Java-Programmierung

BA

Grundbegriffe der OOP
mit Java-Beispiel

Autor: Dipl.-Math.
E. Engelhardt

Stand: 16. Mai 2009

Inhalt

1 Inhalt
- 1.1 Kritik an bisheriger Heran-gehensweise
- 1.2 Prozedurale Herangehensweise

2 Was ist objektorientierte Programmierung
- 2.1 Grundidee der OOP
- 2.2 Prinzipien der OOP
  - 2.5.1 Objekte und Klassen
  - 2.5.2 Vererbung und Ableitung
  - 2.5.3 Prinzip der Kapselung
  - 2.5.4 Funktions-Überladung
  - 2.2.5 Polymorphismus

3 OOP in Java
- 3.1 Klassen in Java
- 3.2 Interfaces
- 3.3 Objekte
- 3.4 Methoden

4 OOP in Java am Beispiel
- 4.1 Hauptprogramm AutoFahren
- 4.2 Was soll Java?
- 4.3 Ein C-Programm ausführen
- 4.4 Ein Java-Programm ausführen
- 4.5 „Hallo Welt"-Applikation
- 4.6 „Hallo Welt"-Applet
- 4.7 Allgemeine Form einer Klasse
- 4.8 Zugriffsberechtigungen

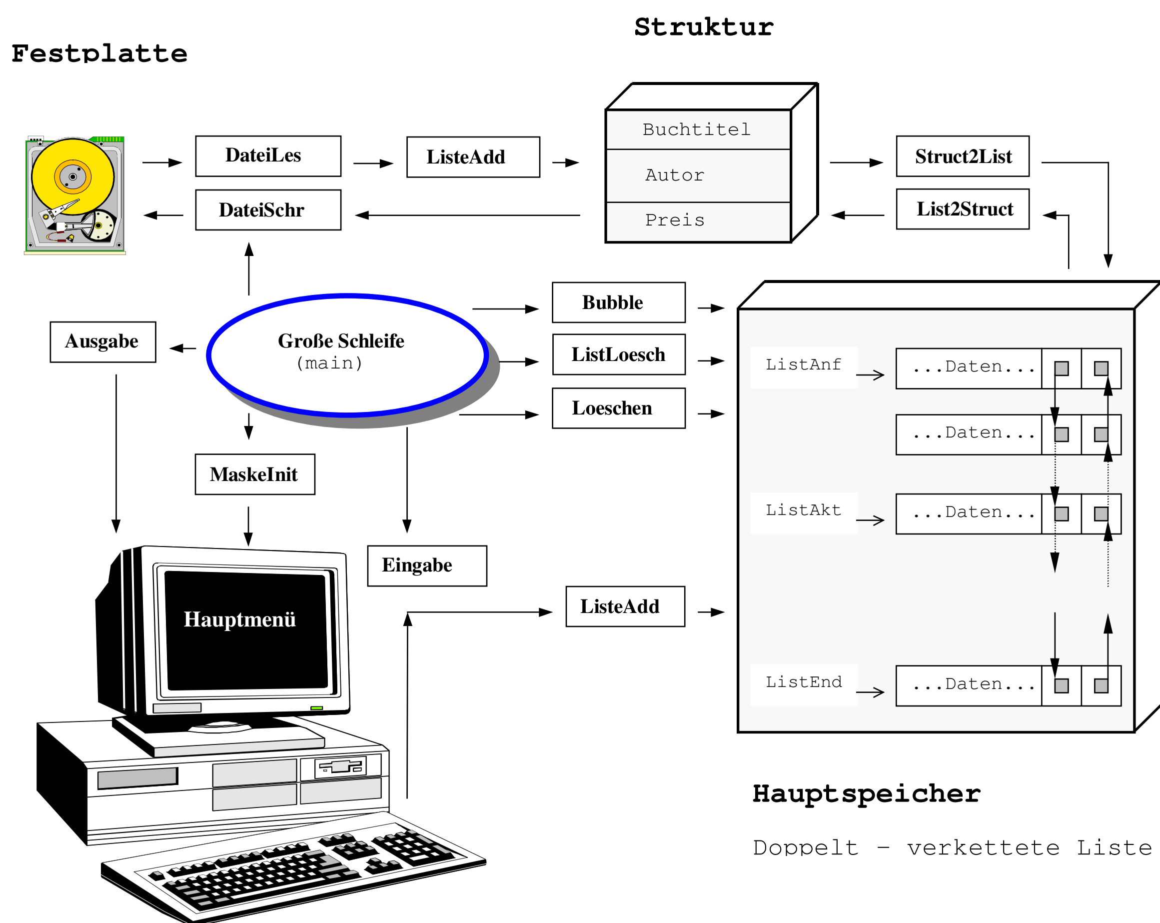
1 Inhalt

- Programmiersprachen
- OOP-Grundbegriffe allgemein
- OOP in Java
- Beispiel: AutoFahren
  - Hauptprogramm
  - Klasse Auto mit Attributen
  - Konstruktoren
  - Datenkapselung
  - Vererbung
  - Überschreiben von Methoden

1.1 Kritik an bisheriger Herangehensweise

- Problemlösungsbereich wird mit unterschiedlichen Konzepten beschrieben (semantische Lücke)
- wenig Unterstützung bei inkrementellen Erweiterungen (Variantenbildung)
- wenig Unterstützung für systematische Wiederverwendung
- Trennung der Datenstrukturen und Verarbeitungsstrukturen

1.2 Prozedurale Herangehensweise



2 Was ist objektorientierte Programmierung?

- Konzept zur Entwicklung von Programmen
- Verlagerung des Schwerpunkts eines Programms von den Algorithmen auf die Daten
- wichtige Prinzipien
  - Datenabstraktion
  - Modularisierung
  - Hierarchisierung (Klassenhierarchie)
  - Klassen und Objekte
  - Prinzip der Kapselung (Information hiding)
  - Funktions-Überladung
  - Vererbung und Ableitung (evtl. auch Mehrfachvererbung)
  - Polymorphismus

2.1 Grundidee der OOP

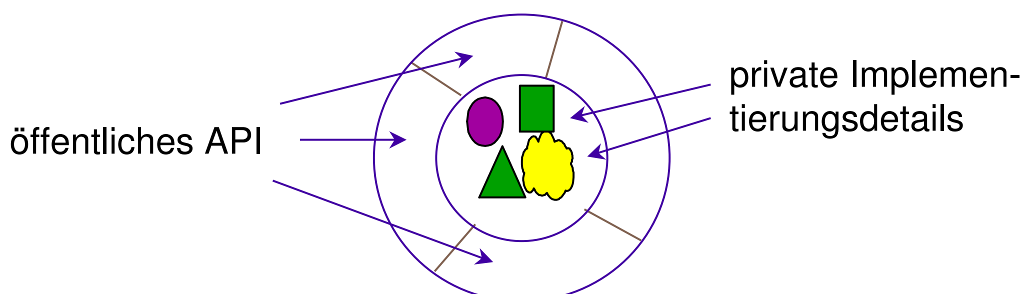
- Verknüpfung von Objekten mit eigener Aktionsfähigkeit
- Sichere Programme durch Festlegungen, wer Daten verändern darf
- Namen von Unterprogrammen können mehrfach verwendet werden.
- Wiederverwendbarkeit durch Klassen und Vererbung
- Intelligente Objekte reagieren auf Botschaften. Sie wählen, auf welche Botschaft sie reagieren.

2.2 Prinzipien der OOP

- Definition von Objekt-Orientierung nach B. Meyer
  - 1. Object-based modular structure
  - 2. Data abstraction
  - 3. Automatic memory management
  - 4. Classes
  - 5. Inheritance
  - 6. Polymorphism and dynamic binding
  - 7. Multiple and repeated inheritance

2.2.1 Objekte und Klassen

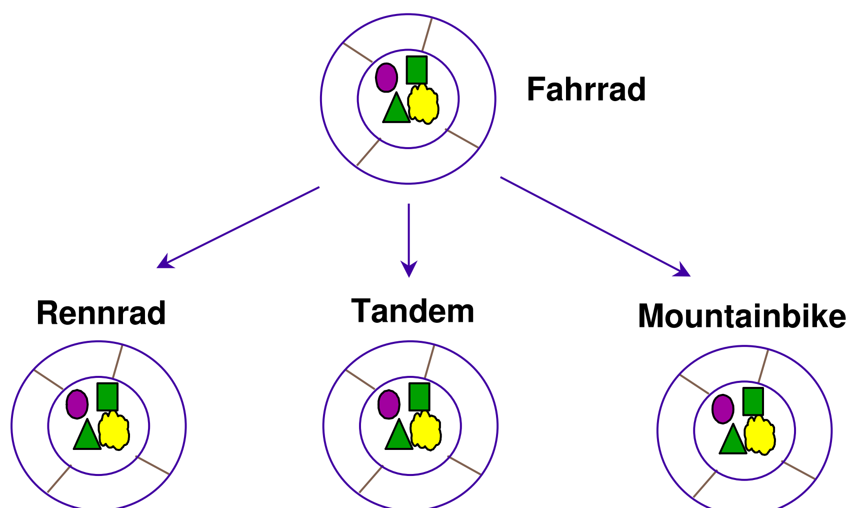
- Was ist ein Objekt (object)?
  - Ein Objekt ist ein Bündel aus Variablen und dazugehörigen Methoden
  - Software-Objekte werden häufig dazu verwendet, alltägliche Objekte der realen Welt in einem Modell abzubilden



- Objekt
  - Abbildung eines konkreten Gebildes der Realität
  - Instanz einer Klasse, d.h. ein konkretes Exemplar einer Klasse
  - Kapselung von Daten (Attributen) und Methoden
- Klasse
  - allgemeine Beschreibung eines oder mehrerer Objekte mit übereinstimmenden Attributen (Eigenschaften) und Methoden (Funktionalität)
  - Muster für Objekte (Bauplan für Objekte gleicher Art)
  - Konstrukt zur Definition eines Objekts
  - enthält Deklarationen der Attribute und Methoden

2.2.2 Vererbung und Ableitung

- Was ist eine Vererbung (inheritance)?
  - Eine Klasse erbt den Status und das Verhalten von ihrer Super-Class. Dies ist eine sehr mächtiger und natürlicher Weg, Software zu organisieren und zu strukturieren



- Nutzung („erben") von Eigenschaften bereits vorhandener Klassen (Basisklassen) durch Ableitung
  - Eigenschaften können verändert werden
  - Eigenschaften können hinzu gefügt werden
- Ziel: wiederverwendbarer Programmcode
- Mehrfachverarbeitung: eine Klasse kann von mehreren Klassen erben (Beispiele: Roller erbt von Spielzeug und Fahrzeug oder Wal erbt von Säuger und von Fisch) Beispiel: C++
- Einfachvererbung: eine Klasse kann nur von einer Klasse erben (z.B. Java)

2.2.3 Prinzip der Kapselung

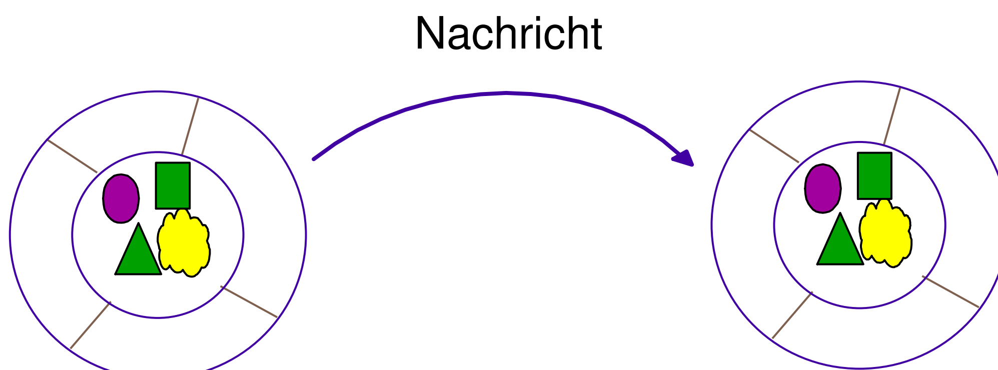
- Datendarstellung und Funktionsablauf werden durch die Schnittstelle versteckt
- Daten eines Objekts sollen nur durch die zugehörigen Methoden des Objekts manipuliert werden können, das heißt kein direkter Zugriff auf die Daten eines Objekts durch das Anwendungsprogramm
- wird erreicht durch die Möglichkeit, die Methoden eines Objekts „sichtbar", die Daten jedoch „unsichtbar" zu deklarieren

2.2.4 Funktions-Überladung

- Mehrere Funktionen können innerhalb einer Klasse denselben Namen haben
- Parameterliste ist entscheidend für die richtige Auswahl der Funktion (Anzahl und Typ der Parameter)
- sinnvoller Einsatz, wenn gleiche Aufgaben für verschiedene Datenformen durchgeführt werden sollen (Drucken von Text und Drucken von Bildern; in beiden Fällen heißt die Methode „drucken")
- In C++ können auch Operatoren (+, *, -, /) überladen werden, wenn z.B. rationale Zahlen oder komplexe Zahlen definiert werden
- in Java können keine Operatoren überladen werden!

2.2.5 Polymorphismus

- Was ist eine Nachricht (message)?
  - Objekte interagieren oder kommunizieren miteinander mittels Nachrichten
  - Aufruf einer Methode eines Objektes ist auch eine Nachricht an das Objekt (Aufforderung)



- Die gleiche Nachricht kann zu Objekten verschiedener Klassen geschickt werden
- eine Methode wird polymorph genannt, wenn sie für mehrere verschiedene Objekttypen verwendet werden kann
- Möglichkeiten, wo sich Methodendefinition der aufgerufenen Methode befinden kann:
  - Klasse des Objektes
  - Superklasse der Objektklasse (Methode wurde vererbt)
  - Methode existiert in Superklasse der Objektklasse, wurde aber in der Objektklasse überschrieben

3 OOP in Java 3.1 Klassen in Java

- Abstrakte Beschreibung von Objekten
- Es existieren in Java keine Variablen und Methoden außerhalb von Klassen!
- Die Superklasse (Basisklasse) aller anderen Klassen ist direkt oder indirekt **„Object"**
  - wenn keine Ableitung angegeben ist, wird von **„Object"** abgeleitet
  - Methode **„clone()"** erzeugt exakte Kopie des Objekts
  - Methode **„toString()"** liefert String-Repräsentation
  - Methode **„equals(Object o)"** prüft auf dieselbe Referenz, wenn nicht anders überschrieben
  - Den Instanzen von „Object" können Instanzen anderer Klassen zugewiesen werden.

Klassen in Java

- Ein Java-Programm (Applikation oder Applet) besteht aus mindestens einer *„public"*-Klasse die in eine Datei mit exakt demselben Namen wie die Klasse + „.java" geschrieben wird!
  - Neben der „public"-Klasse können weitere Klassen in dieselbe Datei oder in andere Dateien geschrieben werden.
  - Maximal eine Klasse pro Datei darf „public" sein
- Eine Klasse kann weitere „innere" Klassen enthalten
  - Beispiel des Namen einer entsprechenden class-Datei: *MeinProgramm$InnereKlasse.class*
  - Objekte der inneren Klasse können nur in der äußeren Klasse erzeugt werden (außer bei statischen inneren Klassen)
  - Aus inneren Klassen kann auf Daten und Methoden der äußeren Klasse zugegriffen werden (auch auf private)

Schlüsselwort „this"

- In jedem Objekt gibt es automatisch die Referenzvariable *„this"* die an alle Methoden übergeben wird
- Anwendungen:
  - Zugriff auf die Instanzvariablen bei gleichnamigen Variablen
  - eigenes Objekt als Rückgabewert einer Methode
  - Übergabe als Parameter
  - Verkettung von Konstruktoren

```
public Test() // Konstruktoren haben keinen Typ!
{
   this(i);   // Aufruf des Konstruktors dieser
              // Klasse mit einem Parameter
}
```

Anonyme Klassen in Java

- Eine besondere Form von inneren Klassen sind anonyme Klassen
  - die Klassendefinition wird in einem Schritt mit der Objekterzeugung verbunden

```
addWindowListener(new WindowAdapter()
    {
        public void windowClosing(WindowEvent we)
        {
            setVisible(false);
            dispose();
            System.exit(0);
        }
    });
```


Kapselung

Modifizierer	Klasse	abgeleitete Klasse	Package	öffentlich
private	x	O	O	O
„package“ (Standard)	x	O	x	O
protected	x	x	x	O
public	x	x	x	x

3.2 Interfaces

- In Java gibt es keine Mehrfachvererbung, dafür gibt es „Interfaces" (Schnittstellen)!
  - Klassen können mehrere Interfaces implementieren
  - Interfaces dürfen nur Konstanten und Methoden-Deklarationen enthalten
  - Klassen, die Interfaces implementieren, müssen alle im Interface enthaltenen Methoden implementieren
  - alle Methoden eines Interface sind automatisch *„abstract"* und *„public"* (sie können nicht „private" oder „protected" sein)
  - Interfaces enthalten keine Konstruktoren und Destruktoren
- Interfaces ohne Methoden und Attribute sind Flag-Interfaces
  - z.B. Interface *„Serializable"*

Adapterklassen

- Adapterklassen erweitern Interfaces
  - Sie können außer den vom Interface implementierten Methoden weitere Methoden besitzen
  - Sie werden häufig bei Ereignisbehandlung eingesetzt
  - In Klassen, die von Adapterklassen abgeleitet sind, müssen nur die benötigten Methoden überschrieben werden
  - Beispiel für eine Adapterklasse:

```
public class xxxAdapter implements xxxInterface
{
   public void methode1() {}
   public void methode2() {}
   public void methode3() {}
   . . .
}
```

3.3 Objekte

- elementare Datentypen sind keine Objekte
  - zu den elementaren Datentypen gibt es korrespondierende Wrapper-Klassen, um Objekte erzeugen zu können
- Objekte werden mit „new" erzeugt (nicht immer)
  - Deklaration (Anlegen der Referenz) und Erzeugung des Objekts (Speicherbelegung) können getrennt erfolgen
  - ein Objekt existiert, solange eine Referenz auf das Objekt existiert; Freigabe erfolgt durch den „Garbage Collector"
  - String-Objekte können auch ohne „new" erzeugt werden (Beispiel: **String Text1= "Hallo";**)
  - Es können anonyme Objekte erzeugt werden, z.B.

```
g.drawPolygon(new int[] {50,100,150},
    new int[] {150,50,150}, 3);
```


3.4 Methoden

- Methoden sind immer in Klassen eingebunden, d.h. es kann keine Methoden außerhalb von Klassen geben
- in Applikationen muss mindestens eine Methode „main" vorhanden sein
- „main" ist „static", deshalb muss von der Klasse „AutoFahren" kein Objekt gebildet werden

```
public class AutoFahren
{
   public static void main(String[] Argumente)
   {
      System.out.println("Hauptprogramm");
   }
}
```

„static“ und „final“

- Statische Methoden und Variablen (Klassenvariablen) sind nur einmal vorhanden
- statische Methoden können nur mit statischen Variablen und Methoden arbeiten
- es muss kein Objekt gebildet werden, um auf statische Methoden und Variablen zuzugreifen
- finale Variablen dürfen nicht verändert werden, sind also Konstanten
- finale Methoden dürfen nicht überschrieben werden
- von finalen Klassen können keine weiteren Klassen ab-geleitet werden (Beispiele: java.lang.String und java.lang.Math)

Konstruktoren

- Konstruktoren in Java haben denselben Namen wie die Klasse und sind typenlos (ohne Rückkehrwert)
- Konstruktoren werden nicht wie andere Methoden aufgerufen, sondern werden automatisch beim Bilden einer Instanz aufgerufen, sie sind deshalb *„public“*
- Java stellt immer einen parameterlosen Standard-Konstruktor bereit

```
public class Auto
{
   public Auto()
   {
      System.out.println("Standard-Konstruktor");
   }
}
```

Destruktoren

- In Java gibt es keine Destruktoren
- In jeder Klasse kann eine Methode „finalize" definiert werden, die aufgerufen wird, wenn der Garbage-Collector das Objekt beseitigt
  - es kann nicht garantiert werden, ob und wann „finalize" aufgerufen wird
  - „finalize" wird nicht direkt aufgerufen
  - eine Aufruf-Reihenfolge bei mehreren Objekten ist nicht garantiert

```
public void finalize()
{
    . . .
}
```


Überladen von Methoden

- Methoden können nicht nur anhand des Typs unterschieden werden
- Methoden gleichen Namens in derselben Klasse unterscheiden sich durch Typ und Anzahl der Parameter
- Das „Überladen" von Methoden wird meist für Methoden angewendet mit gleicher Funktionalität aber für verschiedene Daten

4 OOP in Java am Beispiel

4.1 Hauptprogramm AutoFahren

```
import java.io.*;

public class AutoFahren
{
   public static void main(String Argumente[])
   {
      System.out.println("Programm AutoFahren");
             // hier wird spaeter mit den
             // Instanzen der Klasse "Auto" und
             // "A_Klasse" gearbeitet
    }
}
```


Klasse „Auto“ und Instanzbildung

```
public class Auto        // in Datei "Auto.java"
{
   int Ps= 0;            // Attribute der Klasse
   String Name= "";
}
```

```
   . . .
   System.out.println("Programm AutoFahren");

   Auto Auto1= new Auto();  // Instanz "Auto1"
   . . .
```

Klasse, von der das Objekt gebildet wird

Name des Objektes

Name des Konstruktors (ohne Parameter)

Standard-Konstruktor

```
 . . .
 Auto Auto1= new Auto();      // Instanzbildung
 System.out.println("Ps= " + Auto1.Ps);
 . . .
```

```
public class Auto
{
   int Ps= 0;              // Attribute der Klasse
   String Name= "";

   public Auto()           // Standard-Konstruktor
   {
      Ps= 75; Name= "Auto";
   }
}
```


Anwendung weiterer Konstruktoren

```
. . .
Auto Auto1= new Auto();   // Instanzbildung
System.out.println("Ps= " + Auto1.Ps +
                   "Name= " + Auto1.Name);

// Was getan werden muss, damit folgende Zeilen
// funktionieren, folgt auf der naechsten Folie

Auto Auto2= new Auto(100);   // mit 100 Ps
System.out.println("Ps= " + Auto2.Ps +
                   "Name= " + Auto2.Name);

Auto Auto3= new Auto(90, "Auto3");
                    // "Auto3" mit 90 Ps
System.out.println("Ps= " + Auto3.Ps +
                   "Name= " + Auto3.Name);
. . .
```

Weitere Konstruktoren

```
public class Auto
{
   . . .                  // Attribute der Klasse
   . . .                  // Standard-Konstruktor

   public Auto(int ps)       // 2. Konstruktor
   {
      Ps= ps; Name= "Auto";
   }

   public Auto(int ps, String name) // 3. Konstr.
   {
      Ps= ps; Name= name;
   }
}
```


Weitere Methoden

```
public class Auto
{
   . . .                  // Attribute der Klasse
   . . .                  // Konstruktoren

   public void fahren()        // eigene Methode
   {
      System.out.println("Auto faehrt");
   }
}
```

```
   . . .
   Auto1.fahren(); // Nachricht an Objekt "Auto1"
                   // zur Ausfuehrung der Methode
   . . .
```

Datenkapselung

```
public class Auto
{
   private int Ps= 0;     // Attribute der Klasse
   String Name= "";
   . . .
}
```

- Mit dem Schlüsselwort **„*private*"** ist auf dieses Attribut nicht mehr außerhalb dieser Klasse zugreifbar
- Die Klasse „Auto" lässt sich separat ohne Syntax-Fehler compilieren
- Das Compilieren der Klasse „AutoFahren" bringt Fehler-Meldung: . . . „**`Ps has private access in Auto`**"

„get"- „set"-Methoden

```
public class Auto
{
   private int Ps= 0;     // Attribute der Klasse
   String Name= "";
   . . .

   public int getPs()    // nur mit dieser Methode
   {                     // kann auf das Attribut
      return Ps;         // außerhalb der Klasse
   }                     // zugegriffen werden

   public void setPs(int ps)    // nur mit dieser
   {                            // Methode kann das
      Ps= ps;                   // Attribut außer-
                                // halb der Klasse
   }                            // geaendert werden
}
```

Anwendung der „get"- „set"-Methoden

```
. . .

Auto Auto2= new Auto(100);   // mit 100 Ps

System.out.println("Ps= " + Auto2.getPs() +
                   "Name= " + Auto2.Name);

Auto2.setPs(150);            // 150 Ps

System.out.println("Ps= " + Auto2.getPs() +
                   "Name= " + Auto2.Name);
. . .
```


Ableitung einer Klasse (Vererbung)

- Der Quellcode der neuen Klasse kann auch in die Datei ***„Auto.java"*** geschrieben werden (ans Ende der Datei)
- Es kann auch eine neue Datei ***„A_Klasse.java"*** mit einer entsprechenden public-Klasse angelegt werden
- Es werden alle Attribute und Methoden der Klasse ***„Auto"*** mit Ausnahme der Konstruktoren vererbt
- Auch die Attribute mit „private" werden vererbt (das ist keine Zugriffsverletzung)

```
class A_Klasse extends Auto
{
}
```

Anwendung der neuen Klasse

```
. . .

A_Klasse Auto4= new A_Klasse();

System.out.println("Ps= " + Auto4.getPs() +
                   "Name= " + Auto4.Name);
Auto4.fahren();
. . .
```

- Das Beispiel zeigt, dass die Konstruktoren (außer dem Standardkonstruktor) nicht vererbt werden
- Alles andere funktioniert

Weitere Methode „fahren“

```
// Methode der Klasse „Auto“

public void fahren(boolean elch)
{
   if (elch)
   {
      System.out.println("Achtung Elch!");
      System.out.println("Auto stabilisiert");
   }

   System.out.println("Auto faehrt");
}
```

- Der Aufruf der Methode „fahren“ mit Parameter sieht für beide Klassen völlig gleich aus

Methode überschreiben

```
// Methode der Klasse „A_Klasse"

public void fahren(boolean elch)
{
   if (elch)
   {
      System.out.println("Achtung Elch!");
      System.out.println("Auto kippt");
   }
   else
      System.out.println("Auto faehrt");
}
```

- Bemerkung: Der Fehler beim entsprechenden Autotyp wurde längst beseitigt.

Polymorphie (Vielgestaltigkeit)

```
. . .
// Objekt der Klasse „Auto"

Auto3.fahren();            // ohne Parameter
Auto3.fahren(true);        // mit Parameter
Auto3.fahren(false);

// Objekt der Klasse „A_Klasse"

Auto4.fahren();            // ohne Parameter
Auto4.fahren(true);        // mit Parameter
Auto4.fahren(false);

. . .
```

Konstruktoren der Klasse A_Klasse

```
public class A_Klasse
{   . . .                 // Attribute der Klasse

   public Auto()          // Standard-Konstruktor
   {
      super();
   }
   public Auto(int ps)  // 2. Konstruktor
   {
      super(ps);
   }
   public Auto(int ps, String name) // 3. Konstr.
   {
      super(ps, name);
   }
}
```


Schlüsselwort „super"

- Das Schlüsselwort ***„super"*** steht für Objekt oder Konstruktor der Basisklasse (Superklasse)
- Im Konstruktor muss „super" als erste Anweisung stehen, wenn es verwendet wird!
- Nach „super" können weitere Anweisungen zur Spezifizierung dieser Konstruktoren stehen

```
. . .

A_Klasse Auto5= new A_Klasse();     // ohne Parameter
A_Klasse Auto6= new A_Klasse(200);
A_Klasse Auto7= new A_Klasse(120, "Auto7");

. . .
```

Literaturangaben

[1] Guido Krüger: *Handbuch der Java-Programmierung*,
Addison-Wesley Verlag,
ISBN 3-8273-2201-4

[2] Elke Niedermair / Michael Niedermair:
Internet-Programmierung mit Java,
Data Becker GmbH & Co. KG,
ISBN 3-8158-2086-3

[3] Alexander Niemann: *Das Einsteigerseminar*,
Objektorientierte Programmierung in Java
bhv Verlag Bürohandels- und Verlagsgesellschaft mbH,
ISBN 3-8287-1015-8

[4] Java 2 SDK v 1.2.2*, Grundlagen Programmierung*
HERDT-Verlag für Bildungsmedien GmbH, Nackenheim