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 (Hints and Comments)

Exercise 1 Featherweight Java

a) `Person`-class:

- $fields$(Object) = •
- $fields$(Person) = String name, int age
- $mtype$(m, Person) = • → int
- using T-Class: K = `Person(String name, int age){super(); this.name = name; this.age = age;}` ✓
- Method getAge() using T-Method: this: Person ⊢ this.age : int and int <: int (because of rule T-Field) ✓

`Student`-class:

- $fields$(Person) = String name, int age
- $fileds$(Student) = int studentId
- using T-Class: K = `Student(String name, int age, int studentId){super(); ...}` ✓
- Method me() using T-Method: this: Student ⊢ (Person)this : Person (because of rule T-UCast) ✓

```
goal: class Main extends Object {...} OK
-------------------------------------------------
apply (T-Class)
subgoal: K = Main(){super();} ✓
subgoal: fields(Object) = [] ✓

subgoal: Person main(){return new Person("Berta", new Student("Bob", 22, 12345).getAge());}
         OK in Person
-------------------------------------------------
apply (T-Method)
subgoal: Person <: Person ✓
subgoal: class Main extends Object {...} ✓
subgoal: if mtype(main, Object) = [] → D then [] = [] and Person = D ✓
(because method main is not in Object)

subgoal: this : Main ⊢ new Person("Berta", new Student("Bob", 22, 12345).getAge()) : Person
-------------------------------------------------
apply (T-New)
subgoal: fields(Person) = [String name, int age] ✓
subgoal: String <: String ✓
subgoal: int <: int ✓
subgoal: this : Main ⊢ "Berta" : String ✓

subgoal: this: Main ⊢ new Student("Bob", 22, 12345).getAge() : int
-------------------------------------------------
apply (T-Invk)
subgoal: mtype(getAge, Student) = [] -> int ✓
(subgoal: this: Main ⊢• ✓) (no parameters)

subgoal: this: Main ⊢ new Student("Bob", 22, 12345) : Student
-------------------------------------------------
apply (T-New)
subgoal: fields(Student) = [String name, int age, int studentId] ✓
subgoal: this: Main ⊢ "Bob" : String ✓
subgaol: this: Main ⊢ 22 : int ✓
subgoal: this: Main ⊢ 12345 : int ✓
subgoal: String <: String ✓
subgoal: int <: int ✓
subgoal: int <: int ✓

no subgoals left ✓
```

b)

$$\frac{mbody(\text{getAge}, \text{Student}) = mbody(\text{getAge}, \text{Person}) \qquad mbody(\text{getAge}, \text{Person}) = [].\text{this.age}}{mbody(\text{getAge}, \text{Student}) = [].\text{this.age}}$$

$$\text{RC-New-Arg}\ \frac{\text{R-Invk}\ \dfrac{mbody(\text{getAge}, \text{Student}) = [].\text{this.age}}{\text{new Student("Bob", 22, 12345).getAge()} \to \text{new Student("Bob", 22, 12345).age}}}{\text{new Person("Berta", new Student("Bob", 22, 12345).getAge())} \to \text{new Person("Berta", new Student("Bob", 22, 12345).age)}}$$

$$\text{RC-New-Arg}\ \frac{\text{R-Field}\ \dfrac{fields(\text{Student}) = [\text{String name, int age, int studentId}]}{\text{new Student("Bob", 22, 12345).age} \to 22}}{\text{new Person("Berta", new Student("Bob", 22, 12345).age)} \to \text{new Person("Berta", 22)}}$$

c) Rule T-Let:

$$\frac{\Gamma \vdash e_1 : D \qquad \Gamma, x : D \vdash e_2 : C}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : C}$$

Rule R-Let:

$$\frac{}{\text{let } x = e_1 \text{ in } e_2 \longrightarrow [e_1/x]e_2}$$

Exercise 2 Generics

a)
```
static <A> void flip (Pair<A,A> p) {
    A tmp = p.snd;
    p.snd = p.fst;
    p.fst = tmp;
}
```

b) `Collection<?>` can hold a collection of anything, whereas `Collection<Object>` only holds collections to objects, i.e. `Collection<?> c = new ArrayList<String>();` is valid, whereas `Collection<Object> c = new ArrayList<String>();` is not (remember that generic types are invariant).

c) The output is `overloadedMethod (Collection <?>)`, because overloading is resolved at compile time and at compile time the parameter t has type List<T>, such that the most specific method that can be applied is the one for the generic collection. (see JLS 15.12)

d) The method without the parameter `a` can be written and compiled. But whenever you try to execute it, you will get a class cast exception, except for T = Object.

```
<T> T[] toArray() {
  T[] res = (T[])(new Object[l.size()]);
  for (int i = 0; i < res.length; i++) {
    res[i] = l.get(i);
  }
  return res;
}
```

The parameter serves as a kind of type token. Either the given array can be directly used or if the provided array is not large enough it can be used to create a bigger array, using e.g.

```
public static Object Array.newInstance(Class<?> componentType, int length) throws NegativeArraySizeException
```