LAB2: DOCUMENTATION

Maria Beili Mena Dorado (u199138)
Mireia Pou Oliveras (u198721)

1. Description of the problem

Aquesta segona pràctica consisteix en crear la simulació d'una universitat que conté alumnes, professors, cursos i classes. Tots aquests, es llegeixen d'uns fitxers xml (proveïts amb el codi). Un cop implementats els diferents mètodes i classes, s'hauria de visualitzar a la terminal els noms dels professors i alumnes, i també les aules i assignatures. A més a més, també s'hauria de visualitzar els cursos d'un professor escollit, d'un alumne escollit i una classe escollida.

Per poder fer la tasca hem hagut d'implementar 9 classes amb els seus diferents mètodes:

- La classe "principal": University → és el punt d'accés i conté llistes dels diferents tipus d'entitats de la universitat. Hem hagut de crear el constructor *University()* amb les instàncies de cada classe entitat i tots els procediments per a poder llegir i agafar la informació dels xml. També amb les classes associades de les funcions que les relacionen amb les entitats. A més de les funcions *get()* per tenir les llistes de string de Students, Teacher, Classrooms i Courses. Finalment, també hem implementat les funcions de les Queries aquí: *CoursesofStudent()*, *TeachesrofCourse()*…

Constructor University()

```
public University() {
    studentList = new LinkedList<Student>();
    teacherList = new LinkedList<Teacher>();
    classroomList = new LinkedList<Classroom>();
    courseList = new LinkedList<Course>();

    String name;
    int nia;

    LinkedList<String[]> students = Utility.readXML(type: "student");
    for (String[] array: students) {
        name = array[0];
        nia = Integer.parseInt(array[1]);
        Student student = new Student(name, nia);
        studentList.add(student);
    }
```

A dintre del constructor University() ho hem separat en diferents apartats.

En el primer apartat, hem creat llistes de cada tipus (per després omplir-les). En el segon apartat, hem implementat el codi per omplir les llistes. Finalment, en el tercer apartat, creem i omplim les llistes de les classes que associen les funcions (lectures,assignments,enrollment).

A la imatge de dalt es pot veure com omplim la llista dels estudiants. Per a fer-ho utilitzem Utility.readXML per llegir el fitxer que conté els estudiants i els guardem en una llista. Per a cada estudiant, l'inicialitzem amb el seu constructor (fem una instància i el creem) i l'afegim a la llista d'estudiants creada al primer apartat. Aquest procediment l'hem fet servir per a les 4 classes diferents (student, course, teacher, classroom).

```
LinkedList<String[]> lectures = Utility.readXML(type: "lecture");
for (String[] array: lectures) {
    Classroom classroom = Utility.getObject(array[0], classroomList);
    Course course = Utility.getObject(array[1], courseList);
    Lecture lecture = new Lecture(array[4], Integer.parseInt(array[2]), Integer.parseInt(array[3]));

    lecture.addClassroom(classroom);
    lecture.addCourse(course);
    classroom.addLecture(lecture);
    course.addLecture(lecture);
}
```

A la imatge del costat hi ha el codi del tercer apartat del constructor. Només hi ha el de lectures, però hem fet el mateix per les altres dues classes. Creem una llista, igual que abans, llegint els fitxers xml amb Utility.xml. Per a cada lecture, la inicialitzem amb el constructor. En comptes

d'afegir-la a una llista (com les 4 classes principals), el que fem és afegir-la a les classes que associa així com afegir les classes a la lecture.

Funcions get()

```
public LinkedList<String> getStudents() {
    return Utility.toString(studentList);
}

public LinkedList<String> getTeachers() {
    return Utility.toString(teacherList);
}
```

Per a fer les funcions get() simplement hem retornat la llista que conté la classe que volem agafar passant-la a String. Per a fer-ho utilitzem la funció Utility.toString().

!!Les Queries estan explicades al punt 3!!

- Les classes entitats amb els seus corresponents atributs i constructor. També hem implementat els mètodes *add* que relacionen cada entitat amb les classes associades afegint l'element a la llista enllaçada corresponent.
    1. Classroom: *addLecture( ).*
    2. Course: *addEnrollment( ), addAssignment( ) i addLecture( ).* A més *getStudents() → Querie*
    3. Students: *addEnrollment( ).* A més *getCourses() → Querie*
    4. Teacher: *addAssignment( ).* A més *getCourses() → Querie*

Finalment, també hem implementat un mètode a cada un d'ells anomenat "toString()" que retorna una descripció de cadena de la instància.

Mètodes add() - classes entitats

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

En aquest cas, els mètodes add consisteixen en afegir un element a una llista. Per a fer-ho, simplement hem fet servir llista.add(element)

- Les classes que associen les entitats amb els seus atributs i constructors corresponents, així com els mètodes add que ajuden a la relació.
    5. Assignment: *addTeacher( ) i addCourse ( ).* A més *getCourse() → Querie*.
    6. Enrollment: *addStudent( ) i addCourse( ).* A més *getCourse() → Querie*.
    7. Lecture: *addClassroom( ) i addCourse( ).* A més *getCourse() → Querie*.

Mètodes add() - classes associades

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

En aquest cas, els mètodes add consisteixen en afegir un element a un atribut. Per a fer-ho, simplement hem igualat l'atribut al paràmetre.

- TestUniversity( ) on es crea la instància de University i on fem els prints per comprovar que s'han llegit bé els fitxers i que les funcions fan el seu propòsit correctament.

2. Alternative solution discussed

- University constructor

El constructor de University - University() - és molt llarg i conté moltes operacions. Vam tenir problemes particularment a la part d'agafar classrooms. A l'enunciat posa que el codi d'una classe és un "int", cosa que té totalment sentit perquè és un nombre. El que ens va passar va ser que als arxius XML la classe estava identificada amb nombres com, per exemple, "10.101". Java no llegia bé aquests nombres i a l'hora d'imprimir les classes ens donava error. Vam intentar solucionar-ho de moltes formes, però al final vam arribar a la conclusió que el més fàcil era posar l'atribut Codi de la Classroom com una "String" i treballar d'aquesta manera.

```
LinkedList<String[]> classrooms = Utility.readXML(type: "classroom");
for (String[] array: classrooms) {
    Classroom classroom = new Classroom(array[0]);
    classroomList.add(classroom);
}
```

A més a més, en el mateix constructor, també vam tenir problemes en un altre lloc. A l'hora de llegir el fitxer assignments ens vam adonar que contenia els diferents grups, però que no els podíem obtenir en un bucle finit perquè en cada cas el nombre de grups era diferent. Per tant, després d'intentar fer-ho de diverses formes, vam haver de crear una linkedlist dels grups i anar-los afegint fins que no n'hi hagués més.

```
LinkedList<String[]> assignments = Utility.readXML(type: "assignment");
for (String[] array: assignments) {
    Teacher teacher = Utility.getObject(array[0], teacherList);
    Course course = Utility.getObject(array[1], courseList);
    LinkedList<String> groupList = new LinkedList<String>();
    for (int i=2; i< array.length; i++) {
        groupList.add(array[i]);
    }
    Assignment assignment = new Assignment(groupList);

    assignment.addTeacher(teacher);
    assignment.addCourse(course);
    course.addAssignment(assignment);
    teacher.addAssignment(assignment);
}
```

- Queries (obligatòries)

A les dues Queries obligatòries (explicades al punt 3!!) ens vam trobar amb un altre dubte. El primer cop que vam implementar les funcions, aquestes no funcionaven correctament. A banda dels errors que hi poguessin haver, ens vam adonar que estàvem retornant una llista d'un tipus específic (és a dir "course" en el cas de students i teachers) i no una llista de String. Per aquesta raó, no imprimia bé el print que hi ha a TestUniversity(). Per arreglar-ho es podia fer de dues formes vàlides: o bé a l'hora d'agafar el course d'enrollment/ assignment el retornavem directament com una String, o bé el retornàvem de tipus "Course" i després passàvem la llista a String. Nosaltres ens vam decidir per la segona opció:

```
public LinkedList<String> getCourses(){
    int length = enrollmentList.size();
    LinkedList<Course> courses = new LinkedList<Course>();
    for (int i=0; i<length; i++) {
        Course course = enrollmentList.get(i).getCourse();
        courses.add(course);
    }
    return Utility.toString(courses);
}
```

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

A part d'això, també havíem pensat de fer les queries obligatòries de forma diferent. A classe hem vist que si un atribut en comptes de ser "private" fos "protected" aleshores es podria veure des de classes relacionades. Si ho haguéssim implementat d'aquesta forma, ens hauríem estalviat fer els getCourse() d'enrollment, lecture i assignment. Tot i això, vam decidir que fent les funcions seria més clar i més fàcil.

3. Conclusion

En resum i mirant-la globalment, creiem que la pràctica ha estat ben implementada. Els mètodes i els atributs han estat fets tal com constava a l'enunciat i el test ha tingut resultats positius.

És cert i en som conscients, però, que no hem tingut en compte cap mena d'error a l'hora de l'input. És a dir, si una funció necessita una "Classroom" i li passem un "Int" la pràctica donarà error i ja està, no hi haurà cap opció per canviar-ho ni al codi s'ha implementat res per evitar-ho.

Queries

Abans d'acabar la conclusió volíem parlar de les Queries que s'han hagut d'implementar. Com són mètodes addicionals (exceptuant els 2 primers) creiem que necessiten un apartat dedicat.

1. CoursesOfStudent: obtain the courses of a student

Per a obtenir els cursos d'un estudiant hem hagut d'implementar 3 mètodes diferents. En primer lloc, hem creat una funció a University() que passant-li l'estudiant escollit com una String, el busca a la llista d'estudiants i el retorna. Un cop ja tenim l'estudiant, crida a la funció per buscar els seus cursos.

```
public LinkedList<String> CoursesOfStudent(String student) {
    return Utility.getObject(student, studentList).getCourses();
}
```

Així doncs, en segon lloc hem implementat la funció getCourses() a Student. Aquesta es basa en els "enrollments" de l'estudiant, ja que connecten els estudiants i els cursos. Per tant, dintre la funció creem una llista de cursos i, llegint la llista de "enrollments" de l'estudiant, anem afegint-hi cursos. Finalment, convertim la llista a String i la retornem.

```
public LinkedList<String> getCourses(){
    int length = enrollmentList.size();
    LinkedList<Course> courses = new LinkedList<Course>();
    for (int i=0; i<length; i++) {
        Course course = enrollmentList.get(i).getCourse();
        courses.add(course);
    }
    return Utility.toString(courses);
}
```

Per a poder fer tot això, s'ha hagut de crear una última funció a enrollment: getCourse(), que simplement retorna el curs.

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

2. TeachersOfCourse: obtain the courses of a teacher

Per a obtenir els cursos d'un professor, hem fet servir exactament la mateixa estratègia que amb "CoursesOfStudent". En comptes de fer servir enrollment com a pont entre les dues classes, però, hem fet servir "assignment". Deixem aquí les tres funcions creades:

```
public LinkedList<String> TeachersOfCourse(String teacher){
    return Utility.getObject(teacher, teacherList).getCourses();
}
```

A university().

```
public LinkedList<String> getCourses(){
    int length = assignmentsList.size();
    LinkedList<Course> courses = new LinkedList<Course>();
    for (int i=0; i<length; i++) {
        Course course = assignmentsList.get(i).getCourse();
        courses.add(course);
    }
    return Utility.toString(courses);
}
```

A teacher().

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

A assignment().

3. CoursesOfClassroom: obtain the courses that are given in a classroom

Per obtenir els cursos donada una classe també hem fet servir la mateixa estratègia que en els dos anteriors. En aquest cas, però, la classe que relaciona classroom i course és "Lecture". Deixem aquí les tres funcions creades:

```
public LinkedList<String> CoursesofClassroom(String classroom){
    return Utility.getObject(classroom, classroomList).getCourses();
}
```

A university().

```
public LinkedList<String> getCourses(){
    int length = lectureList.size();
    LinkedList<Course> courses = new LinkedList<Course>();
    for (int i=0; i<length; i++) {
        Course course = lectureList.get(i).getCourse();
        if (!courses.contains(course)) {
            courses.add(course);
        }
    }
    return Utility.toString(courses);
}
```

A classroom().

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

A Lecture().