



## Übung zur Vorlesung *Einführung in die Informatik 2 für Ingenieure (MSE)*

Christoph Anneser (anneser@in.tum.de)

<http://db.in.tum.de/teaching/ss23/ei2/>

### Blatt Nr. 3

Dieses Blatt wird am Montag, den 15.05.2023 besprochen.

Zugehörige Vorlesungsaufzeichnung: [Vorlesung 4](#)

### Aufgabe 1: Overloading

Welche der folgenden Methoden-Überladungen sind erlaubt und welche nicht? Überprüfen Sie Ihre Antworten indem Sie die Beispiele in Java programmieren. Eine unterstrichener Methodenname bedeutet dass es sich um eine Klassenmethode handelt.

a) 

Car
-speed : double
+accelerate(mph : double)
+accelerate(kmh : double)

b) 

MetricCar
-speed : double
+accelerate(kmh : double)

ImperialCar
-speed : double
+accelerate(mph : double)

c) 

Car
-speed : double
+accelerate(kmh : double)
+accelerate(seconds : int)

d) 

Car
-speed : double
+accelerate(seconds : int)
+accelerate(seconds : int) : double

e) 

Car
-speed : double
+toString() : String
+toString() : String

f) 

Car
-speed : double
+accelerate(kmh : double)
+accelerate(mph : double, slope : double)

Car
-speed : double
+load(passengers : Set<Passenger>)
+load(luggage : Set<Suitcase>)

g)

## Aufgabe 2: Dynamisches Binden - Teil 1

Angenommen, es gibt die folgenden Klassenbeziehungen:

```

1 abstract class A {}
2 class B extends A {}
3 class C extends B {}
4 class D extends B {}
5 class E extends D {}

```

Geben Sie in der Tabelle an, welche Kombinationen von statischem und dynamischen Typen möglich sind.

		dynamischer Typ				
		A	B	C	D	E
statischer Typ	A					
	B					
	C					
	D					
	E					

## Aufgabe 3: Dynamisches Binden - Teil 2

Überlegen Sie sich welche Methoden aufgerufen werden, wenn man die Klasse `DynamicDispatch` ausführt. Überprüfen Sie anschließend ihre Vermutung, indem Sie das Programm tatsächlich ausführen.

```

1 class DynamicDispatch {
2     public static void main(String [] args) {
3         A a = new A();
4         B b = new B();
5         C c = new C();
6         D d = new D();
7
8         A[] array = {a, b, c, d};
9         for (A element : array) {
10            System.out.println("x()");
11            element.x();
12            System.out.println("\ny()");
13            element.y();
14            System.out.println("\nz()");
15            element.z();

```

```

16     System.out.println("\n=====\\n");
17     }
18 }
19 }
20
21 class A {
22     public void x() {
23         System.out.println("->A_x()");
24         z();
25     }
26
27     public void y() {
28         System.out.println("->A_y()");
29         this.z();
30     }
31
32     public void z() {
33         System.out.println("->A_z()");
34     }
35 }
36
37 class B extends A {
38     public void y() {
39         System.out.println("->B_y()");
40         x();
41     }
42
43     public void z() {
44         System.out.println("->B_z()");
45     }
46 }
47
48 class C extends B {
49     public void x() {
50         System.out.println("->C_x()");
51         z();
52     }
53 }
54
55 class D extends A {
56     public void x() {
57         System.out.println("->D_x()");
58         super.x();
59     }
60
61     public void z() {
62         System.out.println("->D_z()");
63     }

```

