

# Objekt-Orientierte Programmiersprachen

Martin Gasbichler  
Holger Gast

18. Oktober 2005

---

## Wir sind . . .

Holger Gast <gast@informatik.uni-tuebingen.de>

- 1975
- 1985
- 1993
- 1995
- 2000
- 2005
  
- Poirot
- Kurrende

---

## Wir sind . . .

Martin Gasbichler <gasbichl@informatik.uni-tuebingen.de>

- 1974
- 1994
- 2000
- 2004

Aktuell:

- Diese Vorlesung
- Diss
- Moritz

---

## Wir sind . . .

- Begeistert von Programmiersprachen
- Begeisterte Programmierer
- Mögen auch Theorie & Konzepte
- Haben Spaß an Lehre
- Statisch bzw. dynamisch (latent) getypt

---

## Wer seid Ihr?

- Welche objekt-orientierten Sprachen kennt ihr?
- Wieviel Programmiererfahrung habt ihr?
- Skala:
  - Kein Eintrag: Ich habe die Sprache noch nie benutzt
  - Anfänger: Sprache gerade eben gelernt
  - Studium: Sprache für Vorlesung & Übungen
  - Handwerker: Sicherer Umfang
  - Profi: Kennt die Tricks, Kniffe und Interna der Sprache

---

## Was wollt ihr hier lernen?

- Eure Erwartungen an die Vorlesung
- Anschließend: Vergleich mit Lernzielen

---

## (Lern-)Ziele

- Konzepte objekt-orientierter Programmiersprachen
  - kennenlernen
  - verstehen
  - wiedererkennen
  - anwenden
- Objekt-orientierte Sprachen
  - richtig auswählen
  - richtig verwenden
  - im Detail verstehen
  - schnell und sicher lernen
- Kommunikation über Programmiersprachen & Programme

---

## Was passiert hier?

```
class A {  
    private int i;  
    public A() {  
        i = 42;  
    }  
    public int add(int j) {  
        return i + j;  
    }  
}
```

- Klasse
- Felder für Daten
- Methoden für Operationen
- Konstruktor: Initialisierung
- Schutzmechanismen

## Was passiert hier?

```
class Cell {
    private int content;
    public void set(int content) {
        this.content = content;
    }
    public int get() {
        return content;
    }
}
```

- Namensbindung (KvP)
- Das this Objekt
- Zustand (KvP)

## Was passiert hier?

```
class User {
    protected Cell c;
    public User() {
        c = new Cell();
    }
    public void store(int i) {
        c.set(i);
    }
}
```

- Objekt-Erzeugung
- Konstruktor-Aufruf
- Objekt
- Instanz einer Klasse
- Methodenaufruf
- Schutzmechanismus: protected

## Was passiert hier?

```
class BackupCell extends Cell {
    protected int backup;
    public BackupCell() {
        super();
    }
    public void store(int i) {
        backup = content;
        super.set(i);
    }
    public void restore() {
        content = backup;
    }
}
```

- Vererbung
- Überschreiben von Methoden
- Aufruf überschriebener Methoden
- Super-Konstruktor
- Default-Wert (KvP)

## Was passiert hier?

```
public void m(Cell c) {
    c.set(42);
}
public void f() {
    Cell c = new Cell();
    m(c);
}
public void g() {
    BackupCell c = new BackupCell();
    m(c);
}
```

- Polymorphie
- Dynamische Bindung
- Statische Typen (KvP)
- Subtyping (KvP)
- Substitutability

## Erkennt ihr was wieder?

```
Object subclass: #Cell
  instanceVariableNames: 'content '
  classVariableNames: ''
  poolDictionaries: ''
  category: 'Intro'!
```

- Klassendeklaration
- Methodendeklarationen
- Felder
- Schutzmechanismus

```
!Cell methodsFor: '...' stamp: '...'!
get
  "get content of cell"
  ^ content! !

!Cell methodsFor: '...' stamp: '...'!
set: c
  "set content of cell"
  content _ c! !
```

## Erkennt ihr was wieder?

```
Cell subclass: #BackupCell
  instanceVariableNames: 'backup '
  classVariableNames: ''
  poolDictionaries: ''
  category: 'Intro'!
```

- Abgeleitete Klasse
- Methoden
- Super-Aufruf

```
!BackupCell methodsFor: '...' stamp: '...'!
restore
  "restore previous content"
  content _ backup! !

!BackupCell methodsFor: '...' stamp: '...'!
set: c
  "set and backup old value"
  backup _ content.
  super set: c! !
```

## Die Differenzen

- Klassendeklaration durch Vererbung
- Vererbung durch Methodenaufruf
- Klassen sind Objekte ([everything is an object](#))
- Dynamische Typen
- Schutzmechanismus ist Sichtbarkeit

## Fazit: OO-Konzepte

- Das war eine Auswahl von OO-Konzepten
- Einige Konzepte kennt ihr bereits, aber . . .
- . . . wahrscheinlich sind für jeden einige neue dabei
- . . . vielleicht habt ihr sie noch nicht wahrgenommen
- Die Konzepte tauchen in verschiedenen Sprachen auf
- Die Ausprägung und Details unterscheiden sich sehr

→ Es lohnt sich, die Konzepte zu studieren

---

## Warum sind Konzepte wichtig?

- Sprachen richtig verwenden
- Mächtigkeit von Sprachen erkennen und einschätzen
- Bekannte Sprachen genauer verstehen
- Neue Sprachen leichter lernen
  - Auf vorhandenes Wissen aufbauen
  - Bekanntes neu entdecken
  - Gezielt Fragen stellen
  - Sprachdefinitionen leichter lesen
- Kommunikation über Sprachen & Programme
- Sprachdefinitionen erklären keine Konzepte

---

## Vergleich Erwartungen

- Welche Erwartungen werden erfüllt?
- Welche werden nicht erfüllt?

---

## Übersicht: Wie könnt ihr das lernen?

- Teil I: Einführung
- Teil II: Java en detail
- Teil III: Klassenbasierte Sprachen
- Teil IV: Objekt-basierte Sprachen
- Teil V: OO in funktionalen Sprachen

---

## Teil I: Einführung

- OCaml (eine funktionale Sprache)
  - Wir schreiben nachher ein großes OCaml Programm
  - OO-Sprachen sind nicht immer die beste Wahl
  - Es schadet nie, etwas anders zu kennen
- Objekt-orientierte Programmierung (nur Zusammenfassung)
  - Motivation für OO-Sprachkonzepte
  - Was soll eigentlich erreicht werden?
  - Welche Ausdrucksmittel werden benötigt?
  - Warum ist OO eigentlich so cool?
  - Sprachbezogen, nicht problembezogen
- Konzepte von Programmiersprachen (als Grundlage)

---

## Teil II: Java en detail

- Eine Sprache exemplarisch genau untersuchen
  - Praxis ist wichtig → wir nehmen Java
  - Verständnis der Konzepte
  - Ziel: Ihr versteht, **warum** die Sprachspezifikation so ist
  - Wir wollen auch die Details verstehen
- ⇒ Wir schreiben einen Compiler für Java

---

## Teil III: Klassenbasierte Sprachen

- Übertragung der Konzepte
- Sprachen:
  - C++ (en detail)
  - Eiffel / Sather

---

## Teil IV: Objekt-basierte Sprachen

- Man braucht eigentlich keine Klassen
- Sprachen:
  - JavaScript
  - VBA
  - Smalltalk
    - Aber das hat doch Klassen!
    - Aber: **Everything is an object** – auch Klassen

---

## Teil V: OO in funktionalen Sprachen

- Rückführung von OO-Konzepten auf funktionale Konzepte
  - Eigene Implementierung
  - Erlaubt Experimentieren
- Sehr elegant!
- $OO \approx MPS + Records + Closures + Update$

---

## Es ist noch kein Meister vom Himmel gefallen . . .

. . . und noch kein Informatiker in einer Vorlesung zum OO-Profi geworden.

Für diese Vorlesung braucht ihr:

- Fundierte Programmiererfahrung in einer OO-Sprache
- Bereitschaft, neue Sprachen zu lernen
- Bereitschaft, sich auf neue Konzepte einzulassen
- Man bekommt den Schein **nicht** allein für Java Kenntnisse!

## Scheinkriterien

- Bearbeitung der Übungsblätter für den Schein
- Es müssen 50% der Punkte erreicht werden
- Aufgaben für 10 Punkte müssen in der Übung präsentiert werden.
- Das letzte Blatt ist ein Pflichtblatt, das zur Zufriedenheit des Tutors bearbeitet werden muss.
- Die ersten Blätter werden nicht bepunktet.
- Teams mit bis zu 3 Mitgliedern
- Benotete Scheine (nur Master & Nebenfächler): Mündliche Prüfung

---

## Organisation

- Wöchentliche Übungsblätter
- Wöchentliche zentrale Übung für Fragen und Lösungen
- Wir haben lange gesucht und einen (!) Tutor gefunden:  
Friedrich Meißner <friedrich.meissner@student.uni-tuebingen.de>

- Uns erreicht ihr
  - Martin <gasbichl@informatik.uni-tuebingen.de>
  - Holger <gast@informatik.uni-tuebingen.de>
  - Sprechstunde jederzeit nach Email-Anmeldung
- <http://www-pu.informatik.uni-tuebingen.de/oopl-0506/>
- Bitte regelmäßig anschauen

## Scheinkriterien

- Ihr seid viele
- Wir haben nur einen Tutor
- Das klassische Übungsmodell funktioniert nicht
- Meldet sich noch jemand von euch als Tutor?
- Scheinkriterien werden am Donnerstag bekannt gegeben.
- Meldet Euch bitte zur Vorlesung an!  
(Wird ab morgen früh freigeschaltet.)

---

## The End

- Hoffentlich haben wir Euch
  - informiert
  - interessiert
  - motiviert
- Und ihr kommt nächstes Mal wieder

Bis Donnerstag!