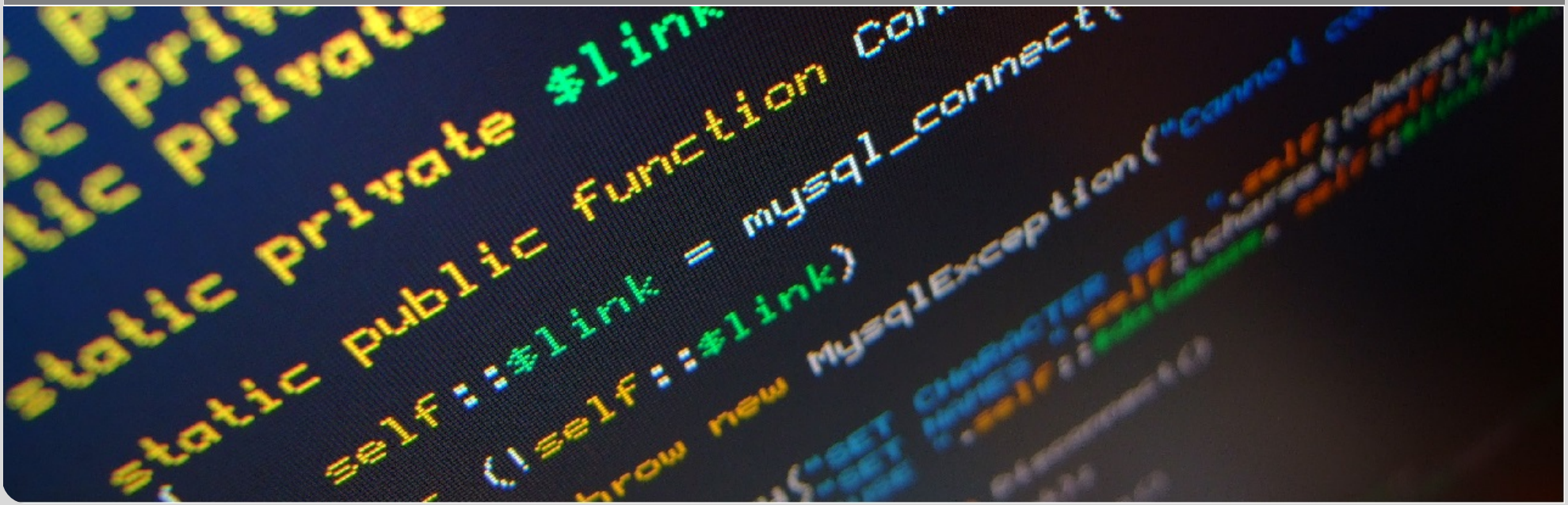
Karlsruher Institut für Technologie

Tutorium Programmieren

Tut Nr.9: Vererbung

Michael Friedrich | 07. / 09.01.2014

INSTITUT FÜR THEORETISCHE INFORMATIK



Outline/Gliederung

Übungsblatt4

- Nutzt die öffentlichen Test als Hilfe

Übungsblatt4

- Nutzt die öffentlichen Test als Hilfe
  → Bei Abschlussaufgaben Abgabekriterium!
  ABER kein hardcoden!!! - Zählt als Betrugsversuch

Einführung

Wird genutzt, um Gleiches zu „gruppieren“

Beispiel

```
class Person {
  String name;
  int age;
}
class Student extends Person {
   int matriculation;
  Tutor tutor;
}
class Tutor extends Student {
//hat auch name, age, matriculation
String name; //anderes name als Person.name!
  Student[] students;
}
```

ACHTUNG Überschreiben von Attributen nicht möglich

Aufpassen!

- Vererbung wird gerne falsch gemacht! $\Rightarrow$ immer IST-Beziehung

SO NICHT

```
class Point {
  int x, y;
}
class Linie extends Point {
  int x2, y2; // huh?
}
```

Aufpassen!

- Vererbung wird gerne falsch gemacht! $\Rightarrow$ immer IST-Beziehung

SO NICHT

```
class Point {
  int x, y;
}
class Linie extends Point {
  int x2, y2; // huh?
}
```

Sondern?

Aufpassen!

- Vererbung wird gerne falsch gemacht! $\Rightarrow$ immer IST-Beziehung

SO NICHT

```
class Point {
  int x, y;
}
class Linie extends Point {
  int x2, y2; // huh?
}
```

Sondern?

```
class Linie {
  Point p1, p2; //HAT–Beziehung
}
```

Vererbung von Methoden

- habt ihr alle schon implizit genutzt, Beispiele?

Vererbung von Methoden

- habt ihr alle schon implizit genutzt, Beispiele?
  - `toString`, `equals(Object o)`, ...

Vererbung von Methoden

- habt ihr alle schon implizit genutzt, Beispiele?
  - `toString`, `equals(Object o)`, ...
- diese Methoden erbt JEDE Klasse von `java.lang.Object`

Vererbung von Methoden

- habt ihr alle schon implizit genutzt, Beispiele?
  - `toString`, `equals(Object o)`, ...
- diese Methoden erbt JEDE Klasse von `java.lang.Object`
  Habt ihr schon überschrieben (daher @Override)

Vererbung von Methoden

- habt ihr alle schon implizit genutzt, Beispiele?
  - `toString`, `equals(Object o)`, ...
- diese Methoden erbt JEDE Klasse von `java.lang.Object`
  Habt ihr schon überschrieben (daher @Override)
- Konstruktoren werden mitvererbt und **müssen immer** in der Subklasse implementiert werden
  → Zugriff auf den Oberklassen Methoden (also auch Konstruktor) über super()

Vererbung von Methoden

- habt ihr alle schon implizit genutzt, Beispiele?
  - `toString`, `equals(Object o)`, ...
- diese Methoden erbt JEDE Klasse von `java.lang.Object`
  Habt ihr schon überschrieben (daher @Override)
- Konstruktoren werden mitvererbt und **müssen immer** in der Subklasse implementiert werden
  → Zugriff auf den Oberklassen Methoden (also auch Konstruktor) über super()

super

```
public Student(name, age, matriculation) {
 super(name, age);
 this.matriculation = matriculatio;
}
```

Polymorphismus

- Eine Unterklasse kann alles, was die Oberklasse kann
  → exakt gleiche Methoden, auch wenn eventuell überschrieben

Beispiel:

```
Student michael = new Student();
Person mike = michael; //Jeder Student IST also gleichzeitig eine
    Person
Person anna = new Student();
Student anne = new Person(); ↯ NEIN!
```

Abstrakte (abstract) Klassen

- Klassen: Nicht instantiierbar
- Methoden: MÜSSEN in Subklasse instantiiert werden
  - nur in abstrakten Klassen erlaubt

Abstrakte (abstract) Klassen

- Klassen: Nicht instantiierbar
- Methoden: MÜSSEN in Subklasse instantiiert werden
  - nur in abstrakten Klassen erlaubt

```
abstract class mammal {
//protected nur in erbenden Klassen sichtbar
  protected String s = "schnaufen";

  abstract public void breathe();
}

class human extends mammal {

  public void breathe() {
    System.out.print(s);
  }
}
```

Interfaces - implements

- ähnlich den abstrakten Klassen aber keine Attribute

Interfaces - implements

- ähnlich den abstrakten Klassen aber keine Attribute
- definieren Schnittstelle
  - also anderer Sinn als abstrakte Klassen!

Interfaces - implements

- ähnlich den abstrakten Klassen aber keine Attribute
- definieren Schnittstelle
  - also anderer Sinn als abstrakte Klassen!
  - definieren die Methoden, die eine Klasse implementieren **MUSS**

```
public interface Comparable {
  boolean equals();
  int compareTo();
}
public Letter implements Comparable {
  char c;

  public boolean equals(Letter c) {
    return this.c==c;
  }
}
```

Interfaces - implements

- ähnlich den abstrakten Klassen aber keine Attribute
- definieren Schnittstelle
  - also anderer Sinn als abstrakte Klassen!
  - definieren die Methoden, die eine Klasse implementieren **MUSS**

```
public interface Comparable {
  boolean equals();
  int compareTo();
}
public Letter implements Comparable {
  char c;

  public boolean equals(Letter c) {
    return this.c==c;
  }
}
```

Methoden automatisch abstract und public

Interfaces

- Eine Klasse kann beliebig viele Interfaces implementieren

Interfaces

- Eine Klasse kann beliebig viele Interfaces implementieren
- Beziehung zwischen Interfaces über extends möglich

Interfaces

- Eine Klasse kann beliebig viele Interfaces implementieren
- Beziehung zwischen Interfaces über extends möglich

Aufgabe

Tut Aufgabe zu Vererbung/Polymorphismus

Rekursion

Bedeutung: Selbstaufruf einer Funktion

Rekursion

Bedeutung: Selbstaufruf einer Funktion

```
public class Fibs {
 public static int fib(final int n) {
    if (n <= 1) {
      return 1;
    }
    return fib(n - 1) + fib(n - 2);
  }
}
```

Rekursion

Bedeutung: Selbstaufruf einer Funktion

```
public class Fibs {
 public static int fib(final int n) {
    if (n <= 1) {
      return 1;
    }
    return fib(n - 1) + fib(n - 2);
  }
}
```

- Was passiert bei dem Aufruf von `fib(4)`?

Rekursion

Bedeutung: Selbstaufruf einer Funktion

```
public class Fibs {
 public static int fib(final int n) {
    if (n <= 1) {
      return 1;
    }
    return fib(n - 1) + fib(n - 2);
  }
}
```

- Was passiert bei dem Aufruf von `fib(4)`?
- Man braucht immer einen **terminierenden** Fall!

Generics

Wie schon letztes Blatt gemerkt, ein Tool, um Gleiches zu komprimieren
→ Listen sind identisch, bis auf ihren Inhalt

Generics

Wie schon letztes Blatt gemerkt, ein Tool, um Gleiches zu komprimieren

→ Listen sind identisch, bis auf ihren Inhalt

```
public class SinglyLinkedList<E> {
  private Node<E> head;

  public void add(E object) {
     Node<E> newNode = new Node<E>(object);
     newNode.setNext(head);
     head = newNode;
  }
}
```

Generics

Wie schon letztes Blatt gemerkt, ein Tool, um Gleiches zu komprimieren

→ Listen sind identisch, bis auf ihren Inhalt

```
public class SinglyLinkedList<E> {
  private Node<E> head;

  public void add(E object) {
     Node<E> newNode = new Node<E>(object);
     newNode.setNext(head);
     head = newNode;
  }
}
```

„E“ ist dann durch euren gewünschter Datentyp ersetzbar, zB

`new SinglyLinkedList<Article>`

Generics

Wie schon letztes Blatt gemerkt, ein Tool, um Gleiches zu komprimieren

→ Listen sind identisch, bis auf ihren Inhalt

```
public class SinglyLinkedList<E> {
  private Node<E> head;

  public void add(E object) {
     Node<E> newNode = new Node<E>(object);
     newNode.setNext(head);
     head = newNode;
  }
}
```

„E“ ist dann durch euren gewünschter Datentyp ersetzbar, zB

`new SinglyLinkedList<Article>`

Aufgabe: restliche Liste mit Generics implementieren