

Reduce, Reuse, Recycle: Objektorientierung II

Prof. Dr. Christian Becker

Universität Stuttgart, Institut für Parallele und Verteilte Systeme

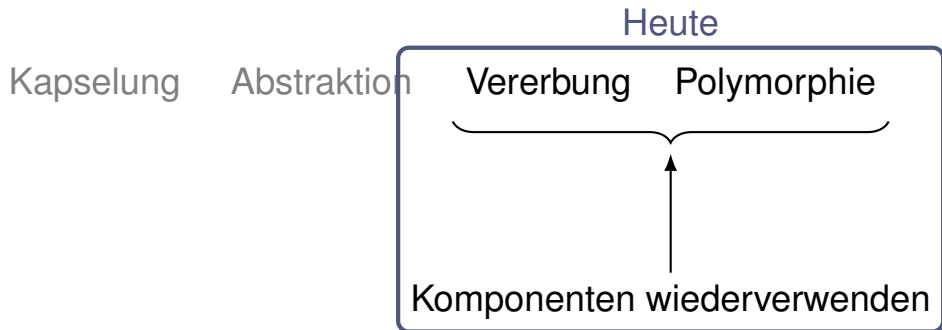
15. Mai 2025



Problemverständnis

- ▶ Menschen sind faul, Programmierer auch.
- ▶ Software ist riesig und kompliziert
- ▶ Man möchte so wenig (Code) wie möglich lesen müssen um zu verstehen was ein Stück Software tut
- ▶ Leider wird man aber immer Code von anderen Leuten benutzen müssen
- ▶ Andere werden Ihren Code benutzen müssen...
- ▶ Schreiben Sie Ihren Code so, dass der nächste Idiot ihn versteht.
- ▶ Sie sind meistens selbst der nächste Idiot

Die Grundkonzepte der Objektorientierung



- ▶ Wir haben Klassen kennengelernt, mit denen wir Daten und Verhalten kapseln konnten
- ▶ Heute lernen wir, wie man Klassen wiederverwendet
- ▶ Wiederverwenden von Software spart Kosten, Zeit: Nicht nur muss man weniger programmieren, man muss auch bei weniger prüfen, ob der Code richtig ist!
- ▶ Vererbung ist eine Methode, Teile von Code wiederzuverwenden

Interfaces

- ▶ Wir haben schon Interfaces kennengelernt als die Menge der `public` Methoden in einer Klasse
- ▶ Man kann den Begriff Interface noch weiter greifen und eine Art Vertrag beschreiben
- ▶ Dieser Vertrag ist zwischen verschiedenen Entwicklungsteams und ermöglicht Kompatibilität
- ▶ Dann schreibt das Interface eine Menge von Interaktionspunkten vor

Ein mögliches Interface

- ▶ Komponenten wiederverwenden erfordert, dass das Interface einer Komponente
- ▶ Sie hätten gerne, dass jemand für sie ein

1

Java



Abstract Data Types (ADTs)



Vererbung









Liskov'sches Substitutionsprinzip

Eigenschaften die für Objekte vom Typ A gelten müssen auch für alle Objekte vom Typ B gelten, wenn B von A erbt. Ein Beispiel bei dem Vererbung zu einer Verletzung des Substitutionsprinzips führen kann ist das Kreis-Ellipse-Problem:

```
1  class Element {  
2      public void scaleX() {}  
3      public void scaleY() {}  
4  }  
5  class Ellipse extends Element {  
6      public void scaleX() {}  
7      public void scaleY() {}  
8  }  
9  class Kreis extends Ellipse {  
10     public void scaleX() {}  
11     public void scaleY() {}  
12 }
```

Java

Das Skalieren einer Ellipse in X-Richtung ist erlaubt. Das Skalieren eines Kreises in X-Richtung würde den Kreis in eine Ellipse verwandeln und einen Fehler darstellen.

Zugriffsbeschränkungen

- ▶ Wir wollen weiterhin so wenig Code wie möglich lesen müssen.
- ▶ Unterteilen in Implementierungsdetails (private) und öffentlichen Zustand / öffentliches Verhalten (public)

```
1  class LinkedList {  
2      private class Node {  
3          ...  
4      };  
5  
6      public LinkedList() { ... }  
7  
8      public void sort() { ... }  
9      public boolean find(int value) { ... }  
10     public void append(int value) { ... }  
11     public void delete(int position) { ... }  
12  
13     private Node front;  
14     private Node back;  
15 }
```

Java

private: Details, die für die Benutzung unwichtig sind

public: Wichtig für die Benutzung

Implementierung weiterhin unwichtig für die Benutzung

protected Einfachvererbung - protected: access rule for subclasses

Polymorphie

- ▶ Polymorphie beschreibt ein Prinzip aus der Biologie, in dem ein Organismus oder eine Spezies viele verschiedene Formen oder Zustände hat.
- ▶ Dieses Prinzip trifft auch auf Vererbungshierarchien in der kann auch in der Objektorientierung zu
- ▶ Kindklassen können eigenes Verhalten beschreiben und sich trotzdem Funktionalität mit ihrer Elternklasse teilen

Polymorphie

Vererbung bildet eine Hierarchie

Objekte können durch den Typ der Elternklassen adressiert werden

```
1 Dog rex = new Dog();  
2 Animal myPet = rex;
```

Java

myPet ist ein zweiter Name für das Objekt rex

Hier verhält sich `myPet` polymorph. Es ist also zum Beispiel möglich verschiedene Tiere in der gleichen Liste zu speichern solange sie alle die gleiche Elternklasse teilen.

Überschreiben und Wiederverwenden

```
1  class Animal {  
2      public void makeSound() { System.out.println("generic_animal_sound_1.wav"); }  
3  }  
4  class Dog extends Animal {  
5      public void makeSound() { System.out.println("wuff"); }  
6  }
```

Java

Das Interface der Elternklasse

```
1  class Animal {  
2      public void makeSound() { System.out.println("generic_animal_sound_1.wav"); }  
3  }  
4  class Dog extends Animal {  
5      public void makeSound() { System.out.println("wuff"); }  
6  }  
7  class Cat extends Animal {  
8      public void makeSound() { System.out.println("miau"); }  
9  }
```

Java

Kindklassen können Verhalten der Elternklasse verwenden oder überschreiben.

- ▶ Es gibt mehrere Implementierungen zu der gleichen Methodensignatur in einer Hierarchie
- ▶ Was bedeutet ein Funktionsaufruf dann?
- ▶ Welche Funktion wird aufgerufen?

Binden von Funktionen

- moritz ist vom Typ Cat und Cat definiert eine Methode makeSound

```
1  class Animal {  
2      public void makeSound() {  
3          System.out.println("sound.wav");  
4      }  
5  }  
6  class Dog extends Animal {  
7      public void makeSound() {  
8          System.out.println("wuff");  
9      }  
10 }  
11 class Cat extends Animal {  
12     public void makeSound() {  
13         System.out.println("miau");  
14     }  
15 }
```

Java

```
1  Cat moritz = new Cat();  
2  moritz.makeSound();
```

Java

Binden von Funktionen

- ▶ myPet ist vom Typ Animal, zeigt aber auf das Objekt max vom Typ Dog
- ▶ Sowohl Animal als auch Dog definieren eine Methode makeSound

```
1 class Animal {  
2     public void makeSound() {  
3         System.out.println("sound.wav");  
4     }  
5 }  
6 class Dog extends Animal {  
7     public void makeSound() {  
8         System.out.println("wuff");  
9     }  
10 }  
11 class Cat extends Animal {  
12     public void makeSound() {  
13         System.out.println("miau");  
14     }  
15 }
```

Java

```
1 Dog max = new Dog();  
2 Animal myPet = max;  
3  
4 myPet.makeSound();
```

Java

?

Binden von Funktionen

static binding

- ▶ Erste Möglichkeit: statisches Binden der Funktionsaufrufe zur Compile-Zeit
- ▶ Da die Referenz (myPet) vom Typ Animal ist, wird die Methode die von Animal definiert wurde aufgerufen

```
1  class Animal {  
2      public void makeSound() {  
3          System.out.println("sound.wav");  
4      }  
5  }  
6  class Dog extends Animal {  
7      public void makeSound() {  
8          System.out.println("wuff");  
9      }  
10 }  
11 class Cat extends Animal {  
12     public void makeSound() {  
13         System.out.println("miau");  
14     }  
15 }
```

Java

```
1  Dog max = new Dog();  
2  Animal myPet = max;  
3  
4  myPet.makeSound();
```

Java

Achtung: So nicht in Java



Binden von Funktionen

dynamic binding

- ▶ Zweite Möglichkeit: dynamisches Binden der Funktionsaufrufe zur Laufzeit
- ▶ Da die Referenz (myPet) auf das Objekt max vom Typ Dog zeigt, wird die Methode die von Dog definiert wurde aufgerufen

```
1  class Animal {  
2      public void makeSound() {  
3          System.out.println("sound.wav");  
4      }  
5  }  
6  class Dog extends Animal {  
7      public void makeSound() {  
8          System.out.println("wuff");  
9      }  
10 }  
11 class Cat extends Animal {  
12     public void makeSound() {  
13         System.out.println("miau");  
14     }  
15 }
```

Java

```
1  Dog max = new Dog();  
2  Animal myPet = max;  
3  
4  myPet.makeSound();
```

Java



Dynamisches und Statisches Binden

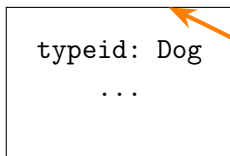
Dynamic (late) binding: die unterste Funktion in der tatsächlichen Hierarchie wird aufgerufen

Static binding: die Methode des Typs wird aufgerufen
early binding: compiler knows the type and can look up the function directly

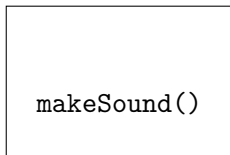
Dynamic Binding

- ▶ Wie funktioniert dynamisches Binden?
- ▶ vtable lookup und dynamic function dispatch

Speicher von max



vtable von Dog

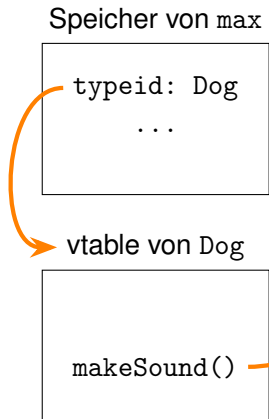


```
1  class Dog extends Animal {  
2      public void makeSound() {  
3          System.out.println("wuff");  
4      }  
5  }  
6  
7  Dog max = new Dog();  
8  Animal myPet = max;  
9  myPet.makeSound();
```

Java

Dynamic Binding

- ▶ Wie funktioniert dynamisches Binden?
- ▶ vtable lookup und dynamic function dispatch



```
1  class Dog extends Animal {  
2      public void makeSound() {  
3          System.out.println("wuff");  
4      }  
5  }  
6  
7  Dog max = new Dog();  
8  Animal myPet = max;  
9  myPet.makeSound();
```

Java



Diamantenproblem

Fahrzeug

Auto Boot

Amphibienfahrzeug



Graphische Modellierung (UML)

- ▶ + bedeutet public
- ▶ – bedeutet private

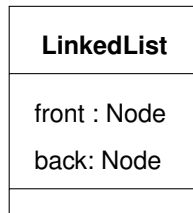
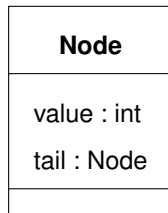
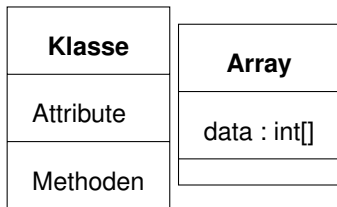
LinkedList
<ul style="list-style-type: none">– front : Node– back : Node
<ul style="list-style-type: none">+ sort()+ find(value : int) : bool+ append(value : int)+ delete(position : int)

```
1  class LinkedList {
2      private class Node {
3          ...
4      };
5
6      public LinkedList() { ... }
7
8      public void    sort() { ... }
9      public boolean find(int value) { ... }
10     public void    append(int value) { ... }
11     public void    delete(int position) { ... }
12
13     private Node front;
14     private Node back;
15 }
```

Java

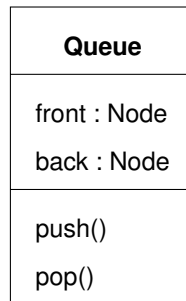
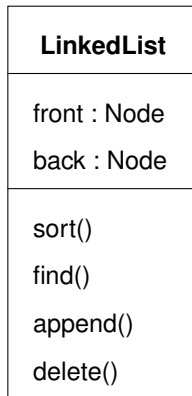
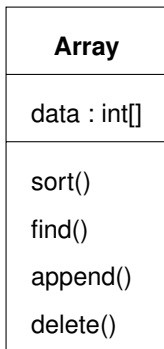
UML hier?

Modellierung von Klassen - UML - Klassendiagramme - Sequenzdiagramme,...



Klassen

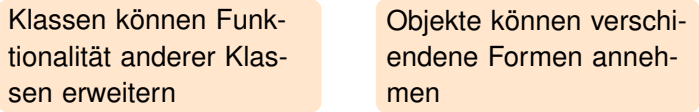
- ▶ Klassen gruppieren einen Zustand und ein Verhalten
- ▶ Das vereinfacht die Benutzung, weil im Kontext der Daten direkt die möglichen Interaktionen definiert werden



Zusammenfassung

Kapselung Abstraktion Vererbung Polymorphie

Klassen können Funktionalität anderer Klassen erweitern



The diagram consists of two orange rectangular boxes with rounded corners. The left box contains the text 'Klassen können Funktionalität anderer Klassen erweitern'. The right box contains the text 'Objekte können verschiedene Formen annehmen'. From the top-right corner of the left box, an orange arrow curves upwards and to the right, pointing towards the word 'Vererbung' in the header. From the top-left corner of the right box, an orange arrow curves upwards and to the left, pointing towards the word 'Polymorphie' in the header.

Objekte können verschiedene Formen annehmen