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 (Hints and Comments)

Exercise 1 University Administration System - Reloaded

a) Advantages:

- Easy to change the current role.
- Easy to extend with new roles, as that does not affect any of the non-role specific parts, whereas subtypes accidentally could modify the behavior of the role independent parts unintended.
- ...

Disadvantages:

- No code for free, everything has to be written for each role
- Guarantees of the type system cannot be used anymore. E.g. no static guarantee anymore, that professors don't register for exams (every one is just a person)
- ...

The given scenario would do something like:

```
public static void main(String... argv) {
        Student s = new Student("Max␣Mustermann");
        Assistant a = new Assistant(s); // copy constructor, i.e. copy all information that remain valid
                // from the existing student object to the new assistant object
        // invalidate s, remove s from all collections, ...
        Professor p = new Professor (a); // same as for assistant
}
```

b) If the set of types/roles can be given at compile time and it will not change for an object during its lifecycle, it is usually a good idea to use subtyping and, if appropriate, inheritance. If the role of an object is likely to change, it may be better to take a delegation approach. In general, the decision depends on a) the language support, b) the used libraries (some favor delegation, some are designed with subtyping in mind) c) the knowledge of the programmer, etc. Bigger software projects usually use both, e.g. parts of the software that have strong dependencies between them may use subtyping and parts that are not integrated so much may use delegation or forwarding.

Exercise 2 Tiny Web Server

a)

b) See enclosed source. To be able to compile the complete webserver without changing the whole source, the old code was marked as deprecated but remained in the implementation. It can be removed if all classes use the delegation pattern instead of subclassing.

Exercise 3 Covariance & Contravariance

Checked Exceptions have to be caught or declared to be thrown by a method.

a) With covariant exceptions try-catch blocks catch all thrown exceptions, even if a subtype of the catched exception is thrown. In a contra- or invariant scenario, the programmer has to update his catch statement every time the exception changes (e.g. due to an update in third party code). Depending on the context a program is used in, this would be uncomfortable or even impossible.

b) Checked exceptions guarantee at compile time, that they are handled by the program. Thus they do not lead to the abortion of the program. See JLS 11.2 for more about compile time checking of exceptions and the reasons for having unchecked exceptions as well.

Exercise 4 Generics

a)
```
static <A> void flip (Pair<A,A> p) {
    A tmp = p.fst;
    p.snd = p.fst;
    p.fst = tmp;
}
```

b) `Collection<?>` can hold a collection of anything, whereas `Collection<Object>` only holds collections to objects, i.e. `Collection<?> c = new ArrayList<String>();` is valid, wheras `Collection<Object> c = new ArrayList<String>();` is not (remember that generic types are invariant).

c) The output is `overloadedMethod (Collection <?>)`, because overloading is resolved at compile time and at compile time the parameter t has type List<T>, such that the most specific method that can be applied is the one for the generic collection. (see JLS 15.12)

d) The method without the parameter `a` can be written and compiled. But whenever you try to execute it, you will get a class cast exception, except for T = Object.

```
<T> T[] toArray() {
  T[] res = (T[])(new Object[l.size()]);
  for (int i = 0; i < res.length; i++) {
    res[i] = l.get(i);
  }
  return res;
}
```

The parameter serves as a kind of type token. Either the given array can be directly used or if the provided array is not large enough it can be used to create a bigger array, using e.g.

`public static Object Array.newInstance(Class<?> componentType, int length) throws NegativeArraySizeException`