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

Date of Issue: 05.05.15
Deadline: 12.05.15
(before the lecture as PDF via E-Mail)

Exercise 1 Inheritance and Subtyping

a) Explain the relation between inheritance and subtyping.

b) Explain the output of the following program:

```java
class Dog {
    public static void bark() {
        System.out.print("woof␣");
    }
}

class Basenji extends Dog {
    public static void bark() { }
}

public class Bark {
    public static void main(String args[]) {
        Dog woofer = new Dog();
        Dog nipper = new Basenji();
        woofer.bark();
        nipper.bark();
    }
}
```

c) Define the role and semantics of the `@Override` annotation in Java. Explain the relation between overriding and dynamic dispatch. *Hint: use the Java Language Specification*

d) Analyze the following program:

```java
package p;

public class Person {
  void print() {}
}

package q;
import p.Person;

public class Assistant extends Person {}

package p;
import q.Assistant;

public class PhDAssistant extends Assistant {
  public void print() {}
}
```

Does the `print` method in class `PhDAssistant` override the `print` method in class `Person` ? *Hint: look at the Java Language Specification in Section 8.4.8.*

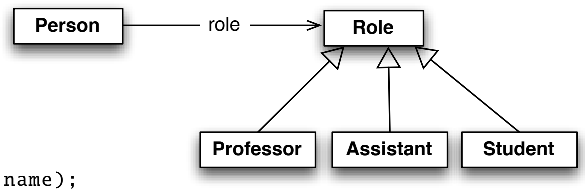
Exercise 2 Super-Calls

Write a program, that would behave differently under the assumption of dynamically bound `super`-calls than it does with statically bound `super`-calls.

Exercise 3 University Administration System - Reloaded

The delegation pattern can be used to simulate inheritance, but it is a more general design pattern (see `http://en.wikipedia.org/wiki/Delegation_pattern`). Note, the meaning of the terms delegation and forwarding varies in the literature, each author has a slightly different notion of them.

Until now the UAS used inheritance to model the different persons at a university, look at the listing below to see a different implementation, which uses delegation to establish the link between a person and the role, it currently has at the university. In this implementation `Person`-objects delegate some calls to their `Role`-object. The figure shows the architecture of the implementation.



```java
package persons;

class Person {
  public String name;
  public Role role;

  public Person(String name) {}
  public void assignRole(Role r) {role = r;}
  public void print() {
    if (role == null)
      System.out.println("Not␣much␣known␣about␣" + name);
    else
      role.print(this);
  }


    public static void main(String... argv) {
        Person p = new Person("Max␣Mustermann");
        p.assignRole(new Student());    // Max starts his career
        p.assignRole(new Assistant());  // Max graduates and starts working at the university
        p.assignRole(new Professor());  // and finally he manages to become a professor

    }

}

interface Role {
    public void print (Person p);
}

class Professor implements Role {
    String room;
    String institute;
    public void print(Person p) {
        System.out.print("Professor␣" + p.name + "'s␣office␣is␣in␣room␣" + room);
    }
}

class Student implements Role {
    int reg_num;
    public void print(Person p) {
        System.out.print(p.name + "␣has␣the␣registration␣number␣" + reg_num);
    }
}

class Assistant implements Role {
    boolean phDStudent;
    public void print(Person p) {
        System.out.print(p.name + "␣is␣a␣PhD␣student:␣" + phDStudent);
    }
}
```

a) Can you think of advantages and disadvantages of the delegation based implementation compared to an inheritance based implementation? How does the scenario of the main method look like in a inheritance based system?

b) Formulate a general guideline, when to favor inheritance over delegation and vice versa.

Exercise 4 Tiny Web Server

You can find the sources of a simple Java-based web server on the lecture's web page. *Important notice: The sources are meant to illustrate / practice concepts of the lecture. Students will refactor / redesign / extend parts of the software system, so currently existing deficiencies are intentional.*

a) Download the ZIP file, unzip it, and start the server via `ant clean compile run`. Check if everything works by requesting the URL `http://localhost:8080/public_html/HelloWorld.html` using a web browser.

b) Analyse the classes `SimpleWebServer` and `LoggableClass`. Introduce an appropriate interface and refactor the existing code of these two classes so forwarding / delegation is used instead of inheritance.

Exercise 5 Reflection and Annotations (optional)

The `Proxy` class (`java.lang.reflect.Proxy`) allows to intercept method calls.

a) Inform yourself about the `Proxy` class using the JDK documentation. Is it a *normal* JDK class? Also inform yourself about Java annotations (for example under `http://docs.oracle.com/javase/tutorial/java/annotations/`).

b) Develop a generic wrapper using the `Proxy` class, which allows, for a certain object, upon invokation of one of its methods, to check annotated method parameters for non-nullness. An example of using the wrapper could look as follows:

```java
public interface Example {
    public void print(@NonNull String s);
}

...

Example e = new Example() {
    public void print(String s) {
        System.out.println(s);
    }
};
e = (Example)Wrapper.wrap(e);
e.print("Hello"); // prints out "Hello"
e.print(null); // should throw an exception
```

The annotation type declaration is given as follows (does not need to be modified).

```java
import java.lang.annotation.*;

@Target(ElementType.PARAMETER)
@Inherited
@Retention(RetentionPolicy.RUNTIME)
public @interface NonNull {}
```

c) For which types and methods does the presented technique work?