hhu.

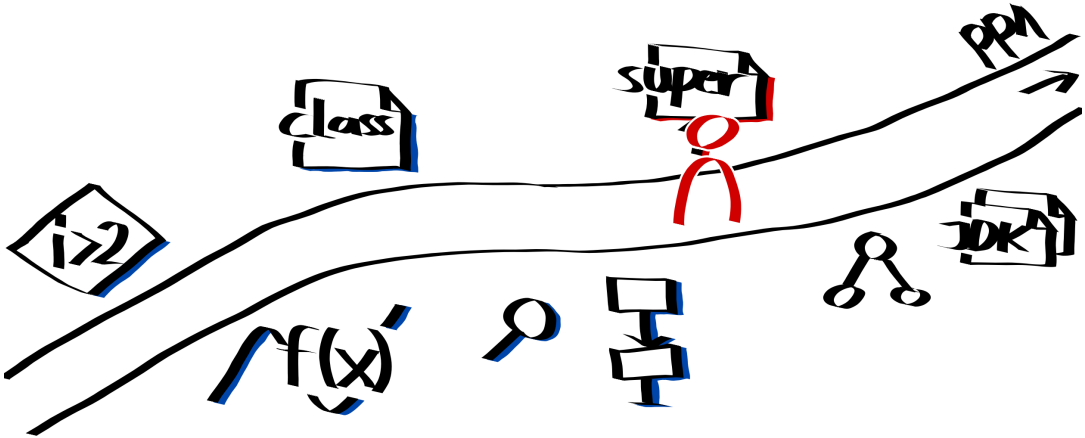
Kapitel 6: Vererbung & Fehlerbehandlung

Abstrakte Klassen & Super-Konstruktoren

Wintersemester 2023/24

Stand 3. August 2023, 15:36 Uhr

Wo stehen wir gerade?



Vorlesungsfeedback

Abstrakte Klassen

Heinrich Heine Universität Düsseldorf

- können nicht instanziiert werden
- enthalten typischerweise mindestens 1 Methode ohne Implementierung („abstrakte Methode“)
- Unterklassen von abstrakten Klassen müssen entweder abstrakte Methoden durch Überschreiben implementieren oder selbst abstrakt sein

```
3  public abstract class FlyingObject {
4
5      private Color color;
6
7      public FlyingObject() {
8          int c = (int)(Math.random() * 255);
9          this.color = new Color(c, c, c);
10     }
11
12     public abstract draw();
13
14 }
```

Interfaces vs. abstrakte Klassen

Interfaces vs. abstrakte Klassen

Heinrich Heine
Universität Düsseldorf

Interfaces

- definieren Datentyp, der nicht instanziiert werden kann
- definieren Schnittstelle für Verhalten (Methodensignaturen)
- Klasse `A` kann Interface *implementieren* (Interface ist Obertyp von `A`)

Abstrakte Klassen:

- definieren Datentyp, der nicht instanziiert werden kann
- können Methoden ohne Implementierung haben
- Klasse `A` kann von abstrakter Klasse *erben* (Abstrakte Klasse ist Obertyp & Oberklasse von `A`)

Interfaces vs. abstrakte Klassen

Heinrich Heine Universität Düsseldorf

Interfaces

- definieren Datentyp, der nicht instanziiert werden kann
- definieren Schnittstelle für Verhalten (Methodensignaturen)
- Klasse `A` kann Interface *implementieren* (Interface ist Obertyp von `A`)

\+ Klassen können mehrere Interfaces implementieren

– alle Interface-Methoden öffentlich

Abstrakte Klassen:

- definieren Datentyp, der nicht instanziiert werden kann
- können Methoden ohne Implementierung haben
- Klasse `A` kann von abstrakter Klasse *erben* (Abstrakte Klasse ist Obertyp & Oberklasse von `A`)

– Klassen können nur von einer Klasse erben

\+ können Methoden mit Implementierung und Instanzvariablen vererben

Interface oder Vererbung?

Heinrich Heine
Universität Düsseldorf

Frage

Wenn ich die Wahl habe, ob ich ein Interface oder eine abstrakte Klasse schreibe: Was sollte ich nehmen?

Interface oder Vererbung?

Heinrich Heine
Universität Düsseldorf

Frage

Wenn ich die Wahl habe, ob ich ein Interface oder eine abstrakte Klasse schreibe: Was sollte ich nehmen?

Praxis (in Java): Interface vorziehen. $\rightarrow$Programmierpraktikum

Können Interfaces auch erben?

Heinrich Heine
Universität Düsseldorf

Können Interfaces auch erben?

Heinrich Heine Universität Düsseldorf

```java
public interface List<T> {
    T get(int index);
    T first();
    T last();
    int size();
}
```

```java
public interface MutableList<T> extends List<T> {
    void add(int index, T value);
    void set(int index, T value);
    void remove(int index);
}
```

Anwendungsbeispiel: Personen an der Uni

Heinrich Heine Universität Düsseldorf

Ziel: Objektorientierte Modellierung der Personenverwaltung einer Universität.

Jede Person hat:

- Name
- E-Mailadresse

Jede:r Student:in hat:

- Matrikelnummer (kann leer sein)
- alle Eigenschaften einer Person

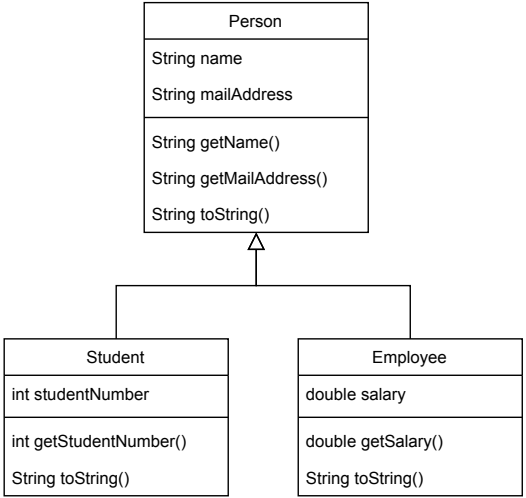
Jede:r Mitarbeiter:in hat:

- Gehalt
- alle Eigenschaften einer Person

Außerdem soll gelten:

- Objektvariablen haben minimale Sichtbarkeit
- alle Objekteigenschaften über Getter abfragbar (maximale Sichtbarkeit)
- Objekteigenschaften unveränderlich
- String-Repräsentation enthält alle Eigenschaften, z. B.
  `Kim Maier, kimai@hhu.de, 4728912`

Grafische Übersicht der Anforderungen



Aufrufreihenfolge von Konstruktoren

- Konstruktor einer Klasse ruft als erstes Konstruktor der Oberklasse auf
- wenn nicht explizit angegeben: Aufruf des Konstruktors ohne Argumente

```java
public abstract class FlyingObject {

  public FlyingObject() {
    System.out.println("new flying
    ↪   object created");
  }

}
```

```java
public class Ball extends FlyingObject {

  public Ball() {
    System.out.println("new ball created");
  }

}
```

Was gibt `new Ball()` aus?

Aufrufreihenfolge von Konstruktoren

- Konstruktor einer Klasse ruft als erstes Konstruktor der Oberklasse auf
- wenn nicht explizit angegeben: Aufruf des Konstruktors ohne Argumente

```java
public abstract class FlyingObject {

  public FlyingObject() {
    System.out.println("new flying
      ↪   object created");
  }

}
```

```java
public class Ball extends FlyingObject {

  public Ball() {
    System.out.println("new ball created");
  }

}
```

```
new flying object created
new ball created
```

super-Konstruktor

- explizite Wahl eines Konstruktors mit `super()` möglich
  - notwendig, wenn Oberklasse keinen Konstruktor ohne Argumente hat
  - muss erste Anweisung im Konstruktor sein

```java
public class Person {
  private String name;

  public Person(String name) {
    this.name = name;
    System.out.println(name);
  }
}
```

```java
public class Employee extends Person {
  private double salary;

  public Employee(String name, double salary) {
    super(name);
    this.salary = salary;
    System.out.println("salary: " + salary);
  }
}
```

Was gibt `new Employee("Kim", 4074.30)` aus?

super-Konstruktor

- explizite Wahl eines Konstruktors mit `super()` möglich
  - notwendig, wenn Oberklasse keinen Konstruktor ohne Argumente hat
  - muss erste Anweisung im Konstruktor sein

```java
public class Person {
  private String name;

  public Person(String name) {
    this.name = name;
    System.out.println(name);
  }
}
```

```java
public class Employee extends Person {
  private double salary;

  public Employee(String name, double salary) {
    super(name);
    this.salary = salary;
    System.out.println("salary: " + salary);
  }
}
```

```
Kim
salary: 4074.3
```

Warum ist diese Reihenfolge sinnvoll?

```java
public class Person {
  private String name;

  public Person(String name) {
    this.name = name;
    System.out.println(name);
  }

  public String getName() {
    return name;
  }
}
```

```java
public class Employee extends Person {
  private double salary;

  public Employee(String name, double salary) {
    this.salary = salary;
    System.out.println("name: " + getName());
    super(name); // das mag Java nicht
  }
}
```

Warum ist diese Reihenfolge sinnvoll?

```java
public class Person {
  private String name;

  public Person(String name) {
    this.name = name;
    System.out.println(name);
  }

  public String getName() {
    return name;
  }
}
```

```java
public class Employee extends Person {
  private double salary;

  public Employee(String name, double salary) {
    this.salary = salary;
    System.out.println("name: " + getName());
    super(name); // das mag Java nicht
  }
}
```

→ wenn `getName()` aufgerufen wird, ist der Name noch gar nicht gesetzt

⇒ sinnvoll, wenn zuerst Eigenschaften der Oberklasse initialisiert

`this` vs. `super`

```java
public class Person {
  private String name;

  public Person(String name) {
    this.name = name;
    System.out.println(name);
  }
}
```

```java
public class Employee extends Person {
  private double salary;

  public Employee(String name, double salary) {
    super(name);
    this.salary = salary;
    System.out.println("salary: " + salary);
  }

  public Employee(String name) {
    this(name, 0);
  }
}
```

Was gibt `new Employee("Kim")` aus?

`this` vs. `super`

```java
public class Person {
  private String name;

  public Person(String name) {
    this.name = name;
    System.out.println(name);
  }
}
```

```java
public class Employee extends Person {
  private double salary;

  public Employee(String name, double salary) {
    super(name);
    this.salary = salary;
    System.out.println("salary: " + salary);
  }

  public Employee(String name) {
    this(name, 0);
  }
}
```

```
Kim
salary: 0.0
```

Werden Konstruktoren vererbt?

Heinrich Heine Universität Düsseldorf

```java
public class Person {
  private String name;

  public Person(String name) {
    this.name = name;
    System.out.println(name);
  }
}
```

```java
public class Employee extends Person {
  private double salary;

  public Employee(String name, double salary) {
    super(name);
    this.salary = salary;
    System.out.println("salary: " + salary);
  }
}
```

Was gibt `new Employee("Kim")` aus?

Werden Konstruktoren vererbt?

Heinrich Heine Universität Düsseldorf

```java
public class Person {
  private String name;

  public Person(String name) {
    this.name = name;
    System.out.println(name);
  }
}
```

```java
public class Employee extends Person {
  private double salary;

  public Employee(String name, double salary) {
    super(name);
    this.salary = salary;
    System.out.println("salary: " + salary);
  }
}
```

```
Test.java:5: error: constructor Employee in class Employee cannot be applied to
  given types;
    Person p = new Employee("Kim");
                   ^
  required: String,double
  found: String
  reason: actual and formal argument lists differ in length
1 error
```

Welcher Super-Konstruktor wird zuerst aufgerufen?

Heinrich Heine Universität Düsseldorf

```java
public class Amphibienfahrzeug extends Boot, Auto {
  public Amphibienfahrzeug() {
    System.out.println("Neues Amphibienfahrzeug");
  }
}
```

```java
public class Boot {
  public Boot() {
    System.out.println("Neues Boot");
  }
}
```

```java
public class Auto {
  public Auto() {
    System.out.println("Neues Auto");
  }
}
```

Welcher Super-Konstruktor wird zuerst aufgerufen?

Heinrich Heine Universität Düsseldorf

```java
public class Amphibienfahrzeug extends Boot, Auto {
  public Amphibienfahrzeug() {
    System.out.println("Neues Amphibienfahrzeug");
  }
}
```

```java
public class Boot {
  public Boot() {
    System.out.println("Neues Boot");
  }
}
```

```java
public class Auto {
  public Auto() {
    System.out.println("Neues Auto");
  }
}
```

Keine Mehrfachvererbung

Eine Klasse kann in Java nur direkt von einer anderen Klasse erben.

Welcher Super-Konstruktor wird zuerst aufgerufen?

Heinrich Heine Universität Düsseldorf

```java
public class Amphibienfahrzeug extends Boot, Auto {
  public Amphibienfahrzeug() {
    System.out.println("Neues Amphibienfahrzeug");
  }
}
```

```java
public class Boot {
  public Boot() {
    System.out.println("Neues Boot");
  }
}
```

```java
public class Auto {
  public Auto() {
    System.out.println("Neues Auto");
  }
}
```

```
Amphibienfahrzeug.java:1: error: '{' expected
public class Amphibienfahrzeug extends Boot, Auto {
                                                  ^
1 error
```

Lernziele

Heinrich Heine
Universität Düsseldorf

Sie können am Ende der Woche ...

- Abstrakte Klassen nach Vorgaben **erstellen**.
- einfache Gegebenheiten der echten Welt objektorientiert **modellieren**.
- Konstruktoren der Oberklasse **aufrufen**.

Wichtige Begriffe

Heinrich Heine
Universität Düsseldorf

Abstrakte Klasse Abstrakte Methode abstract
super()

Übungsblatt

Heinrich Heine
Universität Düsseldorf

- Widgets
- Neujahrsquiz