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 2 (Hints and Comments)

Exercise 1 Required and Provided Interfaces

a) Provided interface for classes

- *outside java.io*: all public methods, constructors, fields, inner classes, etc of `ObjectOutputStream` and its superclasses.
- *inside java.io*: like for classes outside java.io + protected fields, methods and constructors + fields, methods, constructors with the default modifier.
- *subclasses of ObjectOutputStream inside java.io* like all other classes inside java.io
- *subclasses of ObjectOutputStream outside java.io* like all other classes outside java.io + protected field, methods, constructors, etc.

b) To find the required interface of a class, you have to look at the code (documentation, specification) to find out, which objects you can / have to pass to the class in order to work with it. In this case, the OOS requires an OutputStream (parameter to the constructor).

Exercise 2 Introspection and Reflection

a) See enclosed source.

The class can be loaded using `Class.forName(. . .)` since the plugin jar is in the classpath. The implemented interfaces are returned by the method `Class.getInterfaces()`. The method `Class.newInstance()` can be used to create an new instance using the default constructor.

b) See enclosed source.

The method of a given name is given by `Class.getMethod(String, Class<?>. . .)`. This method can be invoked using `Method.invoke(Object, Object. . .)`.

c) No comments.

Exercise 3 Prototype-based Inheritence

a) See enclosed source.

b) The prototype `B` is changed after instantiating object `v2`. Since all instances of `B` have this method afterwards, it can be called on instance `v2`. In Java this would mean to change the class of an already existing object, which is not possible.

Exercise 4 Source Compatibility of Java-Packages

Package `util` The modification breaks source compatibility, because if someone implemented the `List` of the unmodified package, and tries to compile his source with the modified version, the compiler will complain about the not implemented method `m`.

Package `p` This modification is safe. `C` was declared final, so there cannot be any subclasses of `C` and `m` was declared `protected`, therefore the only code that can call `m` is inside `p`, the change of the parameter type is safe because it only affects the package `p` but nothing outside it. The changes from final to non-final and protected to public are safe as well, as it just widens the accessibility, which is no problem as there have not been any subclasses that may clash with the changed declarations.

For more information you may look at `http://softech.informatik.uni-kl.de/Homepage/SourceCompatibility`.