ingenieur
wissenschaften
htw saar

Hochschule für
Technik und Wirtschaft
des Saarlandes
University of
Applied Sciences

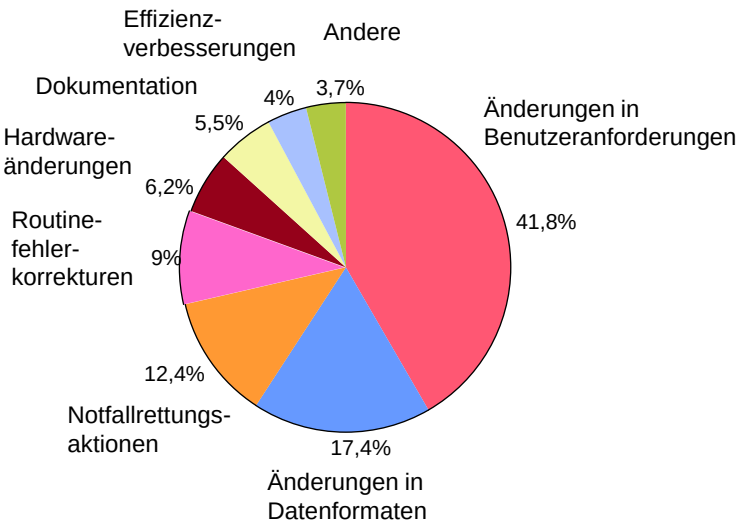
Einführung in objektorientierte Programmierung

Prof. Dr. Helmut G. Folz

Probleme bei der klassischen Software-Entwicklung

- Ständig steigende Wartungsaufwendungen
- Zunehmende Komplexität der Software
- Produktivität der Entwickler steigt nicht schnell genug

Zusammensetzung der Softwarewartungskosten



Probleme bei der klassischen Software-Entwicklung

Schlussfolgerung:

- Erhöhung der Produktivität bei der Softwareentwicklung durch Minimierung der Wartungsaufwendungen wie z. B.
  - ⇨ Erweiterungen,
  - ⇨ Fehlerkorrekturen, ...
- Software ist also so zu entwickeln, dass das Korrigieren und Erweitern leichter wird

Grundforderung: Korrektheit

- **Korrektheit** ist die Fähigkeit von Software-Produkten, ihre Aufgabe exakt zu erfüllen, so wie sie durch Anforderungen und Spezifikationen definiert sind.
  - Die Anforderungen und Spezifikationen müssen korrekt erfüllt sein.
  - Kann man Software so entwickeln, dass es leichter ist, korrekte Software zu schreiben?
    - Die "besten" Fehler sind die, die man gar nicht erst einbaut, die "zweitbesten" die, die der Compiler bereits findet.
    - Die nächstbesten Fehler sind die, die ein Programm zur Laufzeit selbst herausfindet!

Grundforderung: Robustheit

- **Robustheit** heißt die Fähigkeit von Software-Systemen, auch unter außergewöhnlichen Bedingungen zu funktionieren.
  - Wie verhält sich eine Software in unvorhergesehenen Situationen, wie z. B. Arithmetiküberlauf, Speicherfehler, Netzwerkfehler?
  - Wird die Behandlung von Ausnahmesituationen von der Programmiersprache bzw. der Entwicklungsumgebung unterstützt?

Grundforderung: Erweiterbarkeit

- **Erweiterbarkeit** bezeichnet die Leichtigkeit, mit der Softwareprodukte an Spezifikationsänderungen angepasst werden können.
  - Software muss von vornherein so erstellt werden, dass sie leicht erweitert werden kann.
  - Das Erweitern von Software ist nicht die Ausnahme sondern die Regel!

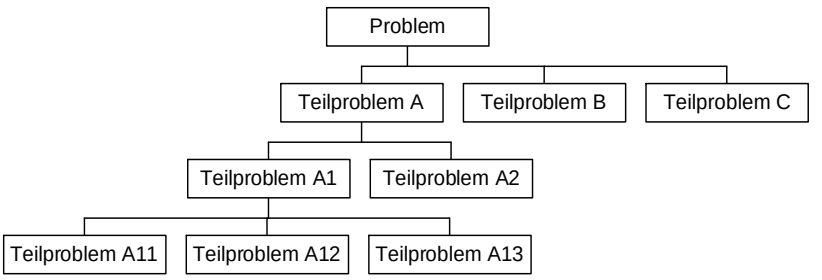
Grundforderung: Wiederverwendbarkeit

- Die **Wiederverwendbarkeit** von Software-Produkten ist die Eigenschaft, ganz oder teilweise für neue Anwendungen wiederverwendet werden zu können.
  - Wie sieht das Wiederverwenden in der Praxis eigentlich aus?
    - Wiederverwenden von Sourcecode weit verbreitet aber problematisch
    - Wiederverwenden von Funktionen
    - Wiederverwenden von übersetzten Programmteilen, Unterprogrammen
  - Hauptproblem beim Wiederverwenden: Das Wiederfinden!

Strukturierte Programmierung

- Anfang der 70er Jahre durch Niklaus Wirth mit der Programmiersprache Pascal eingeführt
- Programm = Algorithmus + Datenstruktur
- Top-Down-Ansatz
  - Zerlege Problem in Teilprobleme
  - Zerlege diese so lange bis Teile in Form von Anweisungen, Prozeduren oder Funktionen abgebildet werden können

Strukturierte Programmierung



Strukturierte Programmierung

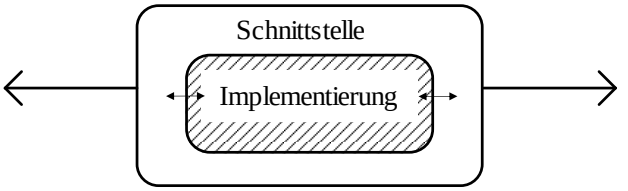
- **Nachteile:**
  - ⇨ Bei größeren Programmsystemen sind Korrekturen oder Erweiterungen aufwändig und problematisch, weil nur schwer abzusehen ist, welche und wieviele Teile eines Programmsystems betroffen sind.
  - ⇨ Die ***Wartbarkeit*** und die ***Erweiterbarkeit*** von herkömmlich entwickelten Softwaresystemen stellen ein generelles Problem dar.
  - ⇨ Die ***Wiederverwendbarkeit*** schon entwickelter Lösungen im Rahmen neuer Softwareprojekte ist begrenzt

Modulare Programmierung

- Aufgliederung eines Softwaresystems in Module
- Module bestehen aus einem Schnittstellenteil und einem Implementierungsteil
- Prinzip der **Datenkapselung**
  - Die Datenelemente eines Moduls können nur über die Schnittstelle erreicht werden und können nur durch eine begrenzte Anzahl an Funktionen und Prozeduren manipuliert werden.
- **Geheimnisprinzip**:
  - Die Daten und Datenstrukturen eines Moduls sind geheim und können nur über die Modul-Schnittstelle gelesen oder verändert werden.

Modulare Programmierung

- Modul mit gekapselter Implementierung und öffentlicher Schnittstelle.



Modulare Programmierung

- **Vorteile:**
  - ⇨ Einzelne Systemteile sind leichter austauschbar.
  - ⇨ Module sind leichter bei anderen Programmsystemen wiederzuverwenden.
  - ⇨ Durch die Datenkapselung lassen sich unerwünschte Seiteneffekte stark einschränken.

Modulare Programmierung

- **Nachteile:**
  - ⇨ Die ***Wiederverwendbarkeit*** eines Moduls ist dann eingeschränkt, wenn es in veränderter oder erweiterter Form wiederverwendet werden soll.
  - ⇨ Module sind vor allem gegenüber Veränderungen von Datenstrukturen empfindlich. Dies kann zu einer unerwünschten Codevervielfachung führen, die wiederum die ***Wartbarkeit*** des Softwaresystems erheblich herabsetzt.

prozedural - objektorientiert

- Strukturierte und modulare Programmierung
  - ⇨ Denken in Abläufen
  - ⇨ reale Problemstellung Schritt für Schritt in Algorithmen umsetzen
  - ⇨ Beispiel betrieblicher Prozess -> Automatisierung
- Objektorientierung
  - ⇨ entspricht mehr der menschlichen Denkweise
  - ⇨ Identifizierung realer Objekte aus der abzubildenden Umwelt
  - ⇨ Beschreibung in ihrer Art
  - ⇨ Bildung von Kategorien
  - ⇨ Darstellung von Zusammenhängen
  - ⇨ Ableitung von neuen Objekten aus bekannten Kategorien

Objekte und Klassen

Was haben diese realen Objekte gemeinsam?

Objekte und Klassen

reales Objekt
- Eigenschaften
- Verhaltensweisen

SW-Objekt
- Attribute
- Methoden

Auto
Attribute: - besitzer - typ - farbe - geschwindigkeit
Methoden: + fahren + bremsen

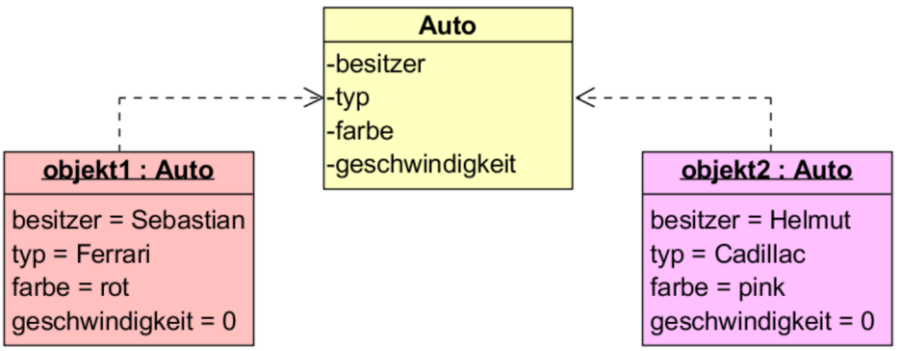
Objekte und Klassen

Definition:

- Eine ***Klasse*** beschreibt eine Menge von Objekten mit gleichen Eigenschaften, gleichen Verhalten, gemeinsamen Beziehungen zu anderen Objekten und gemeinsamer Semantik.
- Eine Klasse definiert also die Eigenschaften und das Verhalten ihrer Objekte.

Objekte und Klassen

- **Klasse** vollständig beschrieben durch:
  - ⇨ **Name**, **Zustand** (statische Eigenschaften), **Verhalten** (dynamische Eigenschaften)
- **Zustand = Attribute**



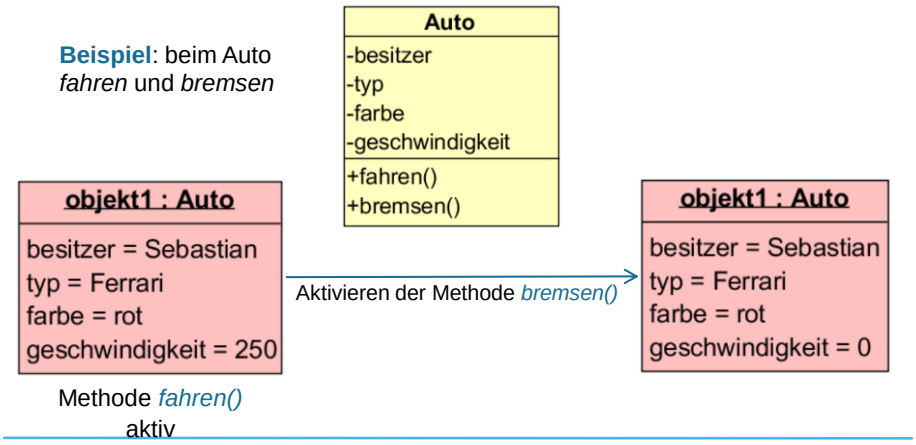
aktueller Zustand ⇔ Werte der Attribute

Objekte und Klassen

- **Verhalten** sind die möglichen Aktivitäten eines Objekts

Methoden oder **Operationen**

Beispiel: beim Auto *fahren* und *bremsen*



Methode *fahren()* aktiv

Objekte und Klassen

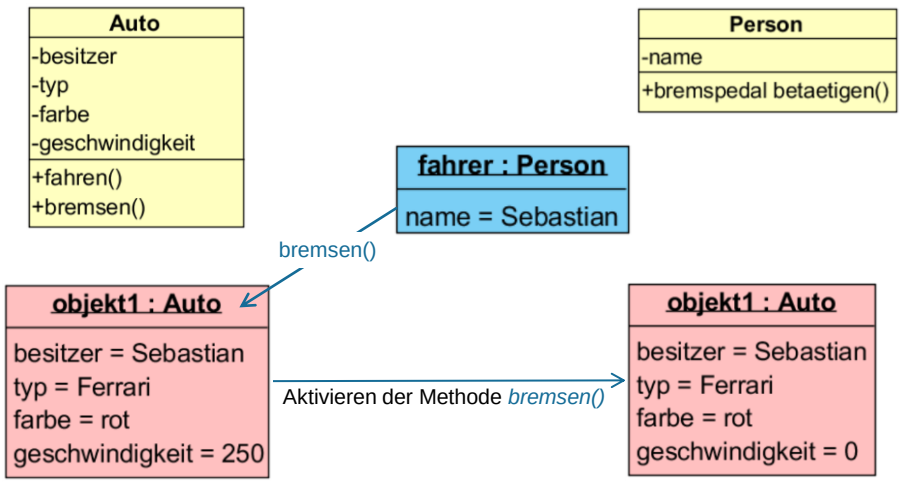
- **Objekte**
  - ⇨ repräsentieren „Dinge“ der realen Welt
  - ⇨ Beispiel: „Sebastians roter Ferrari fährt 250 km/h“
- **Klassen**
  - ⇨ repräsentieren alle Objekte einer bestimmten Art
  - ⇨ Beispiel: „Auto“
- **Methoden**
  - ⇨ Objekte haben Operationen, die aufgerufen werden können
  - ⇨ Operationen heißen in Java Methoden

Objekte und Klassen

- **Wichtige Prinzipien: Kapselung und Geheimnisprinzip**
  - ⇨ Zusammenfassung von Name, Zustand und Verhalten
  - ⇨ Vollständige Beschreibung eines Objekts
  - ⇨ Kommunikation zwischen Objekten mittels Nachrichten
- Änderung des Zustands eines Objekts
  - ⇨ Kein direkter Eingriff zur Änderung eines Attributwertes !!!
  - ⇨ Änderung nur über eine Methode
- Beispiel: Sebastian will sein fahrendes Auto zum Stillstand bringen.
  - ⇨ Keine Zuweisung geschwindigkeit = 0
  - ⇨ Sondern: Anwendung der Methode bremsen()

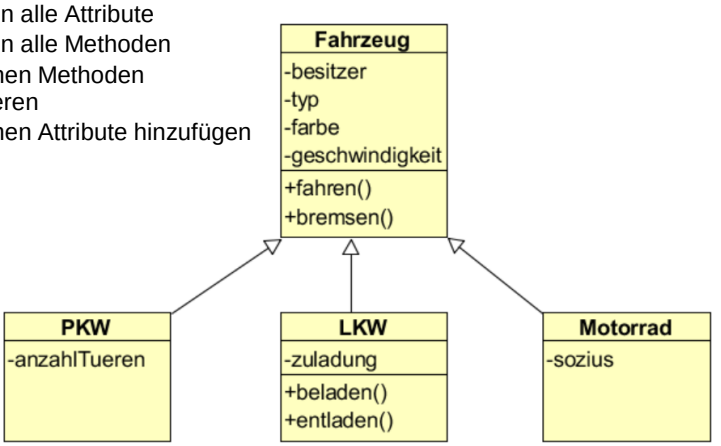
Nachrichten zwischen Objekten

- Beispiel: Methode bremsen() wird von anderem Objekt aufgerufen



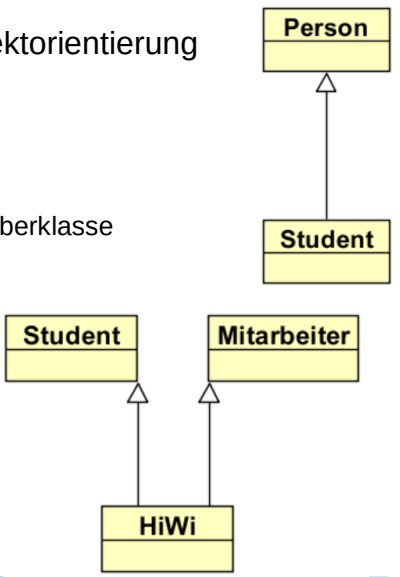
Vererbung

- Von einer allgemeinen Klasse werden Unterklassen abgeleitet
- Die Unterklassen haben folgende Eigenschaften
  - ⇨ Sie erben alle Attribute
  - ⇨ Sie erben alle Methoden
  - ⇨ Sie können Methoden redefinieren
  - ⇨ Sie können Attribute hinzufügen



Vererbung

- Wichtigstes Konzept der Objektorientierung
- Oberklasse/Unterklasse
  - ⇨ auch Superklasse/Subklasse
- **Einfachvererbung**:
  - ⇨ jede Unterklasse hat nur eine Oberklasse
- **Mehrfachvererbung**
  - mehrere Oberklassen möglich

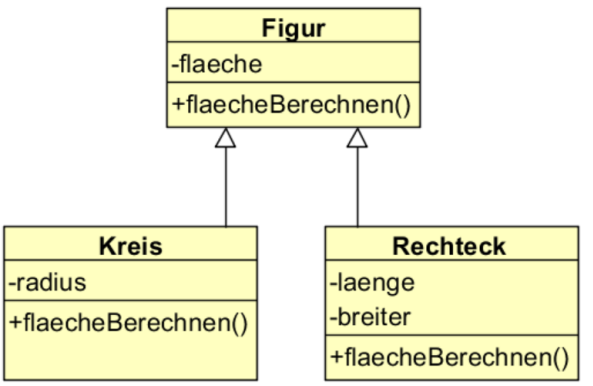


Polymorphismus und dynamisches Binden

- **Polymorphismus** = Vielgestaltigkeit
  - ⇨ Wichtiges Konzept der Objektorientierung
  - ⇨ erlaubt einem Objekt, auf ein und dieselbe Nachricht unterschiedlich zu reagieren
- **Dynamisches Binden**
  - ⇨ zur Laufzeit wird festgelegt, welche Methode durch eine bestimmte Nachricht ausgelöst wird

Polymorphismus und dynamisches Binden

- Beispiel: Klasse Polygon, Unterklassen Kreis, Rechteck



Überschreiben von Methoden

Grundprinzipien der Objektorientierung

- **Datenkapselung**: Die Daten und die dazugehörigen Operationen sind gekapselt in einer Programmeinheit
- **Geheimnisprinzip**: Die Daten sind nur innerhalb des Objekts bekannt. Der Zugriff erfolgt über die öffentlichen Operationen.
- **Vererbung**: Vererbung ist ein Mechanismus, bei dem eine Klasse als Spezialfall einer allgemeinen Klasse definiert wird.
  - ⇨ Dabei "erbt" die Unterklasse automatisch alle Attribute und Methoden der Oberklasse .
  - ⇨ Zusätzlich kann die Unterklasse weitere Attribute und Methoden hinzufügen und geerbte Methoden redefinieren.

Grundprinzipien der Objektorientierung

- **Polymorphismus**:
  - ⇨ Ein gegebenes Programmelement kann sich zur Laufzeit auf Objekte ganz verschiedener Klassen beziehen.
  - ⇨ Methoden von Objekten können unter gleichem Namen angesprochen werden, aber erst zum Zeitpunkt des Programmablaufs muss feststehen, zu welcher Klasse das Objekt gehört und welche Operation tatsächlich zur Ausführung kommt.
  - ⇨ Weil auszuführende Operationen erst zur Laufzeit „gebunden“ werden, heißt diese Technik auch *dynamisches Binden*.