

Vortrag am 06. Februar 2004
Zeitansatz 60min (+180min programmieren)
Werner-von-Siemens Gymnasium Magdeburg



CASE, OOP, UML

Schwerpunkt: Klassendiagramme

Ein Vortrag von Markus Durzinsky

Student der Otto-von-Guericke-Universität
Magdeburg

www-e.uni-magdeburg.de/durzinsk
Mardur@gmx.net



0. Gliederung

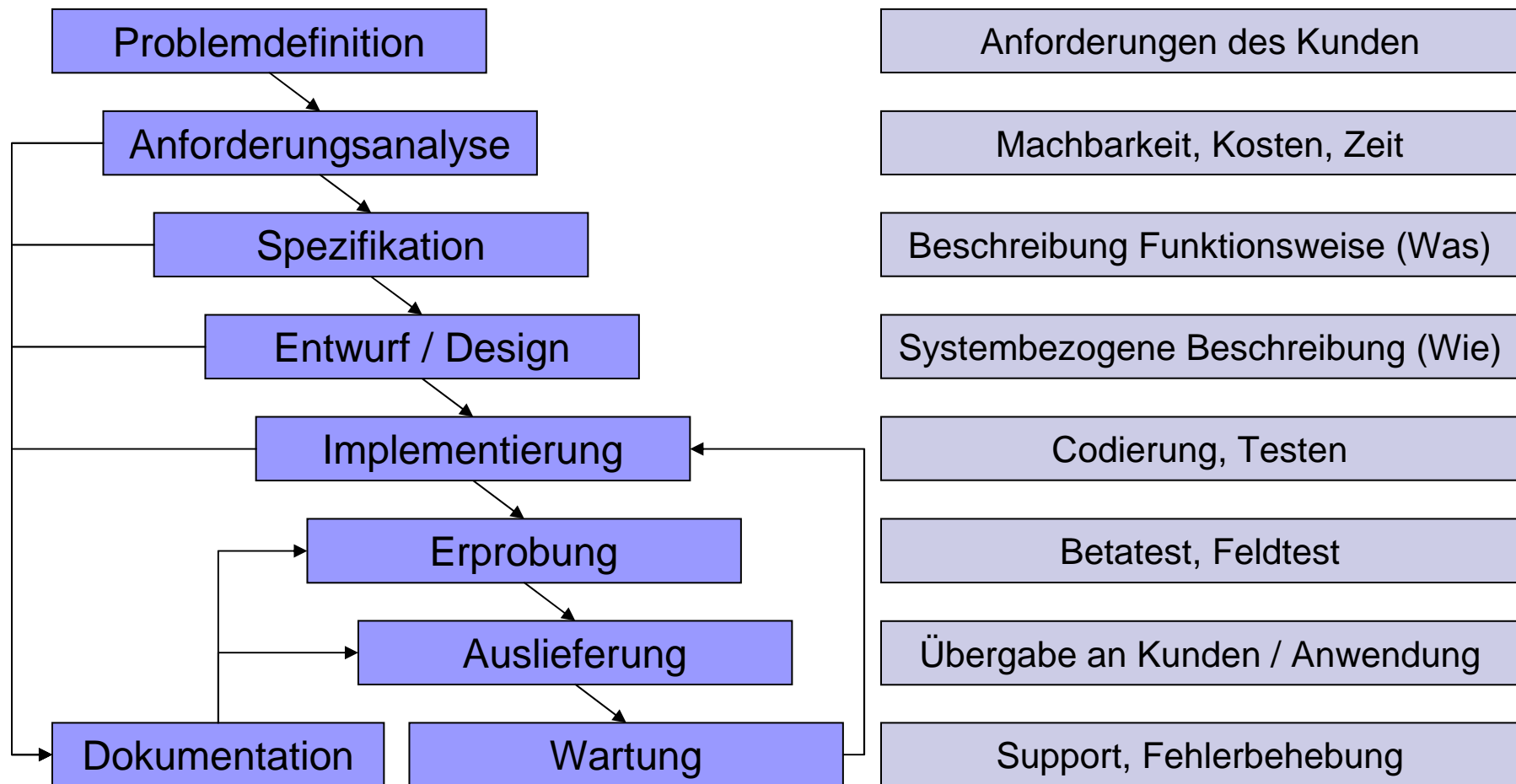
1. CASE – Computer Aided Software Engineering
2. Abstraktion und Modellierung
3. UML – Unified Modelling Language
4. OOP – Objektorientierte Programmierung
5. Klassendiagramme in UML
6. Programmieraufgabe
7. Zusammenfassung
8. Internetadressen / Literatur




1. Software Engineering

- Standards ISO 9126, POSIX, ANSI
- Werkzeuge JBuilder, Rational Rose
- Methoden UML, Aufwandsschätzung
- Maßsystem Speicherplatz, Laufzeit
- Ressourcen Personal, Hard/Software
- Erfahrung

Software-Lebenszyklus

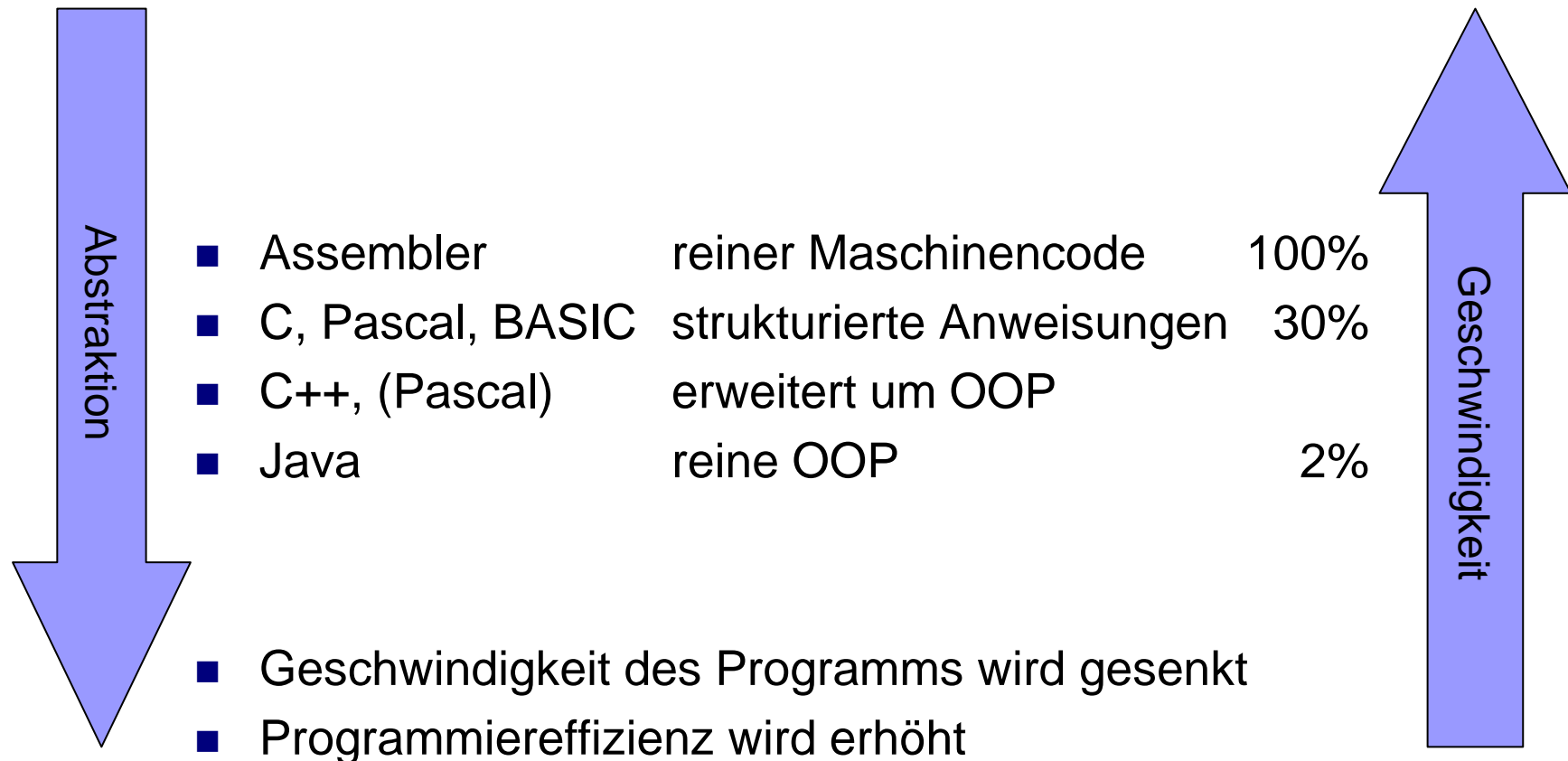




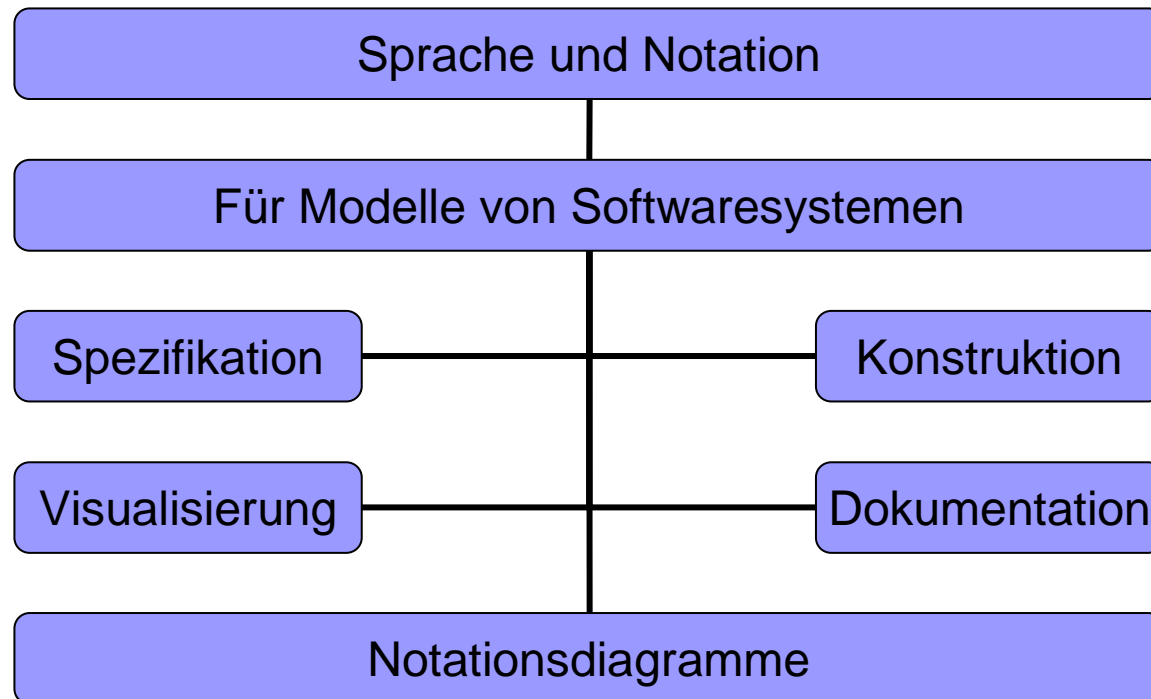
2. Wozu Modellierung und Abstraktion?

- Abstraktion dient der Vereinfachung
- Komplexe Zusammenhänge bleiben überschaubar
- Modellierung zur Abstraktion und schematischen Darstellung

Abstraktionsgrad von Programmiersprachen



3. UML – Unified Modelling Language

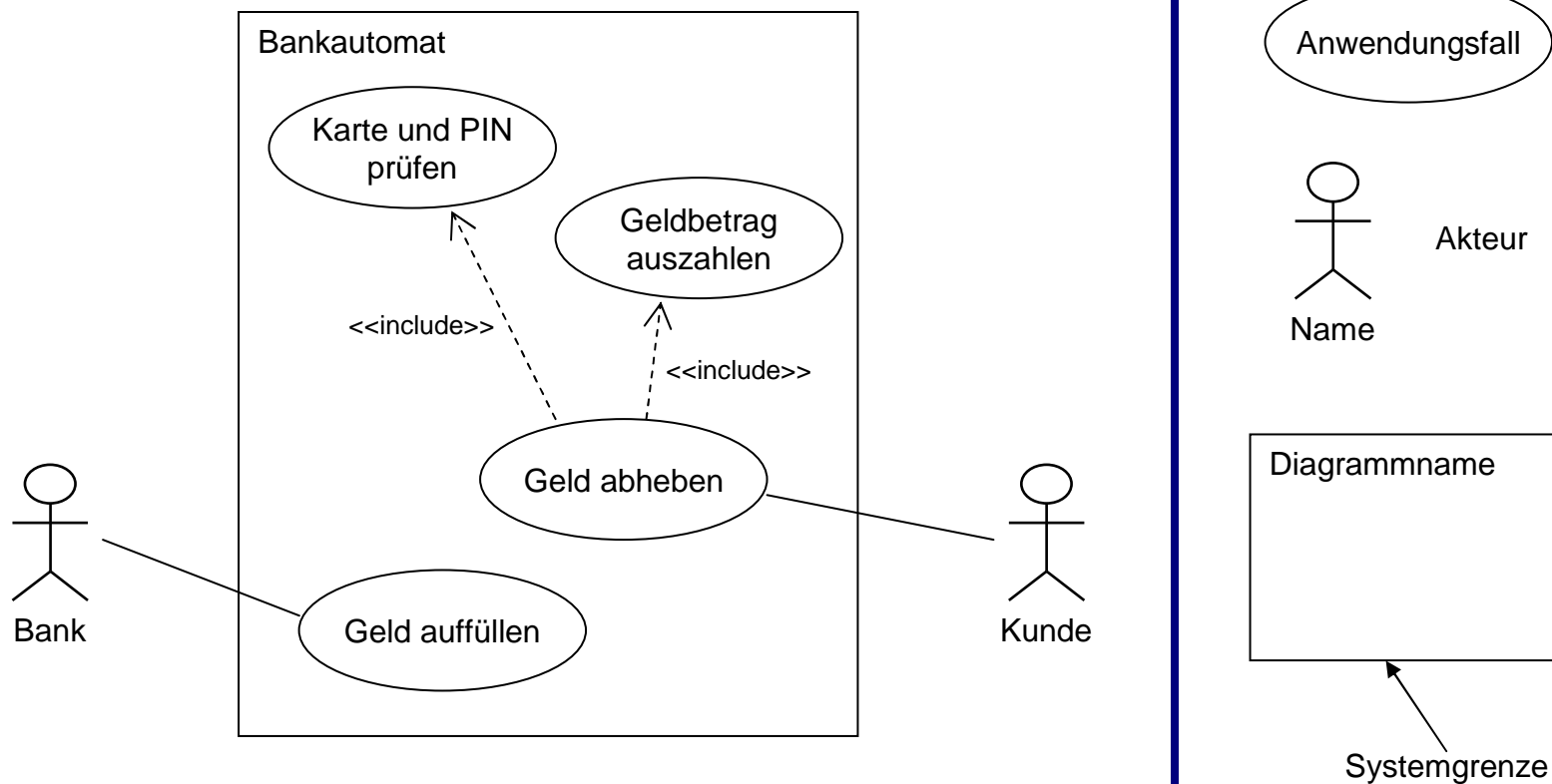




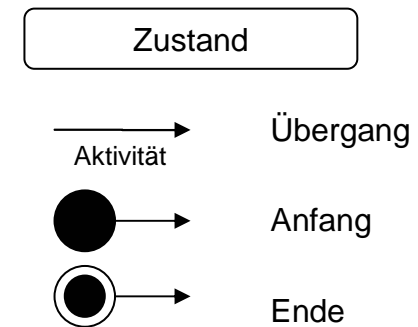
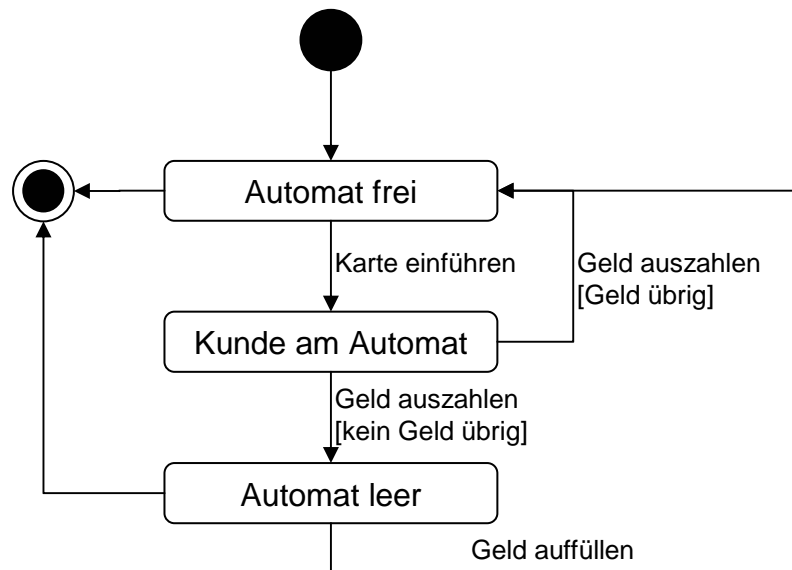
Diagrammtypen

- Anwendungsfalldiagramm (use case)
- Verhaltensdiagramme
 - Zustandsdiagramm
 - Aktivitätsdiagramm
 - Sequenzdiagramm
 - Kollaborationsdiagramm
- Implementierungsdiagramme
 - Komponentendiagramm
 - Verteilungsdiagramm
- Klassendiagramm

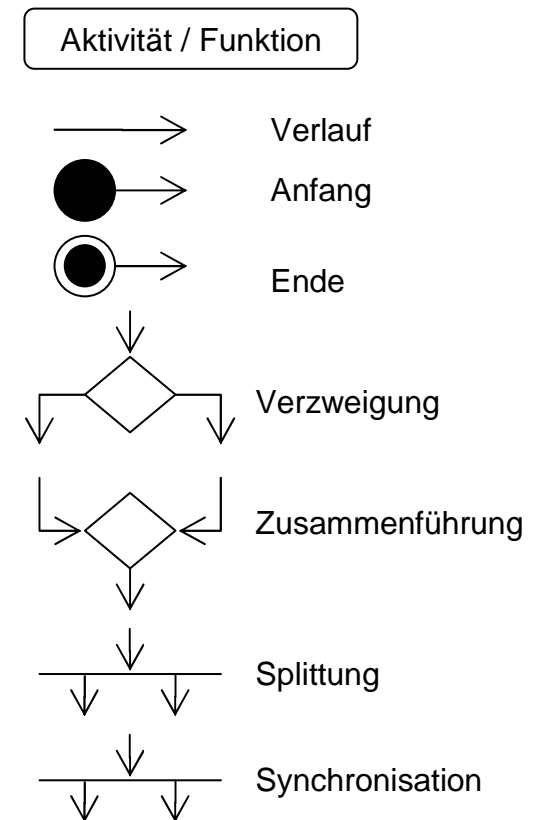
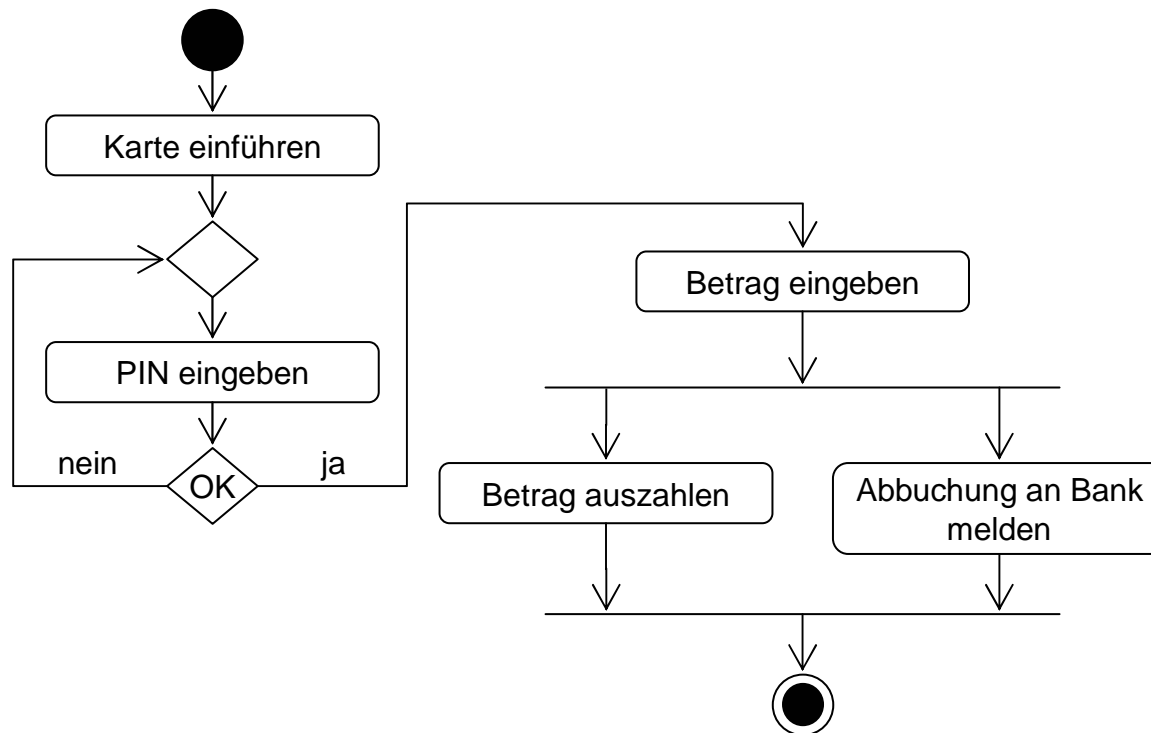
Anwendungsfalldiagramm




Zustandsdiagramm



Aktivitätsdiagramm





4. Crashkurs in abstrakten Datentypen, Datenstrukturen

■ Felder

- mehrere Werte gleichen Typs

```
A : array[0..10] of Integer;
```

```
for i := 0 to 10 do  
  A[i] := 2 * i;
```

■ Datensätze

- mehrere Werte unterschiedlichen Typs

```
type NeuerType : record  
  x, y : Integer;  
  str : String;  
end;
```

```
var r : NeuerType;
```

```
r.x := 20;  
r.str := 'Hello World';
```



Objekte und Klassen

- Objekte repräsentieren Datensätze, erweitert um eigene Funktionen

```
type NeueKlasse = class

    EigenschaftA : Integer;
    EigenschaftB : Real;

    function MethodeA : Integer;
    procedure MethodeB(a : Real; b : Real);

end;

obj := NeueKlasse.Create; { obj ist ein Objekt
                           { bzw. Instanz von NeueKlasse }
obj.EigenschaftA := 1024;
obj.MethodeB(1.234, 2.6432e-6);
```



Beispiel einer Klasse

■ Definieren eines Fahrzeuges

```
type Fahrzeug = class
  farbe : String;
  motorAn : Boolean;
  procedure starteMotor; virtual;
  procedure stoppeMotor; virtual;
end;
```

```
procedure Fahrzeug.starteMotor;
begin
  motorAn := true;
  writeln('Motor gestartet');
end;
```

```
procedure Fahrzeug.stoppeMotor;
begin
  motorAn := false;
  writeln('Motor gestoppt');
end;
```



Vererbung

■ ein spezielles Fahrzeug durch Vererbung

```
type Cabriolet = class(Fahrzeug)
  verdeckOffen : Boolean;
  procedure oeffneVerdeck; virtual;
  procedure stoppeMotor; override;
  { überschreibt Methode aus Klasse Fahrzeug }
end;

procedure Cabriolet.oeffneVerdeck;
begin
  verdeckOffen := true;
end;

procedure Fahrzeug.stoppeMotor;
begin
  verdeckOffen := false;
  inherited stoppeMotor;
end;
```



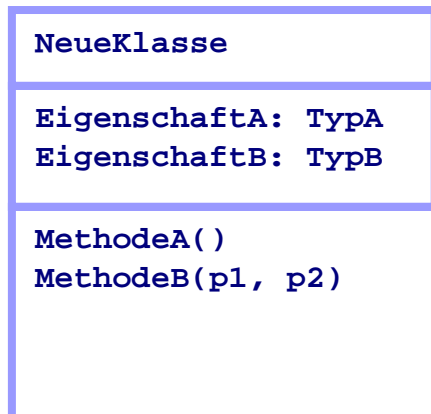
Casting und Polymorphie

■ Entscheidung fällt zur Laufzeit

```
procedure polizeikontrolle(wagen : Fahrzeug);  
begin  
    wagen.stoppeMotor;  
end;  
  
var cabrio : Cabriolet;  
  
Cabrio = Cabriolet.Create;    { neue Instanz erstellen }  
cabrio.starteMotor;  
  
polizeikontrolle(cabrio);    { Casting von Cabriolet nach Fahrzeug }  
  
Was ist cabrio.verdeckOffen ?    true / false
```

5. Klassendiagramme

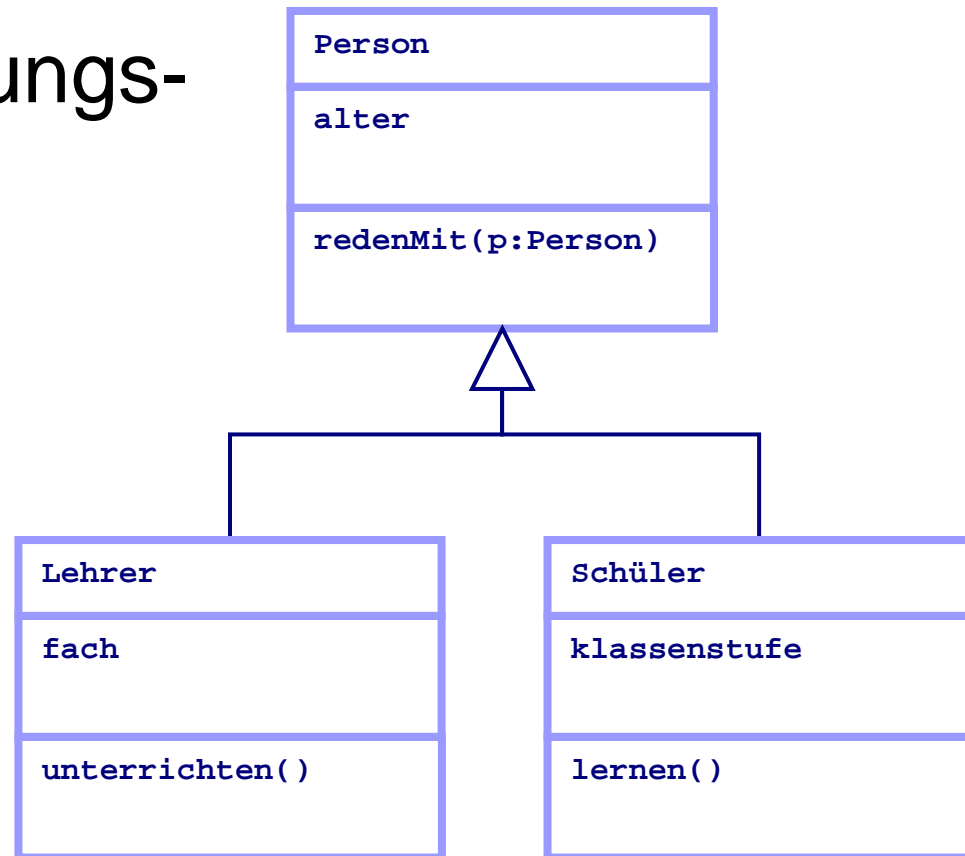
- Darstellung der Klassen des Projektes
- Relationen und Verwandtschaften zwischen Klassen



```
type NeueKlasse = class { oder object }  
  
    EigenschaftA: TypA;  
    EigenschaftB: TypB;  
  
    procedure MethodeA;  
    procedure MethodeB(p1: Typ1; p2: Typ2);  
  
end;
```

Vererbung von Klassen

- stellt eine Vererbungshierarchie dar

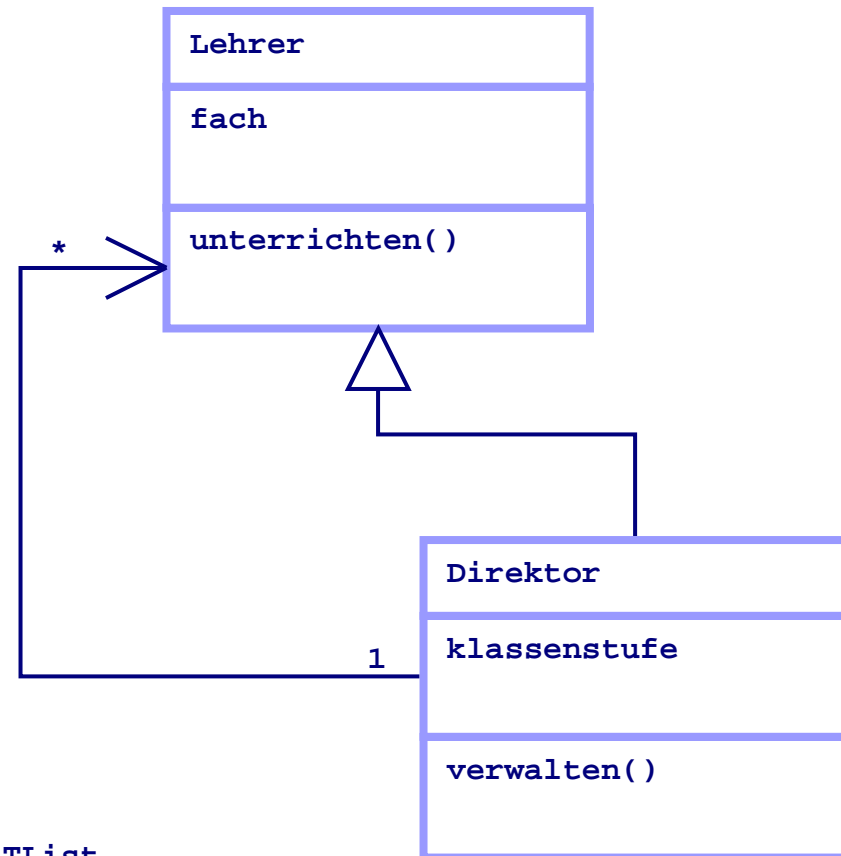


Assoziationen

- Vernetzung von Klassen zur Kommunikation

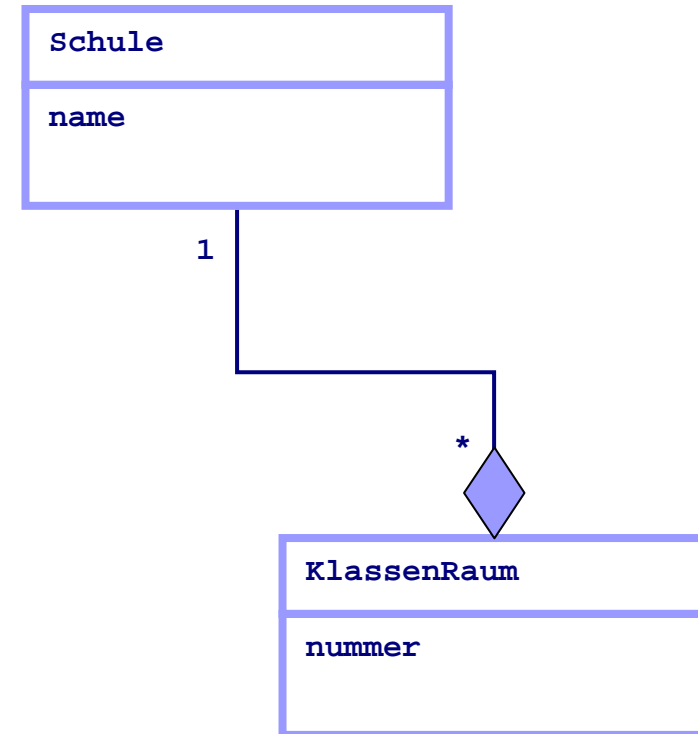


- implementiert als Eigenschaft von Direktor, z.B. `lehrerListe : TList`

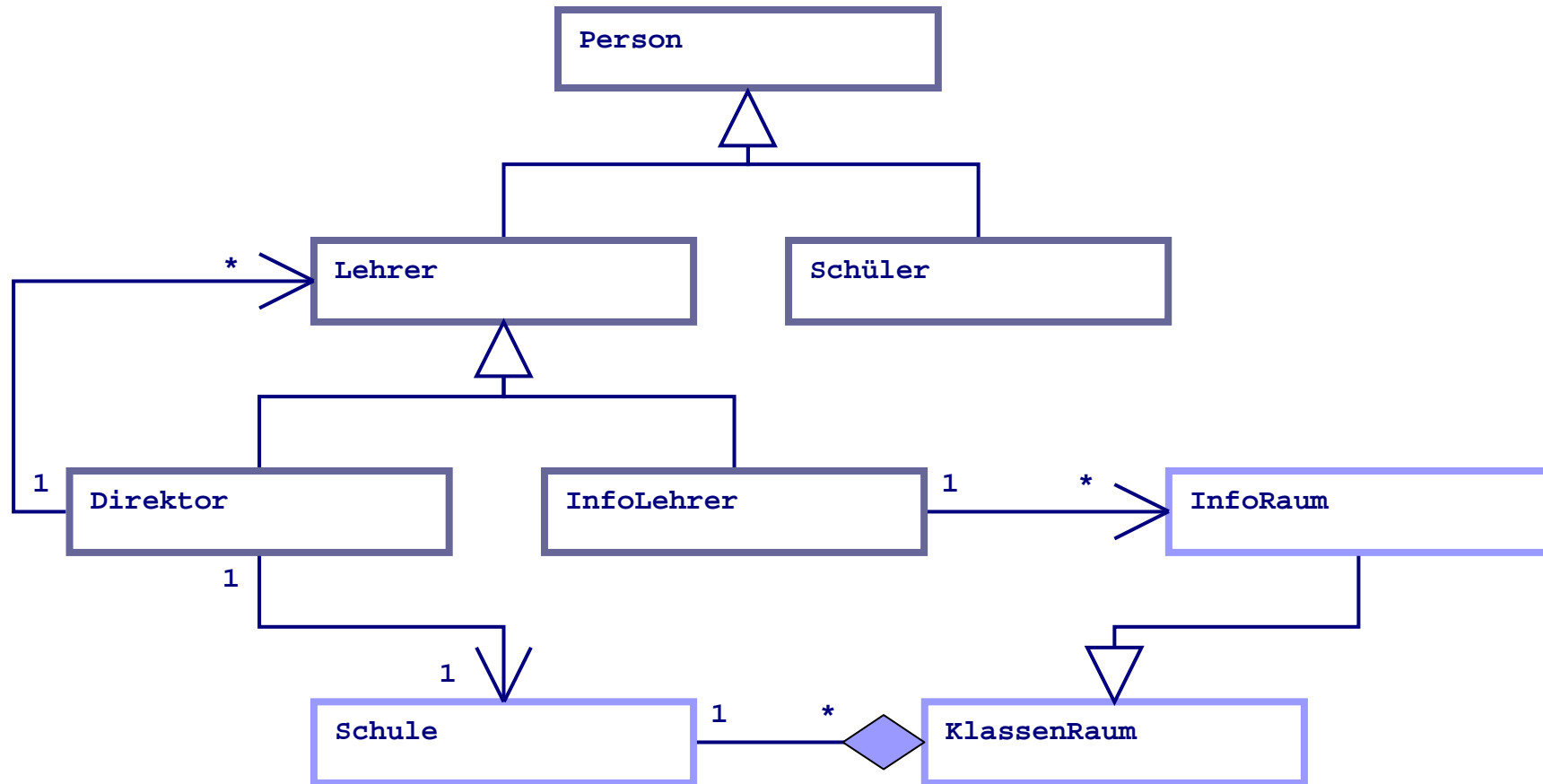


Aggregation

- stellt einen wesentlichen Bestandteil dar



komplexes Beispiel der Aufbau einer Schule

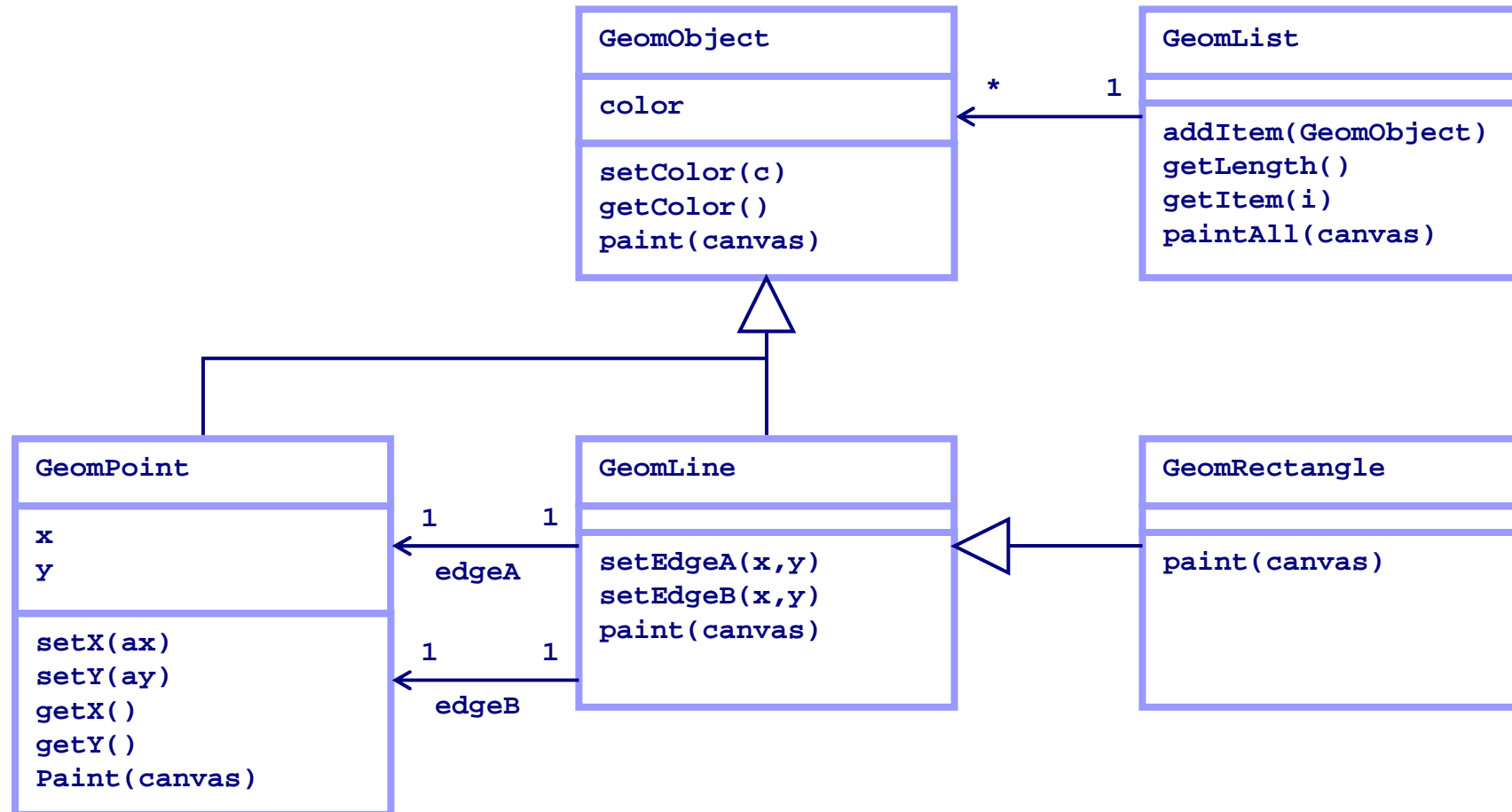




6. Programmieraufgabe

- man Modelliere folgendes System
 - eine Superklasse für geometrische Objekte
 - Klassen für verschiedene Formen (Punkt, Linie, Kreis, etc.), die von der Superklasse erben
 - eine Klasse zum Verwalten der Instanzen
- Welche Methoden/Eigenschaften sollte die Superklasse enthalten?
- Programmieren einer Anwendung zum Erstellen und Zeichnen der Objekte

Lösungsidee





7. Zusammenfassung

- Prinzipien der Objektorientierung
 - Vererbung (inheritance)
 - Polymorphismus
 - Kapselung (encapsulation, data hiding)
 - Abstraktion
 - Modularisierung
- Objektorientierte Programmierung liegt nahe an realen Gegebenheiten (Bsp. Schule)
- UML ist Standard zur Modellierung dieser Gegebenheiten



8. Internetadressen / Literatur

- www.omg.org/uml OMG Unified Modelling Language
 - www.jeckle.de Informationen und Neuigkeiten
 - www.oose.de/uml UML-Notationsübersicht
 - www.oose.de/oep Object Engineering Process
-
- Oestereich, Bernd; Die UML-Kurzreferenz für die Praxis; München, Wien; Oldenbourg Wissenschaftsverlag, 2001
 - Balzert, Heide; Objektorientierung in 7 Tagen; Heidelberg, Berlin; Spektrum Akademischer Verlag, 2000



Informationen zum Vortrag

- der Vortrag wurde gehalten im Rahmen des Spezialistenlagers Informatik am Werner-von-Siemens-Gymnasium (2. bis 6. Februar 2004)
- für den Vortrag ist eine Zeitdauer von 1,5 bis 2 Stunden geplant
- im Anschluss sollte eine Programmieraufgabe gestellt werden, die von den Schülern innerhalb von 2 Stunden selbstständig bearbeitet wird
- zu bedenken ist die informatische Vorbildung der Schüler
 - programmieren in Delphi
 - Kenntnisse über
 - Kontrollstrukturen (if, for, while)
 - Funktionen, Prozeduren
 - **keine** Kenntnisse über :-(
 - höhere Datenstrukturen
 - OOP