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 4

Date of Issue: 12.05.15
Deadline: 19.05.15
(before the lecture as PDF via E-Mail)

Exercise 1 Featherweight Java

a) Use the rules given in Fig. 2 in the paper "Featherweight Java: A Minimal Core Calculus for Java and GJ" to check for correct typing of the following program.

```
class Person extends Object {
  String name;
  int age;

  Person(String name, int age) {
    super();
    this.name = name;
    this.age = age;
  }

  int getAge() {
    return this.age;
  }
}

class Student extends Person {
  int studentId;

  Student(String name, int age, int studentId) {
    super(name, age);
    this.studentId = studentId;
  }

  Person me() {
    return (Person)this;
  }
}

class Main extends Object {
  Main() {super(); }

  Person main() {
    return new Person("Berta", new Student("Bob", 22, 12345).getAge());
  }
}
```

Check if the program is type correct according to the type rules of Featherweight Java. Do a detailed proof of the type correctness of the `Main` object using the procedure shown in the lecture.

You may assume that the types `int` and `String` are built in and valid types in Featherweight Java with the same meaning and syntax as the corresponding types in Java.

b) Give a step-by-step reduction of the expression

```
new Person("Berta", new Student("Bob", 22, 12345).getAge());
```

as shown in the lecture.

c) Extend Featherweight Java with a `let` construct. The syntax should look like in the following example:

```
let x = e1 in e2
```

where `e1` and `e2` are expressions and the variable `x` can be used in `e2`.

Give the needed typing rule(s) as well as the needed reduction rule(s).

Exercise 2 Generics

a) Write a generic static method `flip` which takes an object of class `Pair` (see the slides of the lecture) and flips the elements of the *given* `Pair` object in place (without creating a new `Pair` object). *Hint: In order to flip the elements, both need to be of the same type*.

b) What is the difference between a `Collection<?>` and a `Collection<Object>` ?

c) Explain the output of the following program:

```
public final class GenericClass<T> {
   public void overloadedMethod(Collection<?> o) {
        System.out.println("overloadedMethod␣(Collection<?>)");
    }
    public void overloadedMethod(List<Number> s) {
        System.out.println("overloadedMethod␣(List<Number>)");
    }
    public void overloadedMethod(ArrayList<Integer> i) {
        System.out.println("overloadedMethod␣(ArrayList<Integer>)");
    }

    private void method(List<T> t) {
        overloadedMethod(t); // which method is called?
    }

    public static void main(String[] args) {
        GenericClass <Integer> test = new GenericClass<Integer>();
        test.method(new ArrayList<Integer>());
    }
}
```

d) The interface `Collection<T>` contains a generic method to convert a collection into an array. The method has the signature `<T> T[] toArray(T[] a)`.

What is the purpose of the parameter `a`? Is it possible to write a method with the same behavior with the signature `<T> T[] toArray()`? Justify your answer.