Prof. Dr. A. Poetzsch-Heffter
Mathias Weber, M.Sc.

University of Kaiserslautern
Department of Computer Science
Software Technology Group

# Advanced Aspects of Object-Oriented Programming (SS 2013)

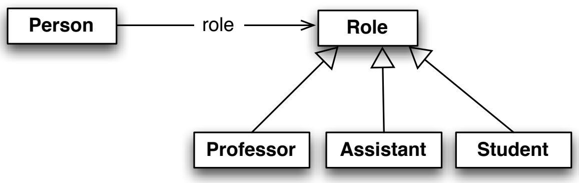
## Practice Sheet 4

Date of Issue: 29.04.13
Deadline: 07.05.13
(before the lecture as PDF via E-Mail)

## Exercise 1 University Administration System - Reloaded

The delegation pattern can be used to simulate inheritance, but it is a more general design pattern (see `http://en.wikipedia.org/wiki/Delegation_pattern`). Note, the meaning of the terms delegation and forwarding varies in the literature, each author has a slightly different notion of them.

Until now the UAS used inheritance to model the different persons at a university, look at the listing below to see a different implementation, which uses delegation to establish the link between a person and the role, it currently has at the university. In this implementation `Person`-objects delegate some calls to their `Role`-object. The figure shows the architecture of the implementation.



```java
package persons;

class Person {
  public String name;
  public Role role;

  public Person(String name) {}
  public void assignRole(Role r) {role = r;}
  public void print() {
    if (role == null)
      System.out.println("Not much known about " + name);
    else
      role.print(this);
  }


    public static void main(String... argv) {
        Person p = new Person("Max Mustermann");
        p.assignRole(new Student()); // Max starts his career
        p.assignRole(new Assistant()); // Max graduates and starts working at the university
        p.assignRole(new Professor()); // and finally he manages to become a professor

    }

}

interface Role {
    public void print (Person p);
}

class Professor implements Role {
    String room;
    String institute;
    public void print(Person p) {
        System.out.print("Professor " + p.name + "'s office is in room " + room);
    }
}

class Student implements Role {
    int reg_num;
    public void print(Person p) {
        System.out.print(p.name + " has the registration number " + reg_num);
    }
}

class Assistant implements Role {
    boolean phDStudent;
    public void print(Person p) {
        System.out.print(p.name + " is a PhD student: " + phDStudent);
    }
}
```

a) Can you think of advantages and disadvantages of the delegation based implementation compared to an inheritance based implementation? How does the scenario of the main method look like in a inheritance based system?

b) Formulate a general guideline, when to favor inheritance over delegation and vice versa.

## Exercise 2 Tiny Web Server

You can find the sources of a simple Java-based web server on the lecture's web page. *Important notice: The sources are meant to illustrate / practice concepts of the lecture. Students will refactor / redesign / extend parts of the software system, so currently existing deficiencies are intentional.*

a) Download the ZIP file, unzip it, and start the server via `ant clean compile run`. Check if everything works by requesting the URL `http://localhost:8080/public_html/HelloWorld.html` using a web browser.

b) Analyse the classes `SimpleWebServer` and `LoggableClass`. Introduce an appropriate interface and refactor the existing code of these two classes so forwarding / delegation is used instead of inheritance.

## Exercise 3 Covariance & Contravariance

a) Discuss the concept of Java's *Checked Exceptions* with respect to covariance and contravariance.

b) Which kind of guarantee is given by *Checked Exceptions*?

## Exercise 4 Generics

a) Write a generic static method `flip` which takes an object of class `Pair` (see the slides of the lecture) and flips the elements of the *given* `Pair` object. *Hint: In order to flip the elements, both need to be of the same type.*

b) What is the difference between a `Collection<?>` and a `Collection<Object>` ?

c) Explain the output of the following program:

```
public final class GenericClass<T> {
  public void overloadedMethod(Collection<?> o) {
      System.out.println("overloadedMethod␣(Collection<?>)");
   }
   public void overloadedMethod(List<Number> s) {
      System.out.println("overloadedMethod␣(List<Number>)");
   }
   public void overloadedMethod(ArrayList<Integer> i) {
      System.out.println("overloadedMethod␣(ArrayList<Integer>)");
   }

   private void method(List<T> t) {
      overloadedMethod(t); // which method is called?
   }

   public static void main(String[] args) {
      GenericClass <Integer> test = new GenericClass<Integer>();
      test.method(new ArrayList<Integer>());
   }
}
```

d) The interface `Collection<T>` contains a generic method to convert a collection into an array. The method has the signature `<T> T[] toArray(T[] a)`.

What is the purpose of the parameter `a`? Is it possible to write a method with the same behavior with the signature `<T> T[] toArray()`? Justify your answer.