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 2 (Hints and Comments)

Exercise 1 JavaScript – Dynamic Typing

a) The problem lies in the expressions of the form `a > b > c`. With parenthesis the expression has the form `(a > b) > c`. `a > b` returns a value of type boolean. JavaScript than converts this value to a number, in this case `0` for false and `1` for true.

A static type system could have helped detecting the problem by pointing out the different types of value on the left and on the right of the `>` operator.

b) The method `Array.map` takes a function and applies this function to all elements of the array. The results are collected in a new array. The problem in this case is, that the function taken by `map` has to take three parameters: The current value, the index and the array itself. The `parseInt` function takes a string value and parses it as an integer. The function can as well take two parameters: The string and the radix of the number (e.g. 10 for decimal numbers).

JavaScript does not check the arity of the function given to `map`. Instead of complaining, the third parameter to `parseInt` is simply ignored. The result of parsing the string-array is `[1, NaN, NaN]`, since `parseInt("2", 1)` (the string "2" in the system with radix 1) it not a number. A static type system could have checked the arity of the function `parseInt`.

c) The expression `[1, 2, 3] + [4, 5, 6]` should mean we concatenate the two arrays (read lists). The `+` operator in JavaScript only operates on numbers and strings. If not both arguments of this operator are numbers, the values are converted to strings.

Strings in JavaScript can be addresses as if they where arrays of characters. So the code to sum the values in the array does also work for strings. As explained before, the arguments to `+` (or in this case `+=`) are than converted to strings.

The `-` operation only works on numbers. If one of the arguments is not of type number, it is converted to type number. The fallback is the special constant `NaN`.

The result of computing `[1, 2, 3] + [4, 5, 6]` is the string '1, 2, 34, 5, 6'. The function `sum` concatenates `0` to the front of this string, resulting in '01, 2, 34, 5, 6'. This string is converted to a number and since it does not represent a valid number, the result is `NaN`. Subtracting 1 from `NaN` yields `NaN` again.

A static type system could have found out multiple problems: The types of the arguments to the `+` operator are not correct. Even if the conversion to string would have been made, the arguments to the `-` operator are not correct and should have resulted in feedback for the programmer.

Exercise 2 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 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 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.