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

Exercise Sheet 1. Object-Oriented Programming
Solution

Exercise 1. Roles in Object-Based Programming

a) Pull `https://github.com/AdrianBajraktari/oose24.git` to get a JavaScript template for this exercise (in `src/exercise1/E1_oop_template.js`). Extend the template as follows:

   1) Create a student object for Alice Wonderland, age 24. Implement a method `getName` that prints the first and last name like this: "Alice Wonderland".

   2) Implement the "functions" `makeTutor`, `makeStudent`, and `makePhD`. In each, first clone the existing role*X* object using the clone "function". Then, add the provided parameters (except for person) as properties to the cloned object (i.e. for student role: `matNr`, `studyProgram`).

   3) Set the parent object of each role*X* to the provided person.

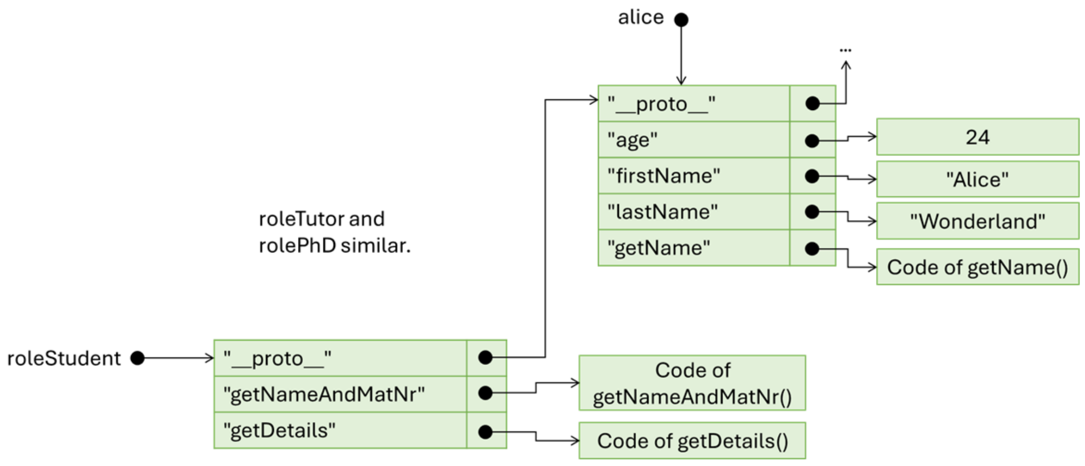
   When done correctly, the overall program should create new role*X* objects which delegate to `alice`, i.e., `alice` takes on different roles during her time at university.

b) How does the memory layout of the object structure from (a) look like? Create a rough sketch.

[0 points]

a) See example code in exercise repository.

b) The following shows the relation for `roleStudent`. The other two role objects are situated similarly.



Exercise 2. Roles in Class-Based Programming

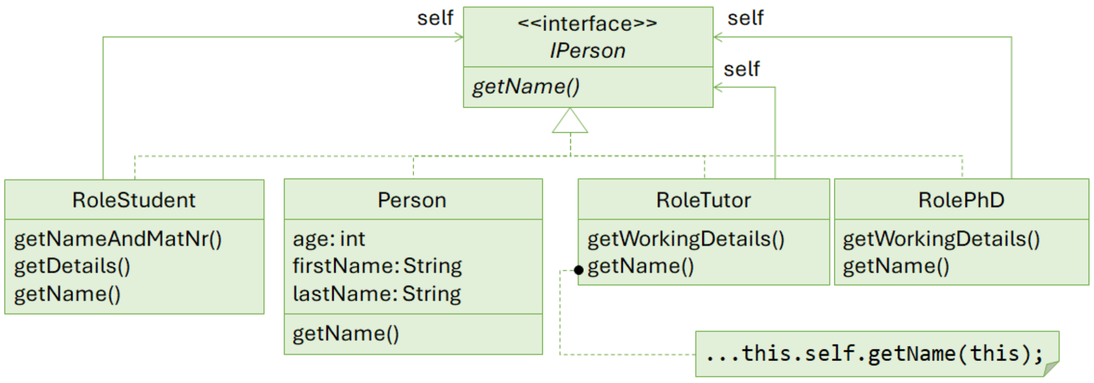
Try to mimic the functionalities realized in exercise 1 using class-based programming in Java by doing the following:

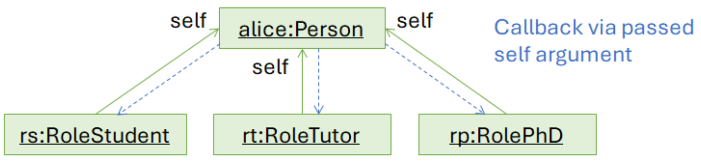
1) Create classes `Person`, `RoleStudent`, `RoleTutor`, `RolePhD`. Add all corresponding attributes and methods to them as in the JavaScript version.

2) Create an interface class `IPerson` with the method `String getName()`. Make all four classes implement the interface.

3) Java has no object inheritance. How can you realize delegation via objects at runtime? Try your own ideas. We will see "the" solution in an upcoming lecture.

[0 points]





Simulating delegation/object inheritance in class-based languages is extensive manual work with only limited similar effects. The major steps:

- Java is strongly typed. Thus, an object must know the interface of its delegatee at compile time. Therefore, all candidates for delegating objects must implement the same interface, realized by implementing the `IPerson` interface class.
- Each class (except `Person`) needs a reference (here called `self`) to the delegatee. To have more flexibility, instead of referencing the `Person` class (which would suffice to reconstruct the example from exercise 1), we instead say that the delegatee is an `IPerson`. At runtime, we substitute with one of the implementing classes' instances.
- Each method (here only `getName(...)`) in the `IPerson` interface must have an explicit argument to mimic "this". When a role object delegates a message, it passes `this` as argument to the `self` parameter.

Example code to the above diagram will be provided in the exercise repository.

$\sum$ 0.0 points