Self Study 1

Jens Tinggaard

October 11, 2023

1. The BookingSystem class contains interfaces for accessing the overall logic of the application.
2. Yes. I included the classes `Student`, `Course`, `Group`, and `Room` to represent the entities in the system. This was done, as it makes the logic of the application much more readable and easier to understand.
3. I have just mentioned the classes used, additionally, I have used the `java.util.ArrayList` class to store the entities. Furthermore, the class I have used the class `java.util.Random`, to generate IDs for the students, and `java.util.Scanner` to read the users input from stdin.
4. *In-depth description of 3 classes*
    1. `Student`: For the students, I have chosen to store the name, ID, and the courses they are enrolled in. Furthermore, all of the "primitive" classes contains a static array of all the instances of themselves. This means that it is possible to spot duplicates in the constructor. The method `addCourse(Course)` adds the given course to the students array of courses, in this implementation I also check whether the student is already enrolled in the course, or is attending more than 5 courses.
    2. `Room`: A room contains a name, as well as a boolean indicating whether it is reserved or not (as all reservations are instataneous). In case it is reserved, it also contains the a reference to the instance of the group who reserved it. Once again, I have created a static array, containing all of the instances of the class, to avoid duplicates. This also allows me to iterate them, once I need to find an available room for a group. This is shown in the following code:

```
public static boolean reserveNextAvailable(Group group) {
   for (Room room : allRooms) {
       // check if group already has reserved a room
       if (room.reservedByGroup == group) {
           return false;
       }
```

```
            // if room is not reserved, reserve it
            if (!room.reserved) {
                room.reserve(group);
                return true;
            }
        }
        // no available rooms were found
        return false;
    }
```

3. `Course`: A course only contains its subject. When creating a new instance, I check whether the course already exists, and if it does, the new instance is not added to the static `allCourses` array. For all my classes I have also overwritten the `toString()` method, to make it easier to print the information about the instances.

5. As previously mentioned, I have overwritten the default `toString()` method for all of my primitive classes. They all return some kind of id of the instance, making it easier to understand.

6. NIL

7. The program is far from fully implemented, there is no sanitizing of the input. There is no errorhandeling, meaning that if the user enters an invalid input, the program will crash. This can also happen if one tries to add a student to a course, but there are not created any courses. However, with the given time, there is no way to implement all of this, as I have already spent a lot of time on this assignment. The three most important tasks to implement, looking forward, would probably be the following:

    1. Errorhandling and sanitizing of input.
    2. Implementing the logic of reserving a room for a specific time.
    3. Implementing the administrators view, as opposed to the students view.

Running the application

I have included a `Makefile`, to make the process of compiling and running the application easier (assuming you have `make` installed). To compile and run the application, simply run `make`, and to remove the compiled `.class`-files, run `make clean`.

The Makefile is basically a wrapper for the two commands:

```
javac *.java
java Main
```