# PRACTICA 2 OOP

## 1. INTRODUCCIÓN

La práctica 2 de Programación Orientada a Objetos consiste en la creación y desarrollo de varias clases, entre ellas, Student, Teacher, Classroom, Course, Enrollment, Assignment , Lecture y University. Cuando se relacionan e implementan de forma eficiente y adecuada, respetando en todo momento los tipos de relaciones que existen entre ellas, crean una aplicación de administración de una Universidad. Para todas las clases previamente mencionadas deberemos crear e inicializar todos los métodos necesarios para que las clases se conecten entre ellas y tengan una funcionalidad en el programa. Para ello, nos hemos ayudado con el Lenguaje Unificado de Modelado (UML) que creamos en el seminario 2. Con el UML que creamos, podemos ver las relaciones que existen entre las clases y trabajar con un modelado visual y sintácticamente rico para la creación del programa.

A partir de aquí, empezamos a desarrollar nuestra aplicación creando los diversos métodos y atributos a todas las clases. Nuestra clase principal es Universidad ya que es el punto de acceso a la aplicación y contiene listas de los diferentes tipos de entidades en la universidad. Aquí vemos cómo hemos creado estas listas de las entidades en nuestra clase Universidad:

```
public class University {

    //atributes

    private LinkedList<Student> students;
    private LinkedList<Teacher> teachers;
    private LinkedList<Classroom> classrooms;
    private LinkedList<Course> courses;
```

A lo que los métodos se refiere, si indagamos en el programa veremos cómo a partir de comentarios de texto y anotaciones se definen y complementan las técnicas e implementación de nuestro código. Para cada clase deberemos crear el método constructor ya que es un método especial para crear e inicializar un objeto creado a partir de una clase y luego deberemos crear los métodos pertinentes a cada clase. A continuación vemos un ejemplo de la clase Students:

```java
//Método Contructor donde inicializamos todas las variables
public Student(String n, int num){
    this.name = n;
    this.nia = num;
    this.enrollments = new LinkedList<Enrollment>();
}

//Método para añadir una inscripción a "student"
public void addEnrollment(Enrollment e){
    this.enrollments.add(e);
}

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

Como podemos ver tenemos el método constructor donde tiene como parámetros el nombre y el nia del estudiante, y aparte tiene la creación de una lista enlazada de tipo enrollment. Seguidamente tenemos los métodos *addEnrollment* y *toString* que nos sirven para hacer que la aplicación funcione y se relacionen con otras clases.

## 2. DESCRIPCIÓN DE POSIBLES ALTERNATIVAS Y RAZONAMIENTO DE NUESTRA DECISIÓN

---

La principal duda que tuvimos al empezar a crear nuestro programa fue si la utilización de las *LinkedList* era la mejor estructura de datos que podríamos utilizar. Como alternativa teníamos pensado utilizar arrays. Sin embargo, pensamos que al utilizar *LinkedLists* tenemos flexibilidad de tamaño y al no saber el tamaño de las clases en nuestro programa utilizar *LinkedLists* es la mejor opción. Así mismo, lo vemos cuando creamos nuestra clase principal University donde nuestros atributos son 4 listas enlazadas de tipo estudiante, profesor, clase y curso. O también cuando creamos cada una de las clases que forman nuestro programa dónde están relacionadas con otra clase, como por ejemplo la clase *teacher* que tiene una lista de *Assignments:*

```java
public class Teacher {
    //Atributes

    private String name;
    private LinkedList<Assignment> assignments;
```

Otra duda que teníamos cuando estábamos programando, era cómo íbamos a generar el código para diferenciar las diferentes relaciones que existían entre las clases. Podemos diferenciar dos grupos de clases donde por un lado se encuentran las clases de agregación

a la clase universidad, que son *Students, Teachers, Courses and Classrooms,* y por otro lado tenemos las clases que son de tipo asociación como *Enrollment, Assignment y Lecture.* Todas estas clases tienen que llamarse en nuestra clase principal *University* para ir añadiendo las *LinkedLists.* La única manera que vemos viable para crear estas relaciones entre clases son las que hemos acabado implementado y las explicamos a continuación.

Para todas aquellas clases que mantienen una relación de agregación simplemente lo que hacemos es mediante un *for* recorrer toda la lista enlazada de la clase para luego añadirla a una variable del tipo de la clase, con los atributos necesarios, para luego mediante el método .add añadir los valores a la nueva *LinkedList* que hemos creado. Véase un ejemplo:

```
public University(){

    students = new LinkedList<Student>();
    teachers = new LinkedList<Teacher>();
    classrooms = new LinkedList<Classroom>();
    courses = new LinkedList<Course>();

    LinkedList<String[]> _students = Utility.readXML( type: "student" );

    for(String[] array : _students){
        int x = Integer.parseInt(array[1]);
        Student student = new Student(array[0],x);
        this.students.add(student);
    }
```

Como podemos ver, creamos una *LinkedList* de tipo *Student* y luego leemos el fichero xml con todos los estudiantes. Seguidamente hacemos este for para recorrer todos los estudiantes y creamos una variable *student* de tipo Student donde agregamos los atributos que tiene *student*. Seguidamente hacemos el método .add con el *student* creado.

Para el otro grupo de relaciones entre clases, el grupo de asociación, debemos hacer otro proceso.

```
LinkedList< String[] > _lectures = Utility.readXML( type: "lecture" );

for(String[] array : _lectures){

    Classroom cl = Utility.getObject(array[0], classrooms);
    Course c = Utility.getObject(array[1], courses);

    int x = Integer.parseInt(array[3]);
    int y = Integer.parseInt(array[4]);
    Lecture lecture = new Lecture(array[2],x,y);

    lecture.addClassroom(cl);
    lecture.addCourse(c);

    cl.addLecture(lecture);
    c.addLecture(lecture);

}
```

En este caso debemos utilizar el método getObject para verificar que el valor del documento txt coincide el elemento de la *LinkedList* para devolver el elemento de la *LinkedList*. Seguidamente hacemos lo mismo que lo que habíamos hecho con las relaciones de agregación con los métodos *.addClassroom* y *.addLecture* previamente creados.

## 3. CONCLUSIÓN

---

Una vez terminado el programa estamos satisfechos con el resultado obtenido. Los puntos donde hemos tenido algún problema han sido cuando no nos funcionaba el programa ya que en los métodos constructores de cada clase no creábamos las listas enlazadas para cada clase que estuviera relacionada y eso hacía que no obtuviéramos ningún valor al ejecutar el código. Una vez creadas estas listas enlazadas ya nos funcionó correctamente el programa. Luego, para la implementación de las queries nos centramos principalmente en la correcta interpretación del esquema de clases UML que se daba en el enunciado de la práctica. De ese modo, pudimos resolver siete queries logrando relacionar clases para llegar a la clase, profesor, alumno… que requería la función.

En conclusión, llegados a este punto damos la práctica por concluida conscientes de que, durante el desarrollo de ésta, hemos logrado mejorar en el entorno ‘java’ y a su vez familiarizarnos un poco más con el paradigma de programación orientado a objetos el cual hace unas pocas semanas apenas éramos conscientes de su existencia.