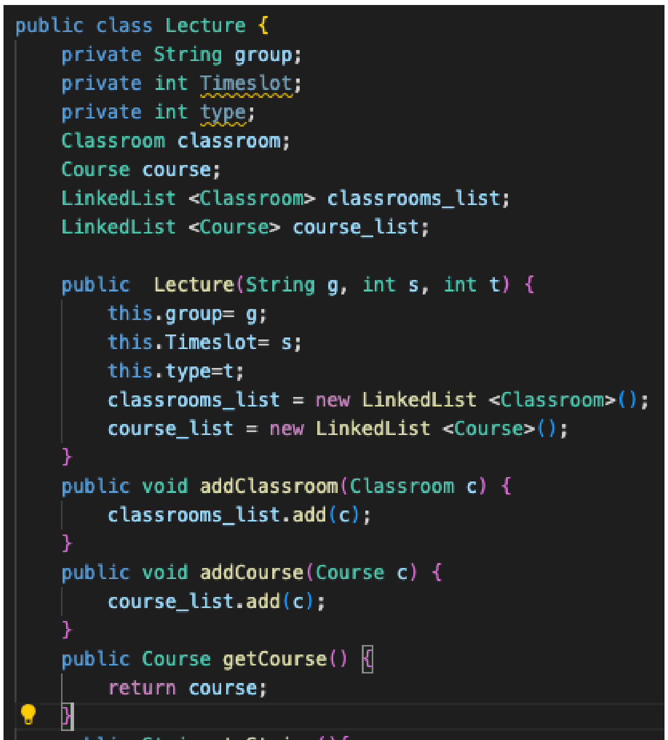


```java
public class Enrollment {
    // attributes
    private String seminargroup;
    Student Student;
    Course Course;
    LinkedList <Student> students_list;
    LinkedList <Course> course_list;

    public  Enrollment(String sg) {
        this.seminargroup= sg;
        /*students_list = new LinkedList <Student>();
        course_list = new LinkedList <Course>();*/

    }

    public void addStudent(Student Student) {
        students_list.add(Student);

    }


    public void addCourse(Course Course) {
        course_list.add(Course);
    }
    public Course getCourse(){
        return Course;
    }
    public Student getStudent(){
        return Student;
    }
```

For Enrollment Class, which exists thanks to student and course relation, we create their attributes, seminar group, we call student and course class. As for our constructor, it is a very simple one, with only one argument (seminargroup), and the other methods **addStudent addCourse**, if we want to add an only student or course.

LECTURE CLASS

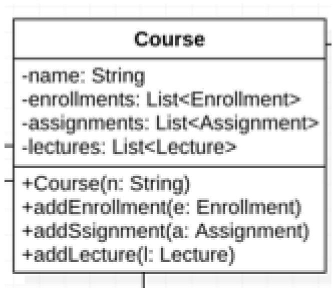


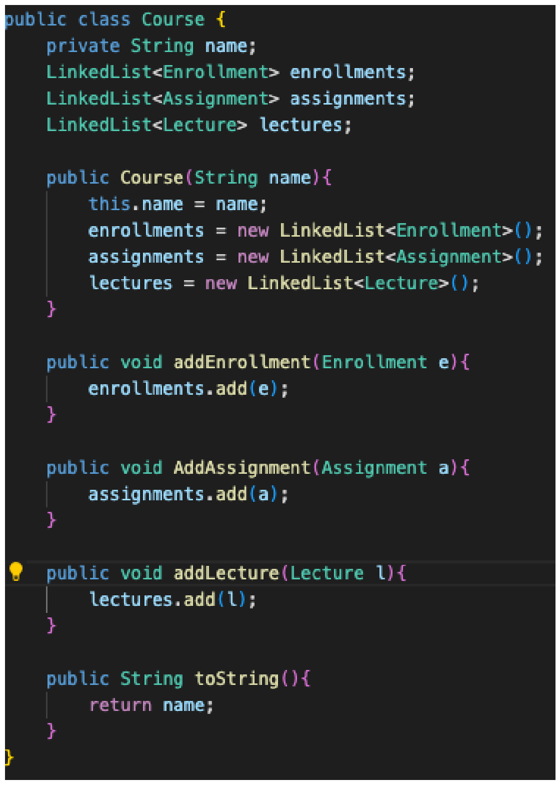


```java
public class Lecture {
    private String group;
    private int Timeslot;
    private int type;
    Classroom classroom;
    Course course;
    LinkedList <Classroom> classrooms_list;
    LinkedList <Course> course_list;

    public  Lecture(String g, int s, int t) {
        this.group= g;
        this.Timeslot= s;
        this.type=t;
        classrooms_list = new LinkedList <Classroom>();
        course_list = new LinkedList <Course>();
    }
    public void addClassroom(Classroom c) {
        classrooms_list.add(c);
    }
    public void addCourse(Course c) {
        course_list.add(c);
    }
    public Course getCourse() {
        return course;
    }
```

For Lecture Class, we create more attributes where we define group, timeSlot, type, classroom and course classes, that are the main arguments in the constructor, and their methods for adding only one classroom and course.

COURSE CLASS

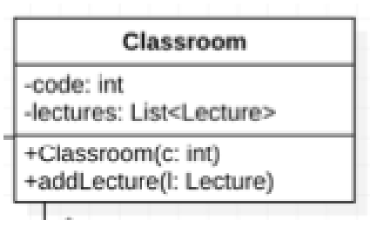

Course
-name: String
-enrollments: List<Enrollment>
-assignments: List<Assignment>
-lectures: List<Lecture>
+Course(n: String)
+addEnrollment(e: Enrollment)
+addSsignment(a: Assignment)
+addLecture(l: Lecture)

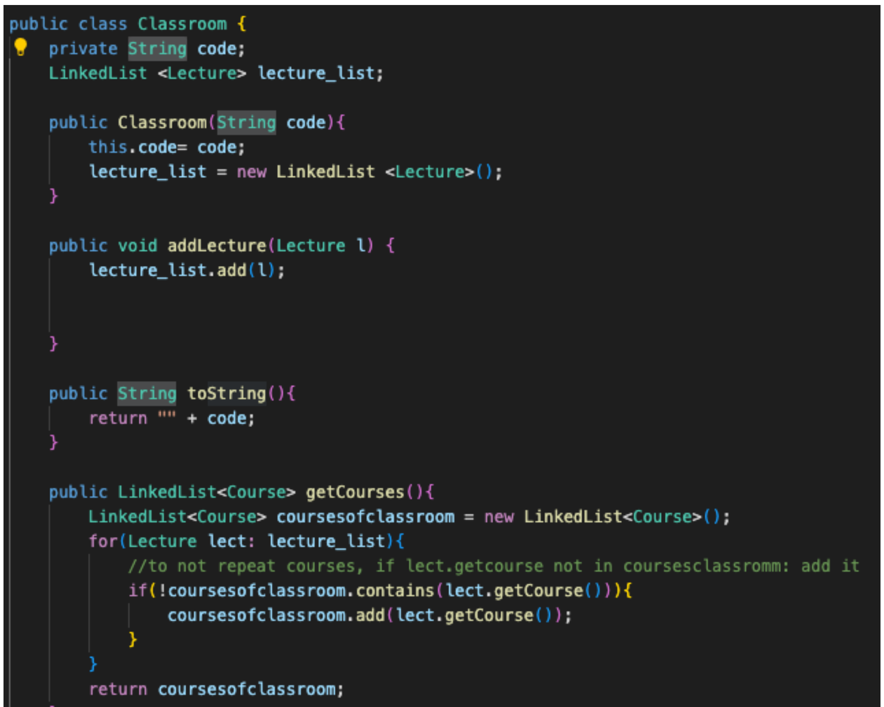


```java
public class Course {
    private String name;
    LinkedList<Enrollment> enrollments;
    LinkedList<Assignment> assignments;
    LinkedList<Lecture> lectures;

    public Course(String name){
        this.name = name;
        enrollments = new LinkedList<Enrollment>();
        assignments = new LinkedList<Assignment>();
        lectures = new LinkedList<Lecture>();
    }

    public void addEnrollment(Enrollment e){
        enrollments.add(e);
    }

    public void AddAssignment(Assignment a){
        assignments.add(a);
    }

    public void addLecture(Lecture l){
        lectures.add(l);
    }

    public String toString(){
        return name;
    }
}
```

For Course Class, that three specific classes depend on them, **assignments, enrollments** and **lectures**, we call their respective lists.Therefore, we only have a main attribute, a **name**, and everytime we create a course this initializes three different lists, enrollments, assignments and lectures. To this extent, the methods are for adding new elements to the three lists.

CLASSROOM CLASS

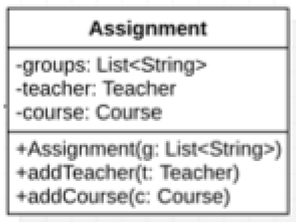


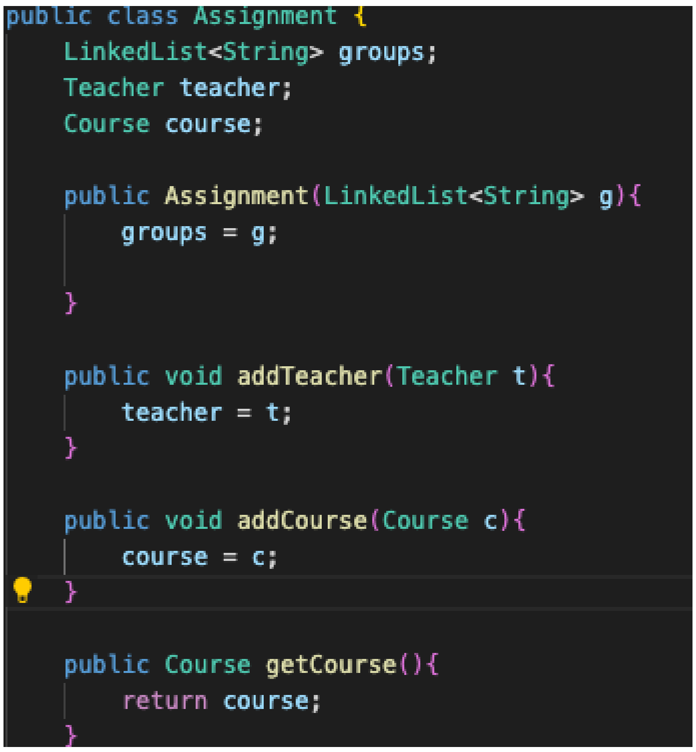


```java
public class Classroom {
    private String code;
    LinkedList <Lecture> lecture_list;

    public Classroom(String code){
        this.code= code;
        lecture_list = new LinkedList <Lecture>();
    }

    public void addLecture(Lecture l) {
        lecture_list.add(l);


    }

    public String toString(){
        return "" + code;
    }

    public LinkedList<Course> getCourses(){
        LinkedList<Course> coursesofclassroom = new LinkedList<Course>();
        for(Lecture lect: lecture_list){
            //to not repeat courses, if lect.getcourse not in coursesclassromm: add it
            if(!coursesofclassroom.contains(lect.getCourse())){
                coursesofclassroom.add(lect.getCourse());
            }
        }
        return coursesofclassroom;
```

For Classroom class, the only composition relationship with University, we define **code** of the classroom, and also it exists with the lecture LinkedList, for every classroom there's a lecture. Indeed, there's a method for adding lectures to their respective LinkedList in case of creating a new one.

ASSIGNMENT CLASS

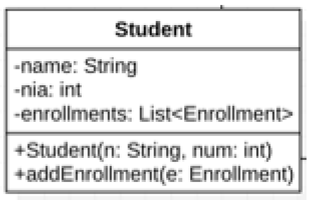


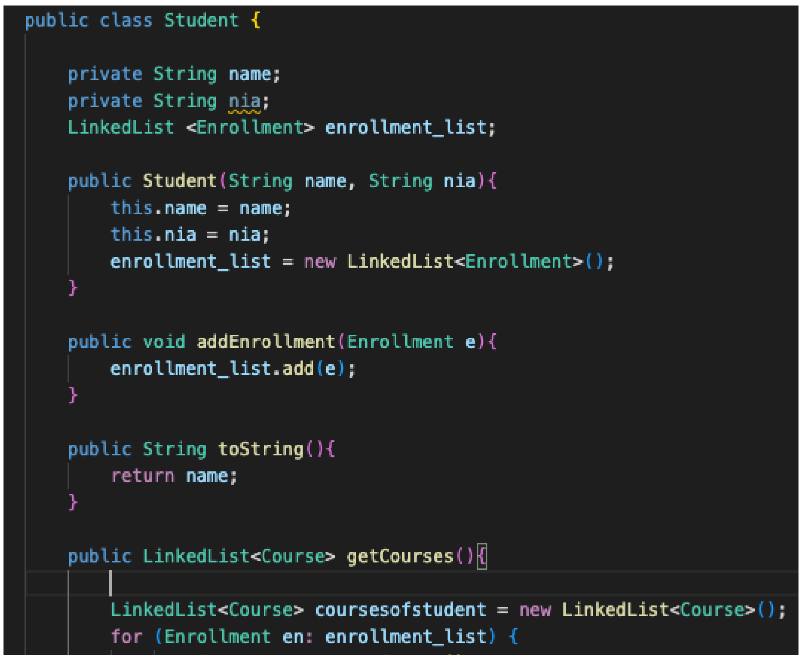


```java
public class Assignment {
    LinkedList<String> groups;
    Teacher teacher;
    Course course;

    public Assignment(LinkedList<String> g){
        groups = g;

    }

    public void addTeacher(Teacher t){
        teacher = t;
    }

    public void addCourse(Course c){
        course = c;
    }

    public Course getCourse(){
        return course;
    }
```

For Assignment Class, that exists because of the relation between Teacher and Course, for every group there's an assignment, this is the reason we need to include the groups LinkedList, also in order to an assignment to exists, demands a Teacher and a Teacher requires a Course consequently their methods have to incorporate add methods for a teacher and a course.

STUDENT CLASS


Student
-name: String
-nia: int
-enrollments: List<Enrollment>
+Student(n: String, num: int)
+addEnrollment(e: Enrollment)

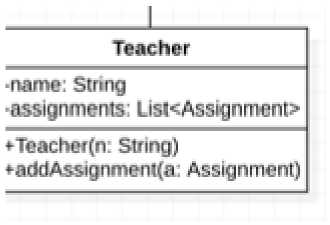


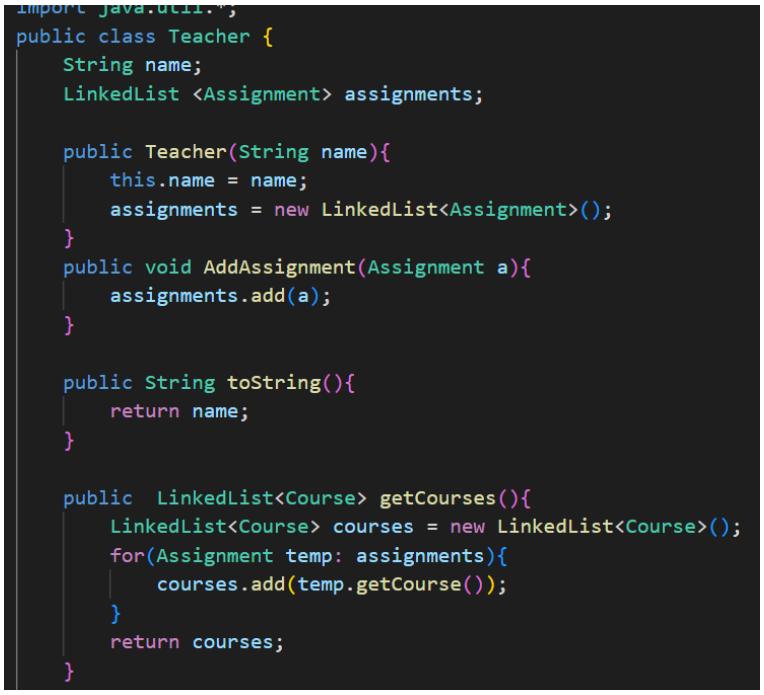
```java
public class Student {

    private String name;
    private String nia;
    LinkedList <Enrollment> enrollment_list;

    public Student(String name, String nia){
        this.name = name;
        this.nia = nia;
        enrollment_list = new LinkedList<Enrollment>();
    }

    public void addEnrollment(Enrollment e){
        enrollment_list.add(e);
    }

    public String toString(){
        return name;
    }

    public LinkedList<Course> getCourses(){

        LinkedList<Course> coursesofstudent = new LinkedList<Course>();
        for (Enrollment en: enrollment_list) {
```

For Student Class we create their main attributes , name , nia and enrollment linkedlist which we associate with their type and visibility. According to our understanding, every student needs an enrollment so in their constructor we include inside in order to initialize enrollment every time a new student is created.

However, for the methods besides the constructor, exists **addEnrollment**, which we call everytime we want to add an extra **Enrollment** to our LinkedList.

Teacher Class

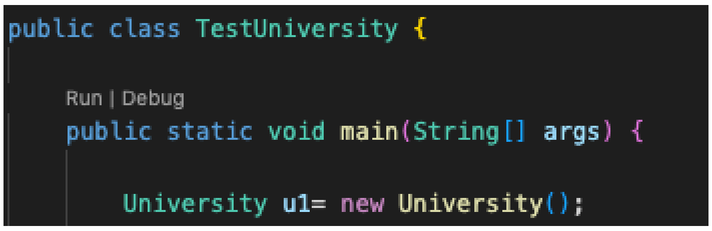




```
import java.util.*;
public class Teacher {
    String name;
    LinkedList <Assignment> assignments;

    public Teacher(String name){
        this.name = name;
        assignments = new LinkedList<Assignment>();
    }
    public void AddAssignment(Assignment a){
        assignments.add(a);
    }

    public String toString(){
        return name;
    }

    public  LinkedList<Course> getCourses(){
        LinkedList<Course> courses = new LinkedList<Course>();
        for(Assignment temp: assignments){
            courses.add(temp.getCourse());
        }
        return courses;
    }
```

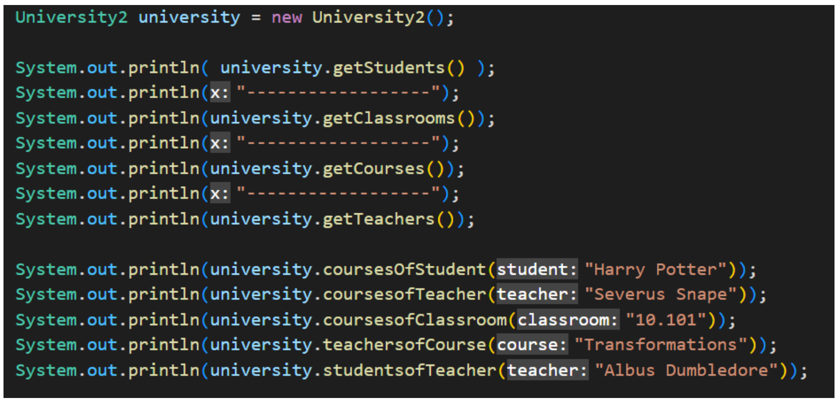
For Teacher Class, we link the assignments list, which we have to include inside of the constructor, and also create their method for adding assignments into our linkedlist. Then we added the method toString that returns the name of the teacher, and finally a method required by the queries, which returns a LinkedList of courses.

TEST UNIVERSITY



```
public class TestUniversity {

    Run | Debug
    public static void main(String[] args) {

        University u1= new University();
```

We create Test University as a main method and create an University instance, just to check if it runs correctly. Then we add all our prints to check if it works fine, and next the prints for 5 queries that we implemented.

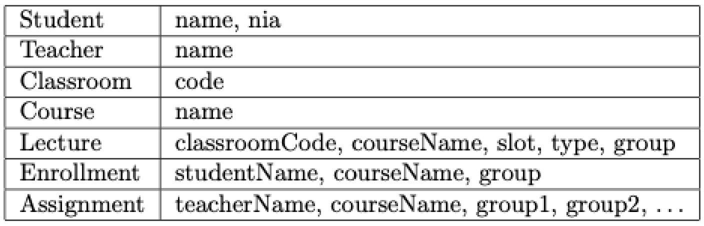


```java
University2 university = new University2();

System.out.println( university.getStudents() );
System.out.println(x: "------------------");
System.out.println(university.getClassrooms());
System.out.println(x: "------------------");
System.out.println(university.getCourses());
System.out.println(x: "------------------");
System.out.println(university.getTeachers());

System.out.println(university.coursesOfStudent(student: "Harry Potter"));
System.out.println(university.coursesofTeacher(teacher: "Severus Snape"));
System.out.println(university.coursesofClassroom(classroom: "10.101"));
System.out.println(university.teachersofCourse(course: "Transformations"));
System.out.println(university.studentsofTeacher(teacher: "Albus Dumbledore"));
```

FILING THE SYSTEM WITH THE XML DATA

Once done with the first part, we start working with the XML data files provided by the code and using Utility class too, all of this is going to take place inside of University Class.

```java
LinkedList< String[] > students = Utility.readXML( "student" );
LinkedList< String[] > teachers = Utility.readXML( "teacher" );
LinkedList< String[] > classrooms = Utility.readXML( "classroom" );
LinkedList< String[] > courses = Utility.readXML( "course" );
LinkedList< String[] > lectures = Utility.readXML( "lecture" );
LinkedList< String[] > enrollments = Utility.readXML( "enrollment" );
LinkedList< String[] > assignments = Utility.readXML( "assignment" );
```

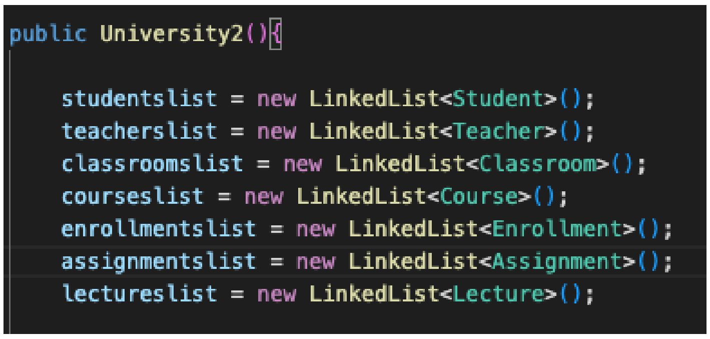


Student	name, nia
Teacher	name
Classroom	code
Course	name
Lecture	classroomCode, courseName, slot, type, group
Enrollment	studentName, courseName, group
Assignment	teacherName, courseName, group1, group2, ...

All our 7 classes, separately, have an XML file with different cases of students, teachers, classrooms… so as a result they're read as a String Array, so all our inputs have to be type string in order to be read and included in the LinkedList or use a handed function to change int to String.

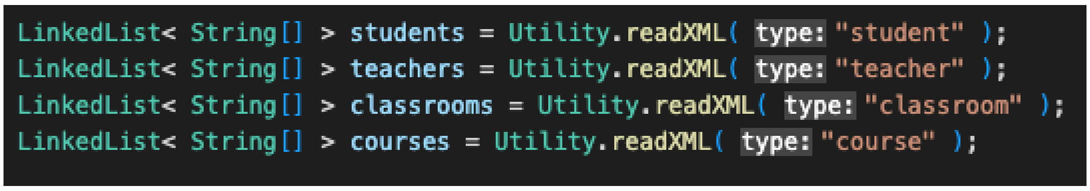
```java
int x = Integer.parseInt( "141512" )
```

We begin with 4 entities (Students, Teachers, Classrooms and Courses), it's necessary to think of this as not in a static way but in a dynamic way, there's a file that everytime reads something needs to include it in a list. In order to do this, we create 7 lists where we'll place our data.



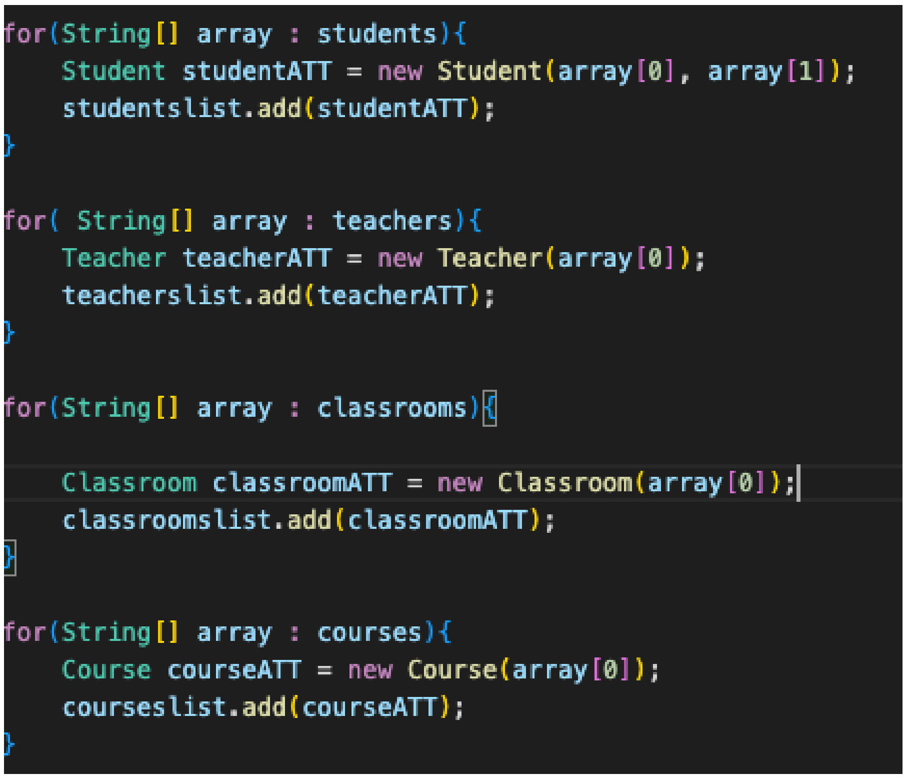
```java
public University2(){

    studentslist = new LinkedList<Student>();
    teacherslist = new LinkedList<Teacher>();
    classroomslist = new LinkedList<Classroom>();
    courseslist = new LinkedList<Course>();
    enrollmentslist = new LinkedList<Enrollment>();
    assignmentslist = new LinkedList<Assignment>();
    lectureslist = new LinkedList<Lecture>();
```

Then using readXML from Utility we pass our 4 entities files.



```java
LinkedList< String[] > students = Utility.readXML( type: "student" );
LinkedList< String[] > teachers = Utility.readXML( type: "teacher" );
LinkedList< String[] > classrooms = Utility.readXML( type: "classroom" );
LinkedList< String[] > courses = Utility.readXML( type: "course" );
```

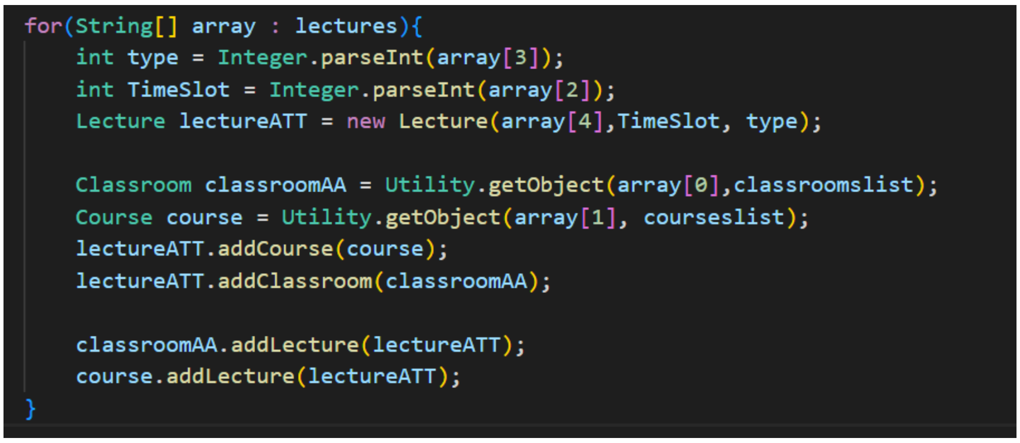
So for our 4 entities we use **for** , it's required an iterative function. If we take a look to every XML files, our inputs are placed like a list, so we can access to them through positions, for example student XML file, our name and NIA are in first and second position, so we access to them like array[0], array [1].



```java
for(String[] array : students){
    Student studentATT = new Student(array[0], array[1]);
    studentslist.add(studentATT);
}

for( String[] array : teachers){
    Teacher teacherATT = new Teacher(array[0]);
    teacherslist.add(teacherATT);
}

for(String[] array : classrooms){

    Classroom classroomATT = new Classroom(array[0]);
    classroomslist.add(classroomATT);
}

for(String[] array : courses){
    Course courseATT = new Course(array[0]);
    courseslist.add(courseATT);
}
```

We do this for each entity, using the same algorithm. We create an instance that saves iteratively our inputs and pass it thanks to **add** function to our empty lists.

To the rest of entities: Lecture, Enrollment and Assignment are computed a little differently from the others. After reading the xml files, we do the same procedure as the other 4 entities.

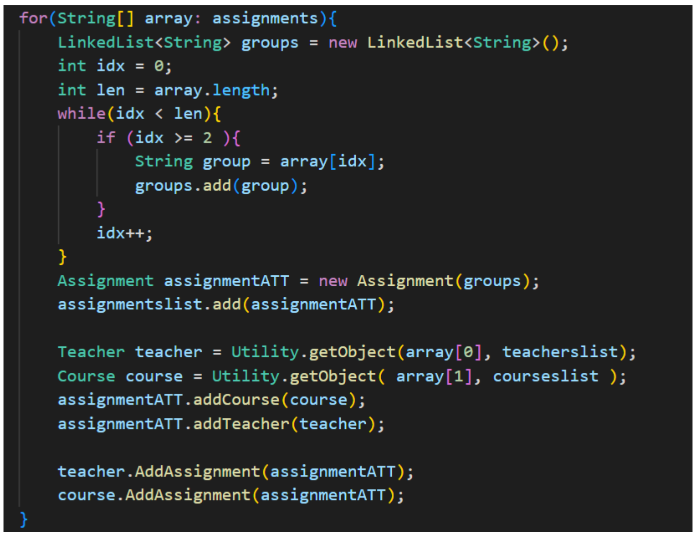


```java
for(String[] array : lectures){
    int type = Integer.parseInt(array[3]);
    int TimeSlot = Integer.parseInt(array[2]);
    Lecture lectureATT = new Lecture(array[4],TimeSlot, type);

    Classroom classroomAA = Utility.getObject(array[0],classroomslist);
    Course course = Utility.getObject(array[1], courseslist);
    lectureATT.addCourse(course);
    lectureATT.addClassroom(classroomAA);

    classroomAA.addLecture(lectureATT);
    course.addLecture(lectureATT);
}
```

First we create an instance of lecture with the given information of the array, and then we need to add the course and the classroom attributes, which we retrieve from the getObject function of utility. Finally, we also need to add the information of this new lecture to the classroom and course.

For the enrollment entity we follow exactly the same process but with Students and Courses.

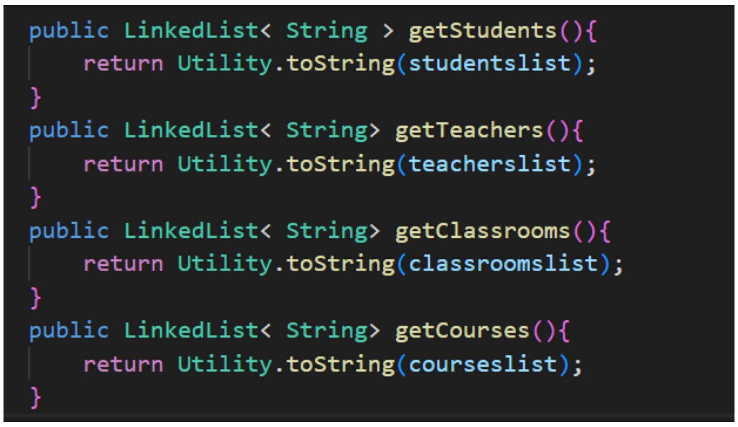
Finally, the assignments class was a bit harder because each assignment can have a different number of groups. For this reason, we need to create a LinkedList of groups that we will pass to the assignment instance. This linkedlist of groups is created knowing that from the array position 2 onwards we will find the group information, because on the 0 and 1st position we find the teacher and course name, so we need to skip it.



```java
for(String[] array: assignments){
    LinkedList<String> groups = new LinkedList<String>();
    int idx = 0;
    int len = array.length;
    while(idx < len){
        if (idx >= 2 ){
            String group = array[idx];
            groups.add(group);
        }
        idx++;
    }
    Assignment assignmentATT = new Assignment(groups);
    assignmentslist.add(assignmentATT);

    Teacher teacher = Utility.getObject(array[0], teacherslist);
    Course course = Utility.getObject( array[1], courseslist );
    assignmentATT.addCourse(course);
    assignmentATT.addTeacher(teacher);

    teacher.AddAssignment(assignmentATT);
    course.AddAssignment(assignmentATT);
}
```

METHODS THAT RETURN LISTS OF ENTITIES

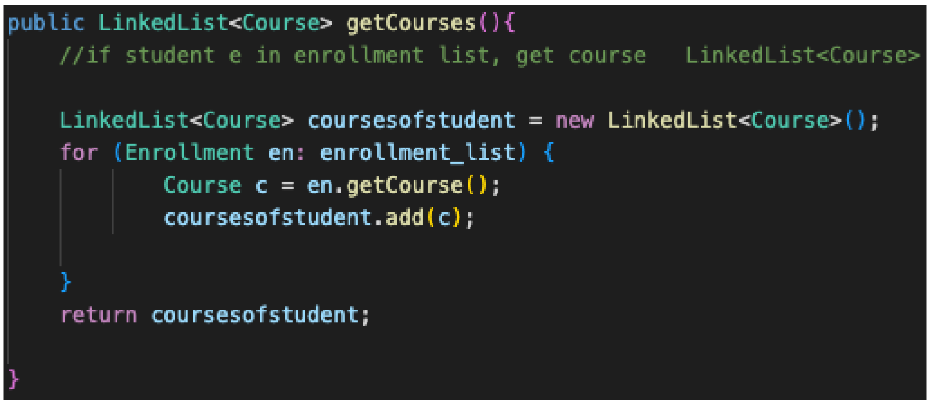
In this part we just created 4 methods to return the lists of university entities, all of them in string format.



```
public LinkedList< String > getStudents(){
    return Utility.toString(studentslist);
}
public LinkedList< String> getTeachers(){
    return Utility.toString(teacherslist);
}
public LinkedList< String> getClassrooms(){
    return Utility.toString(classroomslist);
}
public LinkedList< String> getCourses(){
    return Utility.toString(courseslist);
}
```

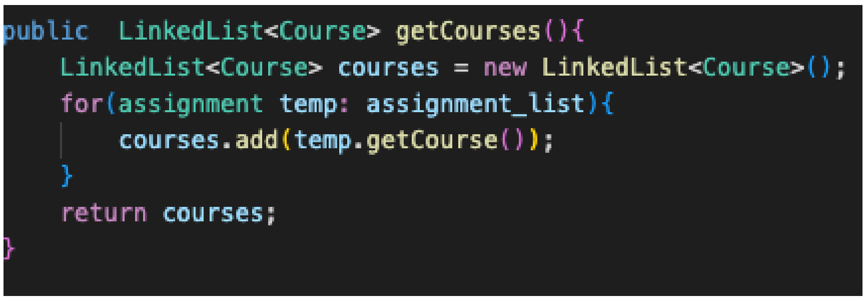
QUERIES

In Seminar 2 were described some queries that we'd to implement in our code, we computed 5 queries.



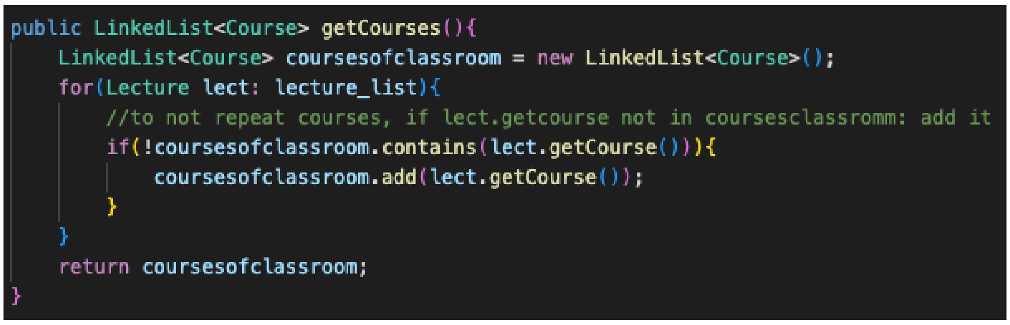
```
public LinkedList<Course> getCourses(){
    //if student e in enrollment list, get course   LinkedList<Course>

    LinkedList<Course> coursesofstudent = new LinkedList<Course>();
    for (Enrollment en: enrollment_list) {
            Course c = en.getCourse();
            coursesofstudent.add(c);

    }
    return coursesofstudent;

}
```

CoursesofStudent will return the courses of students,so it has to be a list that saves this kind of data. For obtaining this information, the students indeed are included in the enrollment list (*en*) so we implement method **getCourse** to search the course of the student and then add it to our main list.



```
public  LinkedList<Course> getCourses(){
    LinkedList<Course> courses = new LinkedList<Course>();
    for(assignment temp: assignment_list){
        courses.add(temp.getCourse());
    }
    return courses;
}
```

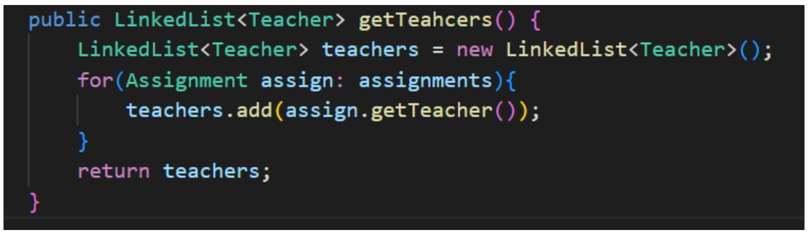
The same for **TeachersOfCourse** (*Courses)* for obtaining the course of the Teacher, this information is given by our ***assignment_list***, so we'll create the main list that will save the courses of Teachers, then apply ***GetCourse***() that'll get the course of the Teacher from our assignment_list called temp, subsequently pass it to courses(main list) through add function, lastly will return it.



```java
public LinkedList<Course> getCourses(){
    LinkedList<Course> coursesofclassroom = new LinkedList<Course>();
    for(Lecture lect: lecture_list){
        //to not repeat courses, if lect.getcourse not in coursesclassromm: add it
        if(!coursesofclassroom.contains(lect.getCourse())){
            coursesofclassroom.add(lect.getCourse());
        }
    }
    return coursesofclassroom;
}
```

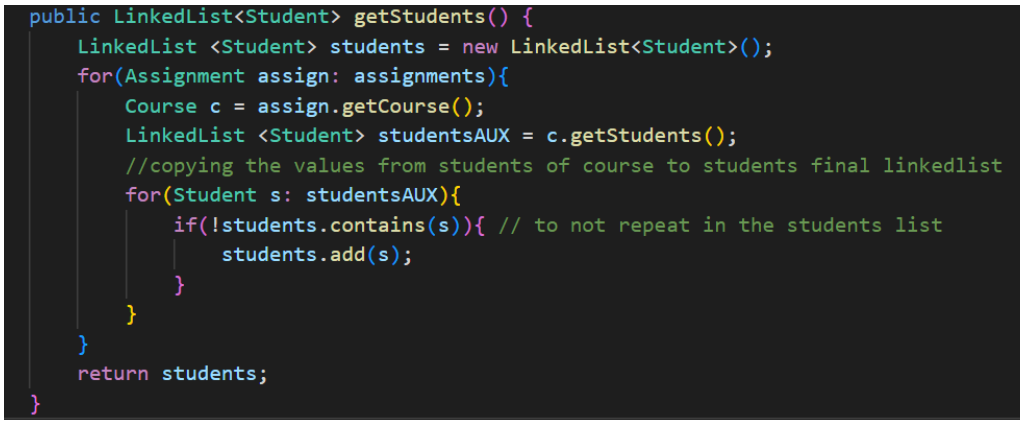
CoursesofClassroom this data is stored inside our **lecture_list** , this has to return the courses that are given in a classroom,so we implement this with a different function called ***contains***, that checks if this course is already in the main list if NOT go to the ***lecture_list***(*lect*) get the course and lastly add it to our course of classroom list.

Teachers of Course obtains a linkedlist of type Teacher of a course. This function uses a method of the class Course called getTeachers, which at the same time it uses a method from assignment to retrieve the correspondent teacher.



```java
public LinkedList<Teacher> getTeahcers() {
    LinkedList<Teacher> teachers = new LinkedList<Teacher>();
    for(Assignment assign: assignments){
        teachers.add(assign.getTeacher());
    }
    return teachers;
}
```

Students of Teacher returns a linkedlist of type Student containing all the students a teacher in total. For this purpose, we create the function getStudents in the teacher class. It looks at each course that the teacher gives class, and we retrieve the students of that course and add them to the final linkedlist.



```java
public LinkedList<Student> getStudents() {
    LinkedList <Student> students = new LinkedList<Student>();
    for(Assignment assign: assignments){
        Course c = assign.getCourse();
        LinkedList <Student> studentsAUX = c.getStudents();
        //copying the values from students of course to students final linkedlist
        for(Student s: studentsAUX){
            if(!students.contains(s)){ // to not repeat in the students list
                students.add(s);
            }
        }
    }
    return students;
}
```

CONCLUSION

In closing this lab we noticed a little bit of difficulty specially on how to relate all the classes and then translate it to Java Language, but once done we both learned new functions, how to relate, and understanding the concept of classes and relationships.

The main difficulties were encountered on adding the elements from the XML file to the assignment list, there were different length of groups in each assignment of the file, so the problem was on how to add this to our defined lists, finally the solution was loops and working with running through lists. Besides that, another difficulty was the translation of our idea to the code, mainly because we're not familiarized with all Java functions that exist, so it was required to do an online research of what we could use to accomplish our purpose.

Even so, we find more challenging solving the seminar of this lab than the lab itself, due to the fact that relations concept it's quite difficult to understand and easily you can confuse one relation with another, they kinda resemble each other, so in the lab the relations were already given in the class diagram so this issue was solved.

To conclude, this was an elaborate lab. We're happy with the results and the knowledge we got from it, and looking forward to learning more about VS Code.