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

UNIVERSITÄT ZU KÖLN

Project Sheet 1. Organization and First Sprint

Hand out: 05.06.2024, 18:00
Hand in: 14.06.2024, 14:00

Disclaimer: The following describes a project that was initially not planned to happen. Because of that, all points collected during this project will add to your homework points but will not influence the minimum amount of points needed for each bonus. That is, since all four regular homework sheets give 10 points each, the amount of points needed to have 100% will be 40. Respectively, you need $40 \times 0.95 = 38$ points for the 10% bonus and $40 \times 0.8 = 32$ points for the 5% bonus. In total, you can achieve up to 65 points if you score 10 points on all four homework and 25 points on the project.

Exercise 1. Organization

In this and the following **Project Sheets**, you and your team (or just yourself if you are working alone) are going to work on a small project. The idea is to take our toy example "Klipsias" from the homework and develop a prototype of a part of the system, namely the course management, the exam and grading management, and the room booking. This project phase is completely independent of the homework, that is, you do not need to extend anything handed in previously (and it might not be 100% applicable anyway), but you are welcome and encouraged to reuse previous solutions where applicable.

The project will roughly be agile-oriented. That means:

- We, the lecturers, will be product owners, scrum masters, and clients at the same time. You will be an agile developer/team.
- The project is developed in sprints, 1 week each.
- There will be (very short) agile meetings at the end of each sprint.
- Each meeting will be a 20-minute time slot within one of the two lecture slots.
- In the meeting, we will do all three main agile meeting types in one:
    - a short sprint review at the beginning (what has been done, what not, where are problems?),
    - a short sprint retrospective in the middle (what went good or bad, both technically as well as process-wise),

  - a short sprint planning for the next sprint at the end (which user stories should be tackled in the upcoming sprint? Is there anything to clarify?).

You should be prepared for each meeting so that the short time is not wasted.

During a sprint, you are free to choose how you want to develop the project. We require only the following:

- Organize all work to do using GitHub's issue management facility.
- Work with git using the feature branch methodology (new branch for each feature/issue, pull requests, code review before merge).
- Major functionalities of the system should be tested using JUnit 5 tests. You do not need to write tests for every single method. We will individually comment on whether tests are missing in the meetings.
- We expect decent code quality: code conventions, code documentation of public methods, and easy-to-read code.

Additionally, you may apply the following techniques as you like:

- Specific programming techniques.
- Design patterns and modeling principles.
- IDE-supported refactoring.
- Test-driven Development.
- CI/CD pipelines.
- User story effort estimation.

Your next steps:

- Tell us (one of your team is enough)
  - if you stay in the groups you formed for handing in homework or if you reorganize yourselves and how.
  - if you (and your group) plan to work on the project.
  - which of the two lecture slots fits you better for the meeting. We will assign you a 20-minute time frame when we have all responses.
- Come to this week's exercise on Friday (07.06.2024) where we have exclusive time reserved for getting the project started, and to receive instant assistance in case things do not go as planned.
- Work on the project as described below in exercise 2. The end of the first sprint is the "hand-in" time and date at the top of this sheet. This is no hard deadline, but we encourage you to stick to these dates to avoid falling behind.

[0 points]

Exercise 2. This Sprint's Work

To get started, we provide you with a template. It is available in the following GitHub repository: `https://github.com/lxndio/oose-24-project`. Fork the template to get started. We recommend working on the project in the *IntelliJ* IDE. When running your project, open `localhost:8080` in your browser to see the web application. The template does not compile until you solve the following:

a) The first step in the development of Klipsias will be to implement some kind of user management. You should start by implementing the basic functionality for working with students and employees:

- Create the models `Person`, `Student`, and `Employee`. The latter two models will derive common functionality from `Person`. It should not be possible to create a `Person` object directly.

> **Hint:** `Person` is only used to derive common functionality. Thus, it must be a `@MappedSuperclass`. This also means that there is no database table for objects of this type. Other classes should have the `@Entity` and, e.g., `@Table(name = "student")` annotations. This tells Spring that the classes are models and should be stored in the database.

- All persons should have a `firstName`, a `lastName`, and an `email` address.
- Students should have a matriculation number (`matNr`), and a `studySubject`.
- Employees should have staff number (`staffNr`), and an `isProfessor` marker.
- Add getter and setter methods for all attributes.

b) Spring provides a database integration to store and load persistent objects from a database. To do this, you need to create a repository for each model class:

- Create a `StudentRepository`. Add a method to find students by their matriculation number.
- Create an `EmployeeRepository`. Add a method to find employees by their staff number.

> **Hint:** Spring allows deriving query methods in a repository. This essentially allows you to implement database functionality without actually implementing the functionality yourself. Take a look at https://docs.spring.io/spring-data/jpa/reference/jpa/query-methods.html for reference.

c) The application must be able to communicate with the provided frontend (views). This is done using controllers, which you must implement as well:

- Create a `StudentController`. Add the required repositories to the controller.

> **Hint:** Using the annotation `@Autowired` on the constructor of a controller allows you to pass repositories as parameters for the constructor. Spring will then automatically pass them to the controller.

- Add a `GET` endpoint at `/students`. The endpoint should take two request parameters: `sort_by`, and `sort_asc`.

Hint: Use the `@RequestParam` annotation to specify the value (i.e., name), default value, and required flag of the request parameters.

- The `/students` endpoint should sort all students according to the request parameters and add the sorted list of students as attribute `students` to the `model`. The parameters `sort_by` and `sort_asc` should be added to the model as attributes with the same names. *Example: The request parameters could be `sort_by=first_name, sort_asc=true`.*
- Create an `EmployeeController`. It should mimic the current functionality of the `StudentController`.

[6 points]

$\sum$ **6.0 points**