Lab Session 2: Implementing class relations
21414-Object Oriented Programming

1 Introduction

We are going to implement the University management application that we designed in Seminar 2. The main part of the lab session consists in implementing the classes that are part of the project, as well as the relations that exist between these classes. Optionally, you may also implement some of the queries that were mentioned in Seminar 2 for extra credit.

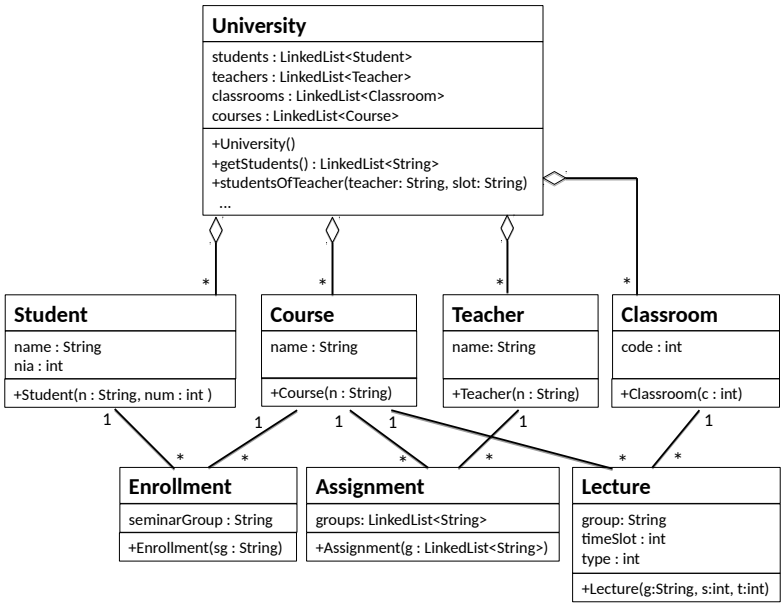
To help you get started we have provided you with a zip file called `P2.zip`. This zip file contains the definition of a Java class `Utility` as well as a directory `src` that contains the project XML files. The class `Utility` contains a static method `readXML` for parsing XML files, and a generic static method `getObject`. You should include these files in your Netbeans project.

On the next page you will find a sample design for the University management application. The three classes `Enrollment`, `Assignment` and `Lecture` are *association classes* whose purpose is to store data that is generated as part of a relation between two other classes. For example, the class `Enrollment` ("matrícula") corresponds to the relation between `Student` and `Course` and contains the seminar group number assigned to the student when enrolling in the course. The class `Assignment` corresponds to the relation between `Teacher` and `Course` and contains the list of groups assigned to the teacher for that course. Finally, the class `Lecture` corresponds to the relation between `Classroom` and `Course` and contains the time slot when the lecture is taught, the type of lecture and the number of the group. The class `University` is the access point to the application and contains lists of the different types of entities at the university.

2 Implementing the design

The first step is to implement all eight classes from the design, including their attributes. To access the class `LinkedList` you have to include an import statement at the top of the class, as before.

Since all the relations in the class diagram are associations (which include aggregations as a special case), each class also needs attributes that correspond to these relations. Think about exactly which these relations should be. For example, the class `Course` is associated with three other classes: `Enrollment`,



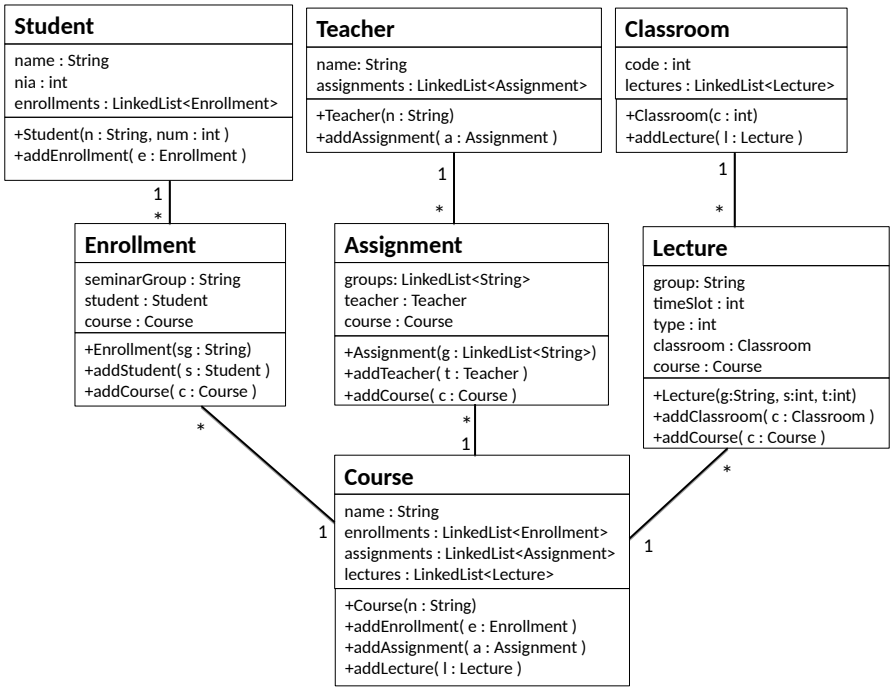
`Assignment` and `Lecture`. Add these extra attributes to the code, and also add methods that allow you to fill the content of these attributes. Usually, these attributes and methods are *implied* by the class diagram; however, on the next page you will find a modified class diagram that explicitly includes these attributes and methods.

For the four entity classes (`Student`, `Teacher`, `Classroom`, `Course`), implement each add method (`addEnrollment`, `addAssignment`, `addLecture` etc.) by adding the element to the corresponding linked list. For the association classes `Enrollment`, `Assignment` and `Lecture`, the corresponding methods add elements directly to the attribute (since the cardinality is 1).

Create a class `TestUniversity` that includes a main method. For now, the only thing the main method should do is create a new instance of `University`. Make sure that everything compiles and runs before you move on to the next part of the lab session.

3 Filling the system with the XML data

In this part we will populate our program with the entities that are defined in the XML files. All this work will be done as part of the constructor method of `University`. To parse XML files we will take advantage of the provided `Utility` class. Specifically, the following method calls parse the XML files of


Student
name : String
nia : int
enrollments : LinkedList<Enrollment>
+Student(n : String, num : int)
+addEnrollment(e : Enrollment)
Teacher
name: String
assignments : LinkedList<Assignment>
+Teacher(n : String)
+addAssignment(a : Assignment)
Classroom
code : int
lectures : LinkedList<Lecture>
+Classroom(c : int)
+addLecture(l : Lecture)
Enrollment
seminarGroup : String
student : Student
course : Course
+Enrollment(sg : String)
+addStudent(s : Student)
+addCourse(c : Course)
Assignment
groups: LinkedList<String>
teacher : Teacher
course : Course
+Assignment(g : LinkedList<String>)
+addTeacher(t : Teacher)
+addCourse(c : Course)
Lecture
group: String
timeSlot : int
type : int
classroom : Classroom
course : Course
+Lecture(g:String, s:int, t:int)
+addClassroom(c : Classroom)
+addCourse(c : Course)
Course
name : String
enrollments : LinkedList<Enrollment>
assignments : LinkedList<Assignment>
lectures : LinkedList<Lecture>
+Course(n : String)
+addEnrollment(e : Enrollment)
+addAssignment(a : Assignment)
+addLecture(l : Lecture)
1
*
1
*
1
*
*
1
*
1
*
1
1

each type:

```
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" );
```

In each case, the result is a list of string arrays, so each element of the resulting list is a string array. Depending on the type of input, the following is an explanation of the content of the string array:

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

Note that each element of this array is a *string*. To convert a string to an integer value you can use the following static method:

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

First process the XML files corresponding to the four types of university entities: `Student`, `Teacher`, `Classroom` and `Course`. For each element in the corresponding list of string arrays, the idea is to create an instance of that entity and add it to the corresponding attribute list in `University`. To create an instance we use the information in the string array to initialize the attributes. The result of this step is to fill the entity lists of the `University` class with instances of each type.

As a next step, process the XML files of the association classes `Enrollment`, `Assignment` and `Lecture`. In this case, you should also create instances of the appropriate class with the help of the content of the string array. However, the `University` class has no lists for storing these instances. Instead, the idea is to mutually connect these instances with the entities that correspond to the names provided in the string array.

For this purpose, the `getObject` method of the `Utility` class returns an entity that corresponds to a description "desc" given a list of entities. If we have just created a `Lecture` instance, the following code shows how to use the `getObject` method to retrieve an instance of `Classroom` given the string in position 0 of the string array `array` and a list of classrooms `classrooms`:

```
Classroom classroom = Utility.getObject( array[0], classrooms );
```

Note that for the above method call to work, each of the four entity classes `Student`, `Teacher`, `Classroom` and `Course` need to include a method `toString` which returns a string description of the instance. This string description is simply the *name* in the case of students, teachers and courses, and the *code* in the case of classrooms. To reconvert an integer to a string you can simply concatenate the integer with the empty string:

```
String myNum = "" + 141512;
```

In the above example, we can now add the newly created instance of `Lecture` to the entity using the method `addLecture` of `Classroom`, and we can add the entity to the instance using the method `addClassroom` of `Lecture`. Again, after adding all the necessary code to process all XML files and add instances, everything should compile and run fine.

3.1 Methods that return lists of entities

You will now implement methods of `University` that returns the lists of university entities: `getStudents`, `getTeachers`, `getClassrooms` and `getCourses`. These methods should all return lists of `strings`. To convert a list of entities to a list of strings you can use the method `toString` of Utility.

Once the methods are done, you can now add calls to these methods in the main program (in `TestUniversity`) to make sure that all entities are created properly:

```
System.out.println( university.getStudents() );
```

4 Queries

What remains is to implement some of the queries that were described in Seminar 2:

- CoursesOfStudent: obtain the courses of a student
- TeachersOfCourse: obtain the courses of a teacher
- CoursesOfClassroom: obtain the courses that are given in a classroom
- StudentsOfTeacher: obtain the students of a teacher and a classroom
- ClassroomOfTeacher: obtain the classroom given the teacher and a timeslot
- TeacherOfStudent: obtain the teacher of a student given a course and a type of class
- ClassroomOfStudent: obtain the classroom of a student and a time slot

- TeacherOfClassroom: obtain the teacher given the classroom and a time slot
- StudentsOfClassroom: obtain the students given the classroom and a time slot

These queries should be implemented as additional methods of `University`, for example:

```
public LinkedList< String > coursesOfStudent( String student ) {
```

As before, you can use the method `getObject` of `Utility` to retrieve an instance of the appropriate entity. You will then have to add methods to that type of entity for implementing the query (e.g. a method `getCourses` in the `Student` class).

With the help of these instructions, implement the first 2 queries in the above list. Optionally you can implement all of them for extra credit. Once a query has been implemented you can test it in the main program:

```
System.out.println( university.coursesOfStudent( "Ron Weasley" ) );
```