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

Date of Issue: 23.04.13
Deadline: **29.04.13 before 12:00**
(before the lecture as PDF via E-Mail)

Exercise 1 Required and Provided Interfaces

Download the source for the class `ObjectOutputStream` from the website of the lecture.

a) Which different *provided interfaces* does the class have? Give examples for each of them.

b) What is the *required interface* of an `ObjectOutputStream`-object?

Exercise 2 Introspection and Reflection

With the techniques of introspection and reflection, it is possible to extend Java programs with plugins.

Our convention is that the class that implements the plugin is named like the plugin itself. Plugins implement the following Java interface:

```
import java.util.List;

interface IPlugin {
   List<String> getMethodNames();
}
```

The method `getMethodNames` returns the names of the methods the plugin implements. In our reduced example, the methods have the following signature: `Object <methodname>(Object o)`.

A sample plugin is given on the lecture homepage. The plugin is named `HelloPlugin` and can be downloaded as `HelloPlugin.jar`.

The `PluginLoader` can assume that the plugin jar files are in the classpath.

Your task is to implement the class `PluginLoader`.

a) Implement a method `IPlugin load(String clazz)` that loads the plugin with the given name. Please also check that the loaded class really implements the needed interface.

b) Implement a method `Object call(IPlugin plugin, String method, Object param)` which calls the given method with the given parameter on the given plugin.

c) Call the first method of the `HelloPlugin` with your name as a parameter and print the result to console.

Exercise 3 Prototype-based Inheritence

Inheritence is not only possible in class-based languages such as Java. It can also be simulated in prototype-based languages such as JavaScript.

Consider the following example program:

```
1  function A(vara, varb)
2  {
3      this.vara = vara;
4      this.varb = varb;
5  }
6
7  A.prototype.set = function(v)
8  {
9      this.vara=v;
10 }
11
12 A.prototype.hello=function()
13 {
14     console.log("Hello␣World!");
15 }
16
17 function B(vara, varb, varc)
18 {
19     A.call(this, vara, varb);
20     this.varc = varc;
21 }
22
23 B.prototype = Object.create(A.prototype);
24
25 B.prototype.set=function(v)
26 {
27     if (v != "") {
28         this.vara=v;
29     }
30 }
31 B.prototype.get=function()
32 {
33     return this.vara;
34 }
35
36
37 var v1 = new A("test", 4);
38 var v2 = new B("test2", 5, v1);
39 v2.set("output␣please");
40 console.log(v2.get());
41
42 v2.hello();
43
44
45 var v3 = new B("something", 42, v1);
46
47 v3.get=function()
48 {
49   return "Result:␣" + B.prototype.get.call(this);
50 }
51 console.log(v3.get());
52
53
54
55 B.prototype.some_function=function()
56 {
57     console.log("Some␣function␣called.");
58 }
59 v2.some_function();
```

a) Implement the example in Java.

b) The change made in lines 55-59 is not possible in Java. Why?

Exercise 4 Source Compatibility of Java-Packages

The packages of a software implementation usually evolves is different ways. A package may be exchanged by another one, classes, fields and methods can be added or removed, and so on. For the developer of the package it is important to know, wether his changes are source compatible or not. A change breaks source compatibility, if a program exists that compiles with the unchange package version but does not compile with the changed one.

Look at the following packages. Is it safe to make the given modifications? If not, give counter examples.

```
// unmodified version
package util;

public interface List {
  public abstract List n() ;
}

// modified version
package util;

public interface List {
  public abstract List n() ;
  public abstract List m() ;
}
```

```
// unmodified version
package p;

public final class C {
  protected C m(Object v) {}
}}

// modified version
package p;

public class C {
  public C m(C v) {}
}}
```