Department Mathematik/Informatik, Abteilung Informatik
Software & System Engineering Group
Software Technology Group
Lecture Object-Oriented Software Engineering, SS 2024

Alexander Korn, Adrian Bajraktari,
Prof. Dr. Michael Felderer


# Homework 1. Object-Oriented Programming

**Hand out: 19.04.2024, 18:00**
**Hand in: 03.05.2024, 14:00**

**How to submit**: For each homework, state who participated with their full name and matriculation number (e.g., as a text file "contributors hw 1"). Remember, you are only allowed to form groups of up to 5 people. Send us the link to your repository that you created in HW0 and push all your code, diagrams, images, text,... to that repository. We will ask you to add us to your repository as maintainer so that we can push back corrections and points.

**Context**: All homework will evolve around a running example project. Each homework sheet will ask you to apply a topic from the lecture in the context of the system. Be aware that the exercises must not necessarily make a fleshed out system. You will get more information on the project in upcoming sheets. For now, the following will suffice: You work in the software department of the University of Cologne. After numerous problems, countless complaints, and several blackouts, the university decided to replace Klips 2.0 and Ilias as their two-platform solution with an integrated e-Learning platform, developed by your team. The code name for this project is Klipsias. Klipsias will combine all functionality of the previous two systems, like user management, room booking, exam management, etc. Exercises will be formulated as if you actually work in a team on the project in an agile development setting.

**If you do not understand an assignment or there seems to be a mistake or inconsistency, do not hesitate to ask us! Better sooner than too late.**

## **Exercise 1.** Object-Based Programming

You selected the following requirement from the project description during the current sprint: "*Roles of a university member should be assignable at runtime*". The further description of the requirement states: "When interacting with a university member object, client code should be able to interact with as with usual university members and additionally with the role". Your task is to implement the following:

- Create a dummy person object. It must have a firstname, lastname, and an age.

- Create a professor role object with a position kind (can be: "full professor", "honorary professor", or "Jülicher Model professor"), a level (professor levels range from "W1" to "W3"), and a chair.
- It must have two methods:
  - `getDescription()` that returns the full name and chair as a string,
  - `getInfo()` that returns a string composing of the result of `getDescription()` plus the level and kind of professorship.
- Create two roles for the "head of the examination board" and the "head of the department".
  - Both override the `getInfo()` method by calling `getInfo()` of their parent object and adding the title, respectively.
  - Both override the `getDescription()` method by first calling `getDescription()` of their parent object and then appending the title respectively.
- Write code that prints the results of calling `getInfo()` of both roles on the console.

**Hint**: Use WebStorm as IDE for JavaScript. You may need to install Node.js first.

**[4 points]**

## **Exercise 2.** Interface and Class Hierarchy

You work on the user types of the system. In your team, you decided to draft a type hierarchy first, before implementing the user management. During inspection of the old systems, you agreed that there are two major distinct types of users: Students and employees. Some functionalities should only deal with employees, some only with students, but some have to process both types. Thus, you summarize both students and employees as university members. You further decided to make use of interface design. Only interface types are used as static types in code. You therefore should now implement the following in Java:

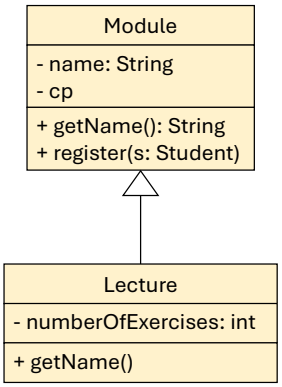
- Create classes `Employee`, `Student`, and `UniMember`.
- Create interface classes for university members, employees, and students.
- Apply reasonable subclassing (`extends` relation) among the classes and among the interfaces, respectively.
- Let the concrete classes each implement (`implements` relation) their respective interface.
- Just in: There should be distinct representations for professors, PhD's and tutors. Create classes for these types and apply reasonable subtyping to existing classes.
- Add one operation to each interface class: `String getName()` to the university member, `float getSalary()` to the employee and `String getSubject()` to the student.
- Now, you must implement these operations as methods in each implementing class respectively (your IDE will help you by generating stubs). For demonstration purposes, you can decide for yourself how each class implements the needed methods. You may introduce attributes as you wish. Of course, your implementation should make sense in the context of the system.
- Show that your code works with a short demo. Create instances for all class types and print their methods to the console.

**Hint**: Use IntelliJ as IDE for Java.

**[3.5 points]**

## **Exercise 3.** Implementation of Class-Based Languages

Describe for the following two classes and accompanying client code how instances are stored in memory, and how overriding works technically and draw a memory layout as in the lecture.




```
Module module = new Module("Dummy", 6);
Lecture oose = new Lecture("OOSE", 6, 18);
```

**[2.5 points]**

$\sum$ **10.0 points**