TU München, Fakultät für Informatik
Lehrstuhl III: Datenbanksysteme
Prof. Alfons Kemper, Ph.D.

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

Christoph Anneser (anneser@in.tum.de)
http://db.in.tum.de/teaching/ss22/ei2/

## Lösungen zu Blatt 3

### Aufgabe 1: Overloading

Welche der folgenden Methoden-Überladungen sind erlaubt und welche nicht? Überprüfen Sie Ihre Antworten indem Sie die Beispiele in Java programmieren.

### Lösung 1

Das Laufzeitsystem muss herausfinden, welche der beiden überladenen Methoden es aufrufen muss, wenn diese im laufenden Programm verwendet werden. Damit dies möglich ist, müssen sich diese entweder in der Anzahl der Parameter oder in den Parametertypen unterscheiden.

a) **Nicht erlaubt:** Beide Methoden haben genau einen Parameter vom Typ `double` und sind damit nicht unterscheidbar.

| Car |
|---|
| -speed : double |
| +accelerate(mph : double)<br>+accelerate(kmh : double) |

b) **Erlaubt:** Hier gibt es keinen Konflikt, da die Methoden in verschiedenen Klassen definiert sind.

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

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

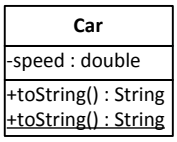
c) **Erlaubt:** Die verschiedene Parametertypen erlauben dem Laufzeitsystem die Unterscheidung.

| Car |
|---|
| -speed : double |
| +accelerate(kmh : double)<br>+accelerate(seconds : int) |

d) **Nicht erlaubt:** Der Typ des Rückgabewertes wird nicht zur Unterscheidung herangezogen.

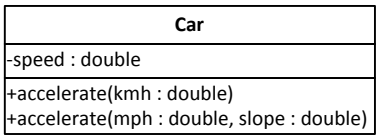
| Car |
|---|
| -speed : double |
| +accelerate(seconds : int)<br>+accelerate(seconds : int) : double |

e) **Nicht erlaubt:** Klassenmethoden werden in diesem Fall wie Objektmethoden behandelt.

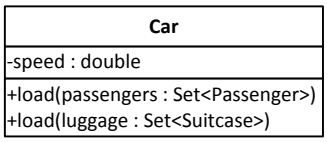


[1]

f) **Erlaubt:** Die verschiedene Anzahl Parameter ermöglicht die Unterscheidung.




g) **Nicht erlaubt:** Die generische Typinformation wird bei der Kompilierung entfernt und steht zur Laufzeit nicht mehr zu Verfügung. Beide Methoden haben dadurch genau einen Parameter vom Typ `Set<Object>` sind damit nicht mehr unterscheidbar.




### 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 {}
```

| | | dynamischer Typ | | | | |
|---|---|---|---|---|---|---|
| | | A | B | C | D | E |
| statischer Typ | A | | ✔ | ✔ | ✔ | ✔ |
| | B | | ✔ | ✔ | ✔ | ✔ |
| | C | | | ✔ | | |
| | D | | | | ✔ | ✔ |
| | E | | | | | ✔ |

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

### 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] Unterstrichen bedeutet hier Klassenmethode.

```
class DynamicDispatch {
  public static void main(String[] args) {
    A a = new A();
    B b = new B();
    C c = new C();
    D d = new D();

    A[] array = {a, b, c, d};
    for (A element : array) {
      System.out.println("x():");
      element.x();
      System.out.println("\ny():");
      element.y();
      System.out.println("\nz():");
      element.z();
      System.out.println("\n=====\n");
    }
  }
}

class A {
  public void x() {
    System.out.println("-> A x()");
    z();
  }

  public void y() {
    System.out.println("-> A y()");
    this.z();
  }

  public void z() {
    System.out.println("-> A z()");
  }
}

class B extends A {
  public void y() {
    System.out.println("-> B y()");
    x();
  }

  public void z() {
    System.out.println("-> B z()");
  }
}

```

```
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   }
64 }
```

**Lösung 2**

Das Konzept der Vererbung beinhaltet in Java, dass eine Variable von einem bestimmten Klassentyp auch Objekte von deren Untertklassen referenzieren kann. Ruft man nun eine Methode auf dem durch die Variable referenzierten Objekt auf, gibt es zwei Möglichkeiten:

- Führe die Methode der Oberklasse aus
- Führe die überschriebene Methode einer der Unterklassen aus

Von *dynamischer Bindung* spricht man, wenn in solchen Fällen immer die speziellste Variante der Methode aufgerufen wird, d.h. bei einem Objekt der Unterklasse würde, wenn diese existiert, die überschriebene Methode dieser Unterklasse aufgerufen werden. Wird stattdessen immer die Methode der Oberklasse genutzt, so spricht man von *statischer Bindung*. Bei Java hat man dabei keine Wahl: Methoden werden immer dynamisch gebunden. Die dynamische Bindung ist von enormer Bedeutung für die objektorientierte Programmierung, da die Flexibilität der Vererbung nur durch dynamische Bindung zum Tragen kommt. Ohne dynamische Bindung könnten Unterklassen das Verhalten der Oberklasse nicht anpassen, sondern nur um neue Methoden erweitern.

Zurück zur eigentlichen Aufgabe: Die Ausgabe des Programms verdeutlicht, in welcher Reihenfolge die Methoden aufgerufen werden.