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 2015)

Practice Sheet 3 (Hints and Comments)

Exercise 1 Inheritance and Subtyping

a) "In programming language theory, subtyping or subtype polymorphism is a form of type polymorphism in which a subtype is a datatype that is related to another datatype (the supertype) by some notion of substitutability, meaning that program constructs, typically subroutines or functions, written to operate on elements of the supertype can also operate on elements of the subtype. If S is a subtype of T, the subtyping relation is often written S <: T, to mean that any term of type S can be safely used in a context where a term of type T is expected." (Source: `http://en.wikipedia.org/wiki/Subtype_polymorphism`

The main purpose of inheritance is the reuse of code in different contexts. It makes implementing subtypes easier, because the subtype implementation only needs to describe the difference from the supertype, every unchanged behavior is given by the supertype implementation. If two classes are related with an inheritance relation, they also stand in a subtype relation.

b) The output is `woof woof`. The method `bark` is a static method, and therefore is bound statically. As `woofer` and `nipper` are variables of the static type `Dog`, the implementation in class `Dog` is chosen for calls.

Note: Java allows static methods to be called either on an object or directly on a class. In both cases they are statically bound, and thus the runtime type of the called object does not influence the method choice.

c) Before the Java 1.5 Specification was released, it was not possible to tell the compiler a method should override a method in the superclass. This lead to many mistakes where methods accidentally did not override the method of the superclass but instead added an overloaded method to the subclass.

When annotating a method with `@Override` this method must override a method of the superclass. If this is not the case, the java compiler will signal an error.

§15.12.4.4 of the Java Language Specification describes how methods are invoked at runtime. The method dispatch of `x.m` for a non-static method `m` and `T` is the static type of `x` starts by looking up the method in the runtime type of `x` called `S`. If `S` has an implementation of `m` and this implementation overrides the method `T.m` than this method is called. Otherwise the lookup procedes up the class hierarchy.

d) Yes, it does. This can be checked by adding the `@Override` annotation to the method. Note: `PhDAssistant` and `Person` are in the same package and widening of the accessibility (changing the modifier from default to `public`) is not affecting the subsignature relation between the two implementations. `Assistant` does not have a print method, but this does not change overriding, it just affects how (if possible) to call the overridden method.

Exercise 2 Super-Calls

```
public class A {
    public void m() { System.out.println(''A''); }
}
public class B extends A{
    public void m() { super.m(); }
}
public class C extends B{
    public static void main(String ... arg) {
        new C().m();
    }
}
```

With static binding new C().m() results in a call to the method m of class A. With dynamic binding it would end in an endless recursion, because the supertype of the current this would be B, so the call would again be dispatched to m in B.

Exercise 3 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 4 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 5 Reflection and Annotations (optional)

a) The class `java.lang.reflect.Proxy` allows you to create dynamic proxy instances which implement a list of interfaces specified at runtime. Statically, a class implementing this list of interfaces does not need to exist.

b) See enclosed source.

c) This technique only works for interface types.