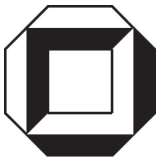

# Universität Karlsruhe (TH)

## Lehrstuhl für Programmierparadigmen


| | | |
|---|---|---|
| Fortgeschr. Objektorientierung SS 2009 | | http://pp.info.uni-karlsruhe.de/ |
| Dozent: | Prof. Dr.-Ing. G. Snelting | snelting@ipd.info.uni-karlsruhe.de |
| Übungsleiter: | Andreas Lochbihler | lochbihl@ipd.info.uni-karlsruhe.de |
| | Dennis Giffhorn | giffhorn@ipd.info.uni-karlsruhe.de |


| | | |
|---|---|---|
| Blatt 6 | Ausgabe: 28.05.2009 | Besprechung: 04.06.2009 |

1. **Multimethoden**

   Gegeben sei das folgende Programmfragment

```
1  class A { }
2  class B extends A {
3     void f(A x) { }
4     void f(B x) { }
5  }
6  class C extends B {
7     void f(A x) { }
8  }
9  class D extends B {
10    void f(B x) { }
11 }
12 ...
13    A a=new A();
14    A a_b=new B();
15    B b=new B();
16    B b_c=new C();
17    B b_d=new D();
18    C c=new C();
19    D d=new D();
20
21    b_c.f(a);
22    b_c.f(a_b);
23    b_c.f(b);
24    b_d.f(a);
25    b_d.f(a_b);
26    b_d.f(b);
27    c.f(a);
28    c.f(a_b);
29    c.f(b);
30    d.f(a);
31    d.f(a_b);
32    d.f(b);
```

(a) Nennen Sie Beispiele, in denen Multimethoden von Vorteil wären.

(b) Welche Methoden werden in normalem Java und welche bei Verwendung von Multi-Methoden aufgerufen?

(c) Gibt es Aufrufe, die jeweils in einer Variante eindeutig, aber in der anderen mehrdeutig sind?

(d) Machen Sie Vorschläge, wie man zur Laufzeit mit Mehrdeutigkeiten umgehen könnte.

2. **Überladung des Rückgabewertes**

   Bei der Überladungsauflösung wird in ADA der Typ des Rückgabewertes mit berücksichtigt, in Java und C++ wird er ignoriert.

   (a) Finden Sie Beispiele, in denen das Verhalten von ADA sinnvoll angewendet werden kann.

   (b) Diskutieren Sie Vor- und Nachteile beider Möglichkeiten.

3. **Rollen in C++**

   Erweitern Sie die Definition von *AnimalRole* so, daß ein Objekt einen Namen nebst Zugriffsfunktionen hat, der sich auch beim Rollentausch nicht ändert (siehe Code auf der nächsten Seite).

```
1  class Animal {
2    virtual void talk() { ... }
3    virtual void think() { ... }
4    virtual void act() { ... }
5  };
6
7  class AnimalRole { // muss nicht Unterklasse von Animal sein!
8  public:
9    enum {DOG, CAT};
10   AnimalRole() {
11     roles[DOG] = 0;
12     roles[CAT] = 0;
13     _role = DOG;
14   }
15   Animal* operator->() {
16     return roles[_role];}
17
18   void role(int r) {
19     _role = r;
20     if (roles[_role]=0) {
21        switch(r) {
22          case DOG: roles[_role] = new Dog(); break;
23          case CAT: roles[_role] = new Cat(); break;
24        }
25     }
26   }
27
28 protected:
29   int _role;
30   Animal* roles[2];
31   // hier Rollen-gemeinsame Membervariablen
32 };
33
34 class Dog: public Animal {
35 public:
36   void talk() { ... "wuff" ... }
37   void act() { ... }
38 };
39
40 class Cat: public Animal {
41 public:
42   void talk() { ... "miau" ... }
43   void think() { ... }
44 };
45
46 //Verwendung:
47 AnimalRole x;
48 x.role(AnimalRole::DOG);
49 x->talk(); x->think(); x->act();
50 x.role(AnimalRole::CAT);
51 x->talk(); x->think(); x->act();
```