Lab 1: Level 1 activities

prof. dr. Irma Ravkic

This activity level is for those who wish to brush up some basic concepts of object-oriented programming (classes & interfaces) + do some new things with data structures. When you finish your project you will share it with a person I designate to you. That person should be able to compile your code and run your driver code.

Lab-Level-1-1 OOP principles: classes and objects

Create a package called *students* and in it a class called *Student* (in Student.java file) representing students having the following properties: *student number*, *name*, *surname* and *date of birth.* For the date of birth type use *LocalDate* class from *java.util.*

- Each object of class *Student* needs a way to be created. Write a constructor that will create student objects and its provided with all the necessary information above.
- We need a way to "ask" each Student object what its properties are. Write the necessary getters (you don't need to write setters since all the fields should be final - no changes in students records should be allowed, right?).
- We often need to print objects in a nice format. Override the *toString()* method from te *Object* class in order to print the students' info. For example if we have student called *A*, with surname *B*, date of birth *01/01/1990* and *studentID* = 1. Your toString() method should print the info as: Student -> Name = A, Surname = B, Date of Birth = 01/01/1990, Student ID = 1.

After you finished creating this, create a new class called *StudentDriver* in package called *apps* and create two student objects there. Note: LocalDate is rather peculiar in the updated Java 8 API. To create a new LocalDate object you do

```java
LocalDate.of(int year, int month, int day);
```

Please check: https://docs.oracle.com/javase/8/docs/api/java/time/LocalDate.html

Lab-Level-1-2 Inheritance: UndergraduateStudent and PhDStudent class

1. Create two additional classes called *UndergraduateStudent* and *PhDStudent*. These two classes should inherit from the class *Student* because each of these **are** students. Each class should have additional fields and methods:

   - *UndergraduateStudent* has the following:
     - field of type *String* called major
     - a method called *chooseMajor* which takes one input parameter: *String* denoting a major.
   - *PhDStudent* has the following:
     - field *defenceDate* of Java's *LocalDate* type.
     - field *enrollmentDate* of Java's *LocalDate* type.
     - field *phDdurationYears* of denoting the number of years a PhD should take.

     Your constructor takes *enrollmentDate* and *phDdurationYears*. You can assume the enrollment date is the day the *PhDStudent* object is created! The class also has the following method:

     - *scheduleDefence* which takes a proposed *LocalDate* for the defence and returns true/false denoting whether scheduling is succesfull or not. The method will return true if the proposed defence date is more than *phDdurationYears* years later than the enrollment date. Otherwise it returns false. If the result is true, the method sets the defence date for the student.

     To calculate the difference between two dates in years use this command in the appropriate method:

     ```
     long years = java.time.temporal.ChronoUnit.
     YEARS.between(this.enrollmentDate, defenceDate);
     ```

2. In your *StudentDriver* create two objects each student class, and print their info. (You can modify the two students you created in the previous step)

Lab-Level-1-3 Method Overriding: toString() in subclasses

When you print objects of each of the three classes introduced above, what is printed starts with "Student class ->". Why? Well because your toString() method is only defined in the superclass *Student*. If you want for each class to print its own type, you need to **override** the *toString()* method in subclasses. This means that for an Undergrade student object your code should print:

```
Undergrad Student -$>$ Name = A, Surname = B,
Date of Birth = 01/01/1990, Student ID = 1
```

and for PhD student should print:

```
PhD Student -$>$ Name = A, Surname = B,
Date of Birth = 01/01/1990, Student ID = 1
```

Lab-Level-1-4 StudentRecord class in package records

Create a class called *StudentRecord* in a new package called *records* that will contain the following information: an instance member of *Student* class, and an integer denoting a grade. Create all necessary constructors and getters and setters.

Lab-Level-1-5 Interfaces: Course Management

Now, think about how students can enroll into classes. Obviously, each course has some common behaviour. For example, if you have an *Online Course* or *Directed Study Course*, you want to make sure that they all have the same common behaviour of a course. For that reason, create *CourseManagement* interface in a new *management* package that will have the following methods:

- boolean enroll(Student st);
- boolean drop(Student st);
- double calculateClassAverage();

Lab-Level-1-6 Composition class: Course

Create a new class called *Course* in *management* package that implements interface *CourseManagement* - see above. Let this class have the following instance variables:

- a **list** keeping the *StudentRecord(s)* of already enrolled students
- one **queue** (you can use a java.util queue for now) representing the waiting list ofr *StudentRecords*
- a variable representing course capacity. Since there is a waiting queue for each course, this also means that each course should have a limited *capacity*.

Then do the following:

1. Create a constructor for this class.
2. Implement the method of enrolling the students. If the capacity is reached add the student record to the waiting list! Return false if the student is added to the waiting list, otherwise return true.

3. Implement the method of dropping the students. When a student is dropped (removed from the enrolled list) return true and if there is a student in the waiting list, enroll the student first in the queue. Hint: To drop a student from the enrolled list you need *equals* implemented in your *StudentRecord* and *Student* class (content based access - Chapter 5). Consider two *StudentRecord*s equal if their *Student*s are equal, and two *Student*s are equal if their student numbers are equal. Note: when you are comparing two records, make sure you're comparing two students in those records.

In parallel, add your students to a course and try out your methods to see if equals works!

Lab-Level-1-7 UML Diagram

If you feel a UML diagram of this project might help you better check it out in Figure Lab-Level-1-7.

Lab-Level-1-8 Just think about it

Now you can think how you can make this code slightly better. Think that *StudentRecord* only corresponds to a *Student*. But maybe a *StudentRecord* should correspond to both *Student* and *Course* because of grade. The *StudentRecord* can only contain one grade. But what if the student takes multiple courses? Also, we say that two records are the same if their students are the same: so if a student has two different records with A and B grade then those two records would be considered the same and could lead to a bug.

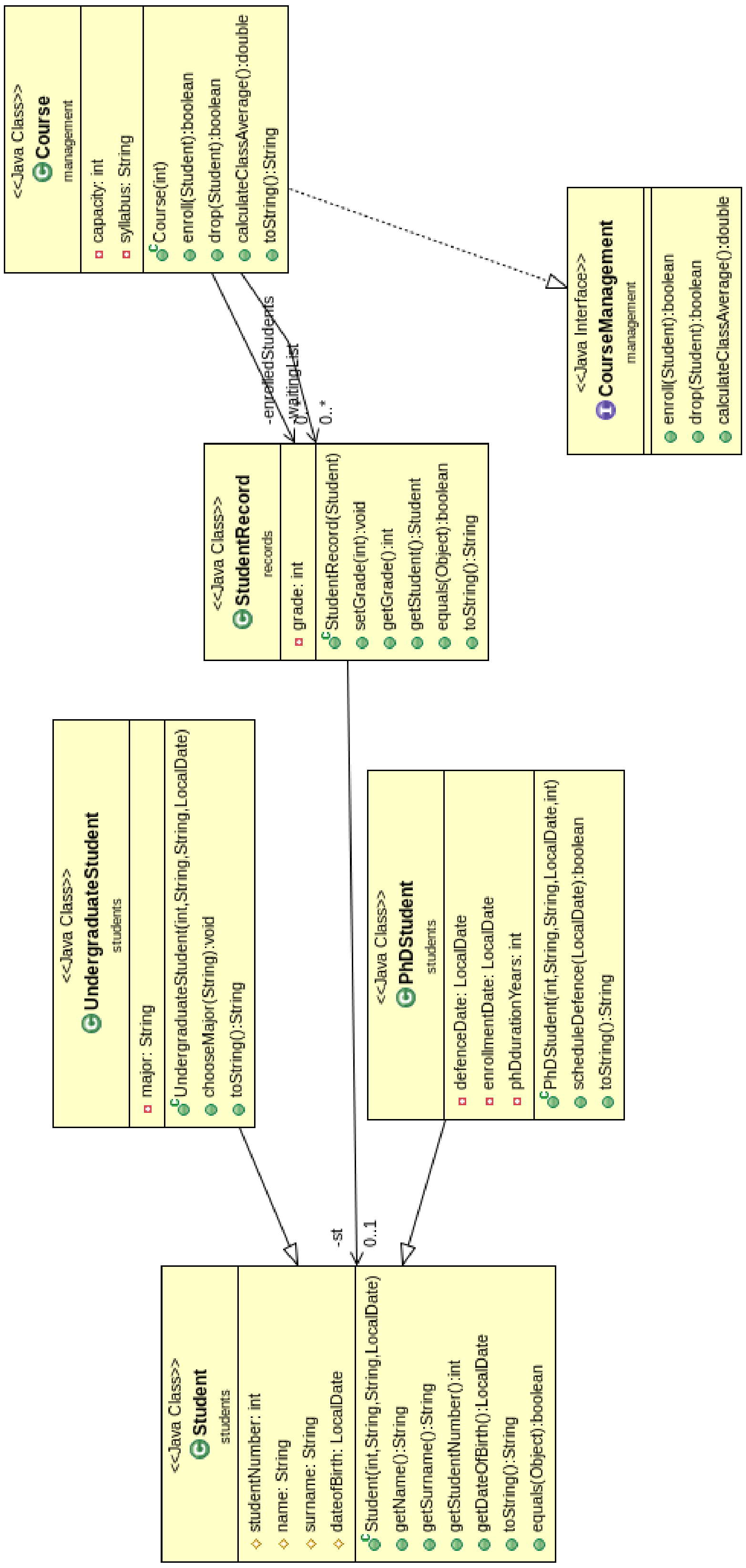

Figure 1: UML Diagram for this project