Prof. Dr.-Ing. Ralf Schenkel
Tobias Zeimetz
Trier University

Wintersemester 22/23

Exercises for the Class

Elements of Computer Science: Programming

Assignment 10

Submission of solutions until 3:30 p.m. at 23.01.2023
at `moodle.uni-trier.de`

- Every task needs to be edited in a meaningful way in order to get a point!
- Please comment your solutions, so that we can easy understand your ideas!
- If you have questions about programming or the homeworks, just ask you teachers!
- **Submission that can't be compiled are rated with 0 points!**

Exercise 1 (No Evaluation: Test class is provided)

A class `UniMember` is given, which can register university members in general. All other classes should be derived from this class. Another given class `Test` shall be used by you and extended for testing purposes to test all(!) functions and classes implemented by you.

Implement the following model:

1. A class `Student` derived from `UniMember`:
   - In addition to the data from `UniMember`, instances of type Student also have a component `matriculationNumber`, which is filled with a continuous and unique **`int`** value for each student.
   - A student is generated by a constructor that expects the same parameters as the constructor for instances of type `UniMember`. The matriculation number is assigned automatically (via an instance counter analogous to`count` in the example `K5B02E_Rectangle`).
2. A class `Staff` also derived from `UniMember`:
   - Instances of type `Staff` have additional components `room` and `phoneNumber`.
   - An employee is generated by a constructor that expects the same parameters as the constructor for instances of type `UniMember` and additionally the information `phoneNumber` and `room`. The values are inserted into the corresponding components.

3. A class `Professor` derived from `Staff`:
   - Instances of the type `Professor` additionally have an array `assistants`, in which assistants are stored. A professor can have a maximum of 10 employees.
   - The constructor for professors looks like the constructor `Staff` and fills the corresponding components.

4. Generate a class `Assistant` that derives from `Staff`:
   - Instances of type `Assistant` also have a component `supervisor` of type Professor.
   - The constructor of the class `Assistant` contains data for all necessary components and additionally sets the superior of the assistant, so it looks like this:

     ```
     public Assistant(String name,..., Professor supervisor)
     ```

     All necessary components of the instance are filled; in addition, the just generated assistant is entered into the array of the assistant of the professor, who is his boss. If the professor already has 10 assistants, an error message is displayed and the program is terminated (with `System.exit(0)`).
   - Implement a method with the signature

     ```
     public void resign()
     ```

     where a assistant quits its job, i.e. is removed from the assistants array of its boss. In addition, he should not have his own boss anymore.

5. Implement a consecutive, unique personnel number `staffNo` starting at 1000 for all employees.

6. Implement an additional method with the signature

   ```
   public boolean employed()
   ```

   that returns whether an instance of any class from the model is an employee of the university or not. All classes of the `Staff` class count as busy, with the exception of assistants after termination.

7. Implement a method with the signature

   ```
   public String toString()
   ```

   in each class that returns all components of that instance.

   For professors, the method should output a list of his assistants and their data. In addition to the first name, surname, address, telephone number, room and employee number, the first name and surname of the boss, if he has one, must also be output when outputting the assistants.

8. Test the methods and classes you have implemented in the given class `Test` with static inputs. Use (implicitly) the function `toString()` for the output. To test `employed()` and `toString()`, you should cache instances of the classes you have defined in variables of the class `UniMember`.

Watch out: Pay attention to reasonable comments in your `.java` files!

Exercise 2 (Evaluation: predefined `main` method)

This task is about a deeper understanding of interfaces. An interface is an "gateway" that makes certain functions available to a class. In order to use the functions, however, they must first be implemented by the class. The interface only provides the framework (the method declarations).

For this task, imagine you are a developer in a game company and you are currently developing a new space game. The first part of your task is to implement the `SpaceAsset` class. A `SpaceAsset` stores the coordinates of an asset in space in the form of an array (`double[] coordinates`). Since you are in space, your coordinates consist of three parts, namely x, y and z. Furthermore, a `SpaceAsset` also stores the speed that your asset (e.g. a spaceship or a rocket) has in space (`double speed`).

Next, implement a constructor of the following form:

```
SpaceAsset(double x, double y, double z),
```

which initializes the coordinate array with the given values. Furthermore, implement all getter methods and a `toString()` method matching the evaluation.

Next, derive a class `SpaceStation` from the class `SpaceAsset`. This class additionally stores the name (`String name`) of an `SpaceStation` object. There, create a constructor of the following form:

```
SpaceStation(String name, double x, double y, double z),
```

You should also extend the output of the `toString()` method with the name of the Space Station. Again, make sure that the output matches the test cases of the evaluation.

Afterwards, derive another class, named `SpaceShip`, from the class `SpaceAsset`. The `SpaceShip` class should be extended with the same aspects as the `SpaceStation` class, i.e. a name (`String name`), a constructor that takes as additional input the name of the space station and a `toString()` method that matches the given test cases.

Implement next an interface `Movable` that declares the following two methods:

- `void move(double x, double y, double z)`
- `void increaseSpeed(int speed)`

Implement for the class `SpaceShip` the interface `Movable` and implement its methods as follows:

- `void move(double x, double y, double z)`
  Shall set the coordinates of a `SpaceShip` object to the passed values of x, y and z.
- `void increaseSpeed(int speed)`
  Shall increase the speed of a `SpaceShip` object by the passed value **.**

Note: This task is intentionally vague, so you will have to deal with design decisions yourself.