

LE 4  
1

## Grundlagen der Informatik

### 2 Grundlagen der Programmierung – Objekte und Klassen (Teil 1)



Prof. Dr. Helmut Balzert  
Lehrstuhl für Software-Technik  
Ruhr-Universität Bochum

© Helmut Balzert 2000

GDI -Programmierung: Objekte & Klassen 1

LE 4  
2

### Lernziele

- Für Beispiele geeignete Diagramme in UML-Notation auswählen und zeichnen können
- Java-Programme auf die Einhaltung der behandelten Syntax prüfen können
- Die dynamischen Abläufe bei der Erzeugung und Referenzierung von Objekten zeichnen und erläutern können
- Die dynamischen Abläufe beim Versenden von Botschaften an Objekte zeichnen und erläutern können.

## **Lernziele**

- **Das Programm »Kundenverwaltung« auf analoge Aufgaben übertragen und modifizieren können, so daß lauffähige Java-Programme entstehen**
- **Aus einem Pflichtenheft ein Klassendiagramm in UML-Notation erstellen und dieses Diagramm anschl. in ein Java-Programm transformieren und um Operationsanweisungen ergänzen können.**

## **Inhalt**

### **2.5 Zuerst die Theorie: Objekte und Klassen**

2.5.1 Intuitive Einführung

2.5.2 Objekte

2.5.3 Klassen

### **2.6 Dann die Praxis:**

#### **Objekte und Klassen in Java**

2.6.1 Deklaration von Klassen

2.6.2 GUI-Klassen

2.6.3 Erzeugen und Referenzieren von Objekten

2.6.4 Senden von Botschaften und Ausführen von Operationen

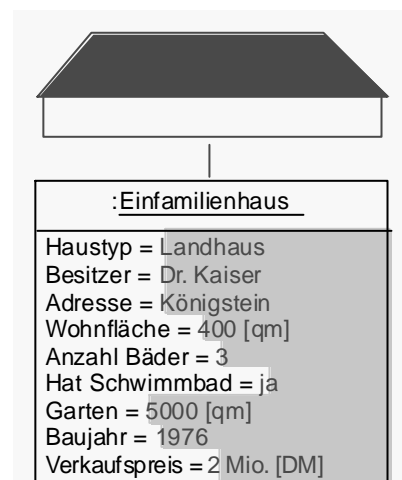
2.6.5 Löschen von Objekten.

## 2.5 Zuerst die Theorie: Objekte und Klassen

- ♦ **Multimedia-Einführung**
- ♦ **Historie**
  - ◆ Grundkonzepte der OO-Software-Entwicklung stammen von Smalltalk-80
  - ◆ Smalltalk-80: 1. objektorientierte Sprache
  - ◆ Entwickelt: 1970 bis 1980 am Palo Alto Research Center (PARC) der Firma Xerox
  - ◆ Klassenkonzept wurde von SIMULA 67 übernommen und weiterentwickelt
  - ◆ Konzepte wurden in Smalltalk-80 – wie in Programmiersprachen üblich – textuell repräsentiert
  - ◆ Heute auch grafische Notationen üblich.

### 2.5.1 Intuitive Einführung

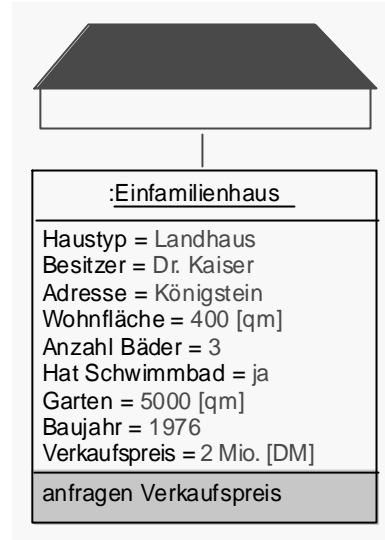
- ♦ **Beispiel: Immobilienfirma Nobel & Teuer**
  - ◆ Firma Nobel & Teuer vermittelt exklusive Einfamilienhäuser
- ♦ **Objekt**
  - ◆ Jedes Einfamilienhaus ist ein Objekt
  - ◆ Attributwerte eines Objekts:



LE 4  
7

## 2.5.1 Intuitive Einführung

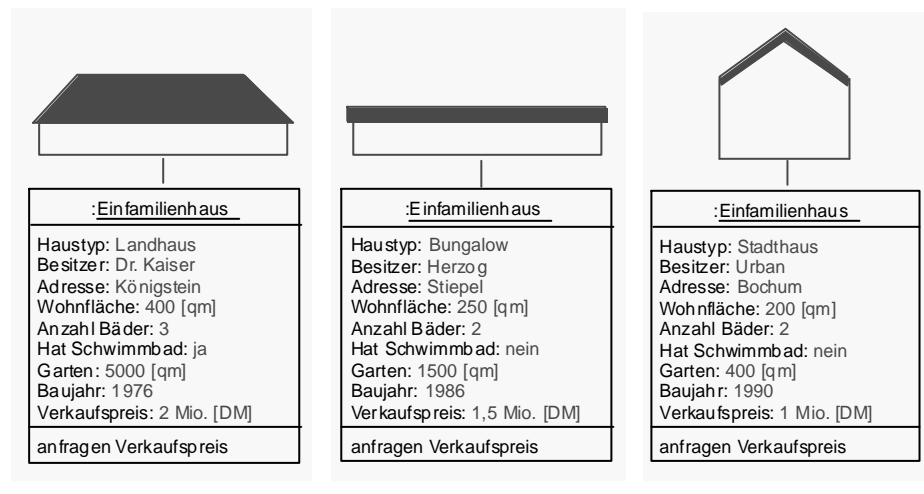
- **Operation (Methode)**
  - ◆ Kommt ein potentieller Käufer zu Nobel & Teuer, muß der Verkaufspreis verfügbar sein
  - ◆ Es wird eine Funktion (Operation) benötigt, die auf das Objekt »Landhaus« angewendet wird:



LE 4  
8

## 2.5.1 Intuitive Einführung

- **Andere Einfamilienhäuser der Firma Nobel & Teuer**



## 2.5.1 Intuitive Einführung

### • Klasse

- ◆ Man spricht von den Attributen »Haustyp«, »Besitzer« usw.
- ◆ Alle Objekte besitzen die gleiche Operation »anfragen Verkaufspreis«
- ◆ Diese Objekte werden daher zur Klasse »Einfamilienhaus« zusammengefaßt

| Einfamilienhaus  |
|--|
| Haustyp<br>Besitzer<br>Adresse<br>Wohnfläche<br>Anzahl Bäder<br>Hat Schwimmbad<br>Garten<br>Baujahr<br>Verkaufspreis |
| anfragen Verkaufspreis   |

## 2.5.1 Intuitive Einführung

### • Klasse

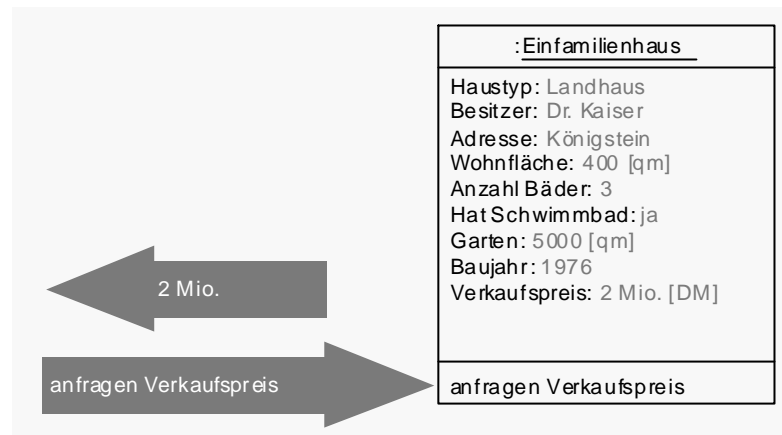
- ◆ ...definiert die Attribute und Operationen ihrer Objekte
- ◆ Ausführung der Operation »anfragen Verkaufspreis« für das Objekt »Landhaus« durch Versenden einer Botschaft

### • Botschaft

- ◆ ...aktiviert eine Operation gleichen Namens
- ◆ Die gewünschten Ausgabedaten werden an den Sender der Botschaft zurückgegeben.

## 2.5.1 Intuitive Einführung

### ♦ Botschaft



## 2.5.2 Objekte

### ♦ Objekt

#### ◆ Allgemein:

- **Gegenstand des Interesses, insbesondere einer Beobachtung, Untersuchung oder Messung**

#### ◆ OO:

- **Objekt ist ein individuelles Exemplar von Dingen:**
  - Roboter, Auto
  - Personen (z.B. Kunde, Mitarbeiter)
  - Begriffe der realen Welt (z.B. Bestellung)
  - Begriffe der Vorstellungswelt (z.B. juristische und natürliche Personen).

## 2.5.2 Objekte

## ♦ Charakteristika

- ◆ Jedes Objekt besitzt eine Objekt-Identität, die es von allen anderen Objekten unterscheidet
- ◆ Ein Objekt kann andere Objekte kennen
  - Zwischen Objekten, die sich kennen, bestehen Verbindungen (*links*)
- ◆ Zustand (*state*) eines Objekts
  - Bestimmt durch Attributwerte bzw. Daten und Verbindungen zu anderen Objekten
- ◆ Verhalten (*behavior*) eines Objekts
  - Beschreibung durch Menge von Operationen

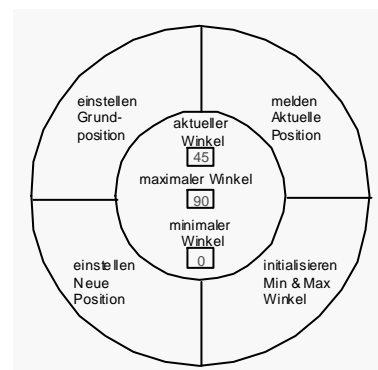
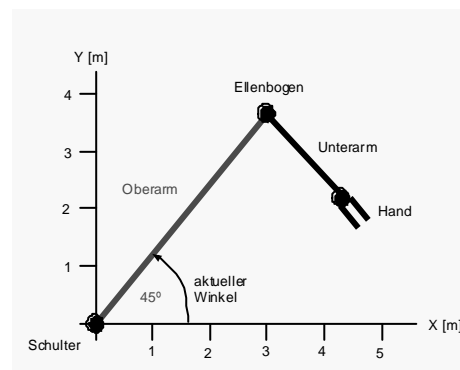
## ♦ Synonyme

- ◆ Exemplar, Instanz, *instance*, *class instance*.

## 2.5.2 Objekte

## ♦ Beispiel: Objekt »Oberarm des Roboters«

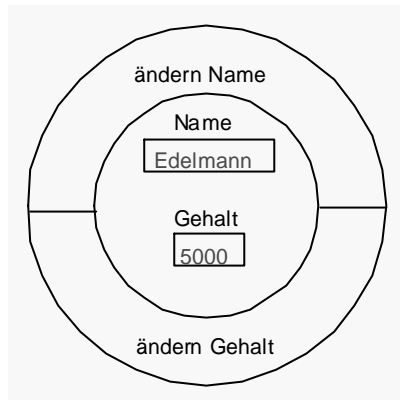
- ◆ Kennzeichnung des Objekts »Oberarm des Roboters« durch 3 Attributwerte
- ◆ Manipulation der Winkel durch 4 Operationen



## 2.5.2 Objekte

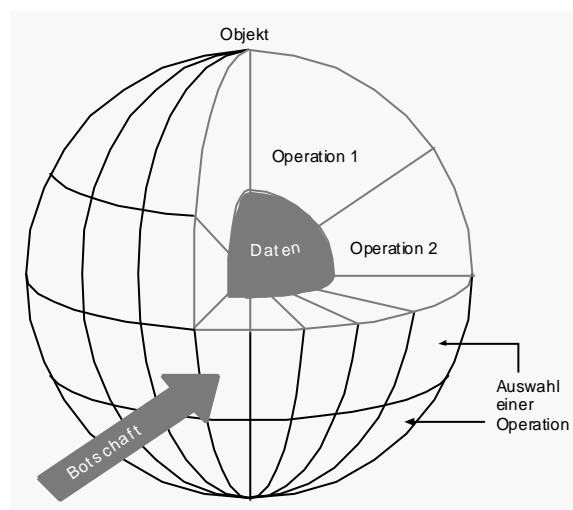
### • Beispiel: Objekt »Mitarbeiter Edelmann«

- ◆ Die Zustände dieses Objekts werden durch 2 Attributwerte bestimmt
- ◆ Das Verhalten wird durch 2 Operationen bestimmt



## 2.5.2 Objekte

### • Geheimnisprinzip



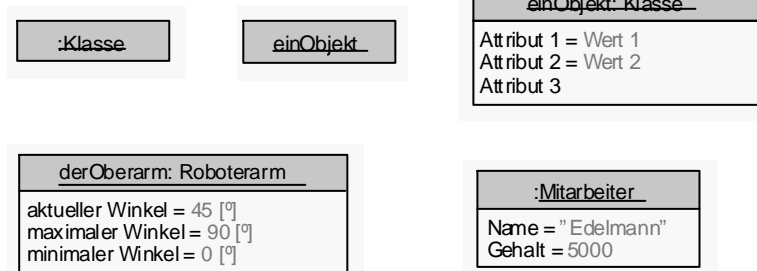
## 2.5.2 Objekte

### ♦ UML-Notation

- ◆ **Objekt: zweigeteiltes Rechteck**
- ◆ **Oberer Teil: enthält den unterstrichenen Klassennamen, mit vorangestelltem Doppelpunkt, zu dem das Objekt gehört**
- ◆ **Besitzt das Objekt einen eigenen Namen, dann steht dieser vor dem Doppelpunkt**
  - **Z.B. einLandhaus: Einfamilienhaus**
- ◆ **Ist die Klasse aus dem Kontext ersichtlich, dann genügt auch der unterstrichene Objektname**
- ◆ **Klassenname**
  - **Beginnt mit Großbuchstaben**
- ◆ **Objektname**
  - **Beginnt mit Kleinbuchstaben.**

## 2.5.2 Objekte

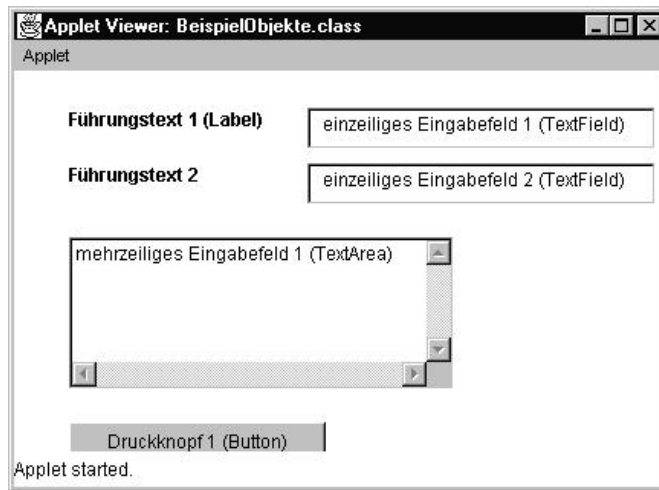
- ◆ **Unterer Teil (optional): relevante Attribute des Objekts mit den Attributwerten, getrennt durch ein Gleichheitszeichen**
- ◆ **Nicht interessierende Attributwerte können weggelassen werden**
- ◆ **Operationen werden in der UML-Notation für Objekte nicht aufgeführt**



## 2.5.2 Objekte

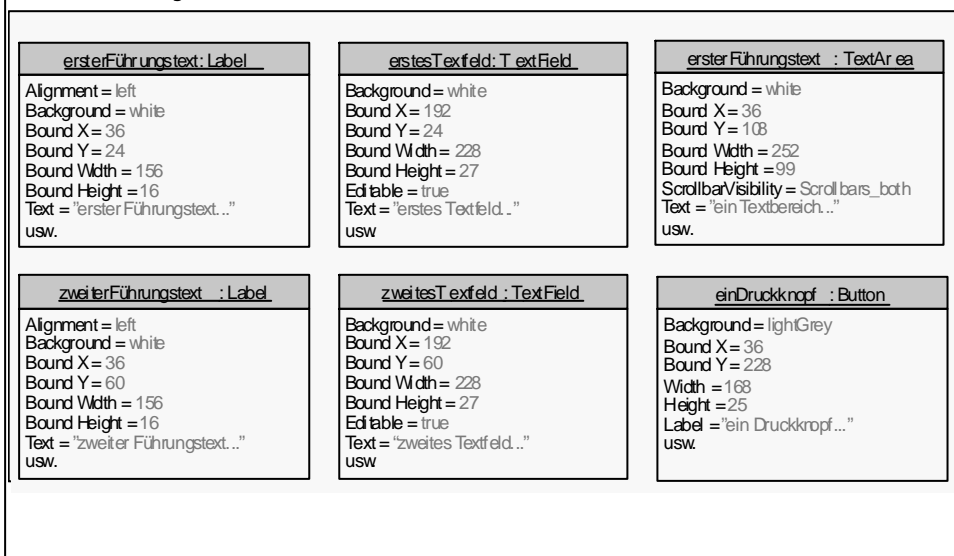
### • Beispiel

#### ◆ Interaktionselemente sind ebenfalls Objekte



## 2.5.2 Objekte

### • Objekte in UML-Notation



## 2.5.2 Objekte

### ♦ Fallstudie ProfiSoft

- ◆ **Pflichtenheft für ein Kundenverwaltungsprogramm**
  - /1/ Firmenname und Firmenadresse jedes Kunden sollen gespeichert werden**
  - /2/ Alle Daten müssen einzeln gelesen werden können**
  - /3/ Alle Daten müssen einzeln geschrieben werden können**
  - /4/ Firmenname und Firmenadresse müssen zusammen geändert werden können (ändern Adresse)**
  - /5/ Die Auftragssumme ist mit dem Wert »0« vorzubesetzen.**

## 2.5.2 Objekte

- ◆ **Eigenschaften der Kunden »KFZ-Zubehör GmbH« und »Tankbau KG« :**

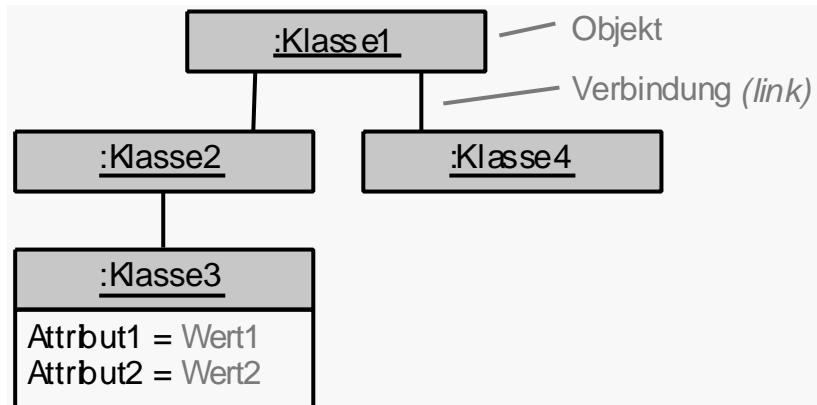
| <u>:Kunde</u>   |
|---|
| Firmenname = KFZ-Zubehör GmbH<br>Adresse = 44137 Dortmund, Poststr. 12<br>Auftragssumme = 0 |

| <u>:Kunde</u>  |
|--|
| Firmenname = Tankbau KG<br>Adresse = 44867 Bochum, Südstr. 23<br>Auftragssumme = 0 |

## 2.5.2 Objekte

### • Objektdiagramme

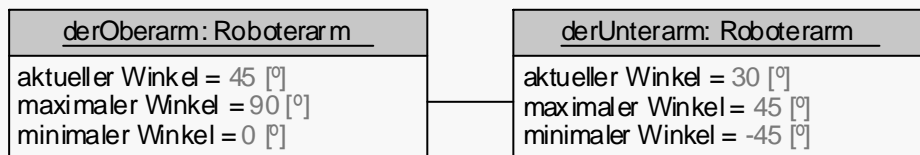
- ◆ Zur Darstellung von Objekten und deren Verbindungen zu anderen Objekten



## 2.5.2 Objekte

### • Beispiel: Roboter

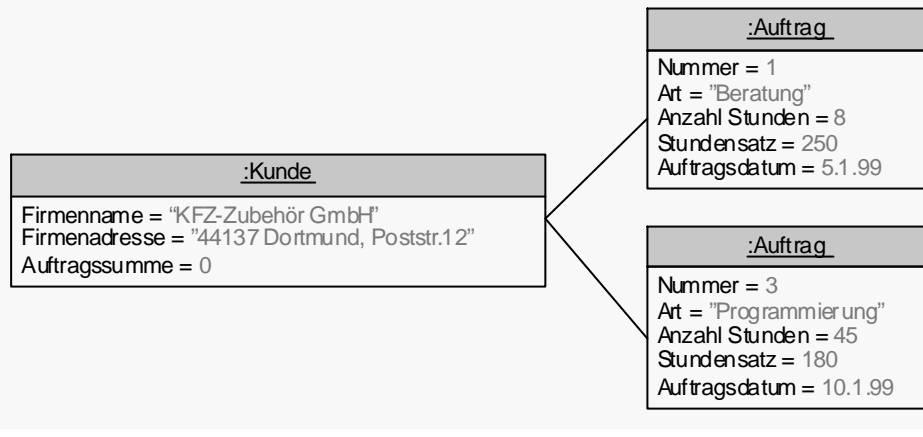
- ◆ Objekte »Oberarm« und »Unterarm« stehen in einer Verbindung



## 2.5.2 Objekte

### ♦ Fallstudie

- ◆ Der Kunde »KFZ-Zubehör GmbH« hat 2 Aufträge erteilt



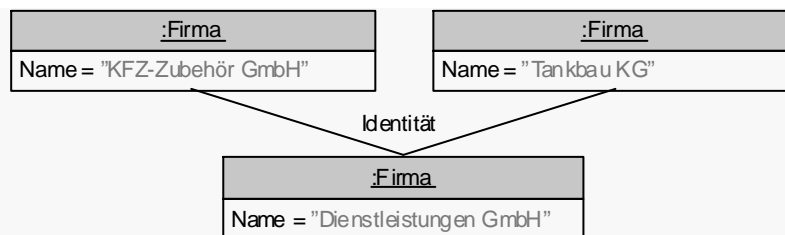
## 2.5.2 Objekte

### ♦ Objekt-Identität

- ◆ Jedes Objekt besitzt eine Objekt-Identität, die sie von allen anderen Objekten unterscheidet
- ◆ Keine 2 Objekte besitzen dieselbe Identität, auch wenn sie identische Attributwerte besitzen

### ♦ Identität von Objekten

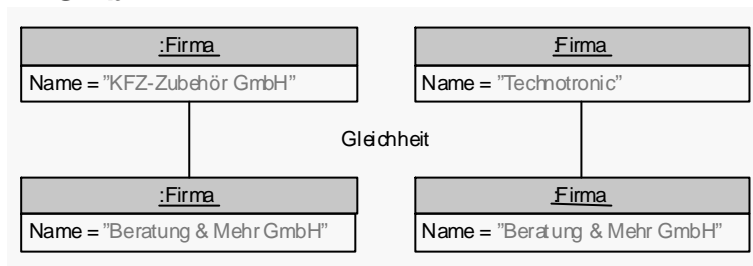
- ◆ Die Firmen »KFZ-Zubehör GmbH« und »Tankbau KG« haben eine gemeinsame Tochterfirma



## 2.5.2 Objekte

### • Gleichheit von Objekten

- ◆ 2 Objekte sind gleich
  - Sie besitzen dieselben Attributwerte
  - Haben aber unterschiedliche Identitäten
- ◆ Die Firmen »KFZ-Zubehör GmbH« und »Technotronic KG« besitzen beide eine Tochterfirma mit dem Namen »Beratung & Mehr GmbH«



## 2.5.3 Klassen

### • Klasse

- ◆ Allgemein:
  - Eine Gruppe von Dingen, Lebewesen oder Begriffen mit gemeinsamen Merkmalen
- ◆ OO:
  - Spezifizierung der Gemeinsamkeiten einer Menge von Objekten
    - mit denselben Eigenschaften (Attributen)
    - demselben Verhalten (Operationen)
    - denselben Beziehungen (Verbindungen)
  - Jede Klasse besitzt einen Mechanismus, um Objekte zu erzeugen (*object factory*)
    - Jedes erzeugte Objekt gehört zu genau einer Klasse.

## 2.5.3 Klassen

### • Klassen für ausgewählte Interaktionselemente

| Label   |
|---|
| Alignment : (Center, Left, Right)<br>Bound X : int<br>Bound Y : int<br>Bound Width : int<br>Bound Height : int<br>Text : String<br>usw. |
| Label()<br>Label(String)<br>getAlignment()<br>getText()<br>setAlignment<br>setText(String)<br>usw.                                      |

| TextArea   |
|--|
| Background : Color<br>Bound X : int<br>Bound Y : int<br>Bound Width : int<br>Bound Height : int<br>ScrollbarVisibility : (Scrollbars_both,<br>ScrollbarsHorizontal only,<br>ScrollbarsNone,<br>ScrollbarsVertical only)<br>Text : String<br>usw. |
| TextArea()<br>TextArea(String)<br>append(String)<br>getText()<br>setText()<br>getScrollbarVisibility()<br>usw.   |

## 2.5.3 Klassen

### • Klassen für ausgewählte Interaktionselemente

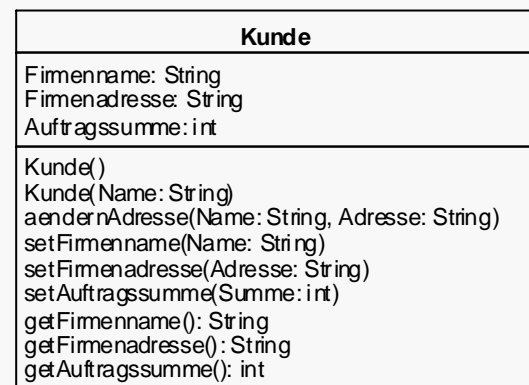
| TextField  |
|--|
| Background : Color<br>Bound X : int<br>Bound Y : int<br>Bound Width : int<br>Bound Height : int<br>Editable : (true, false)<br>Text : String<br>usw. |
| TextField()<br>TextField(String)<br>addActionListener(ActionListener)<br>getText()<br>isEditable()<br>setEditable(boolean)<br>setText()<br>usw.      |

| Button  |
|---|
| Background : Color<br>Bound X : int<br>Bound Y : int<br>Bound Width : int<br>Bound Height : int<br>Label : String<br>usw. |
| Button()<br>Button(String)<br>addActionListener(ActionListener)<br>getLabel()<br>setLabel()<br>usw.                       |

## 2.5.3 Klassen

### ♦ Fallstudie

- ◆ Die Objekte »KFZ-Zubehör GmbH« und »Tankbau KG« haben die gleichen Attribute und Operationen: Klasse Kunde



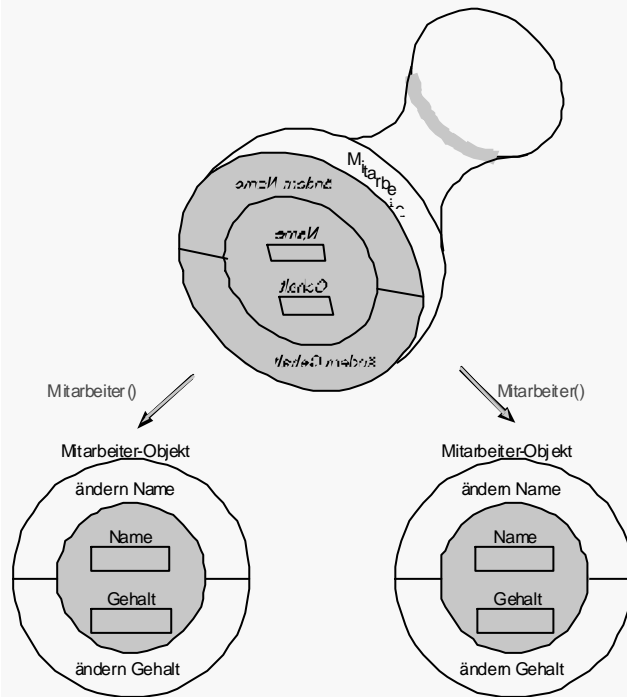
## 2.5.3 Klassen

### ♦ Attribute

- ◆ Jedem Attribut muß ein Typ (*type*) zugeordnet werden
- ◆ Beispiele für Typen
  - Zeichenketten (`String`)
  - Ganze Zahlen (`int`)
  - Zeichen (`char`)
  - Gleitkommazahlen (`float`)
- ◆ UML-Notation
  - `Attributname : Typ`
  - `setAttributname`
  - `getAttributname.`

### 2.5.3 Klassen

- ◆ Erzeugung eines neuen Objekts:  
Konstruktoroperation (kurz: Konstruktor)
  - Kunde(), Kunde(Name:String)
- ◆ Übergabe von Attributwerten über eine Operation an ein Objekt
  - aendernAdresse(Name:String, Adresse:String).

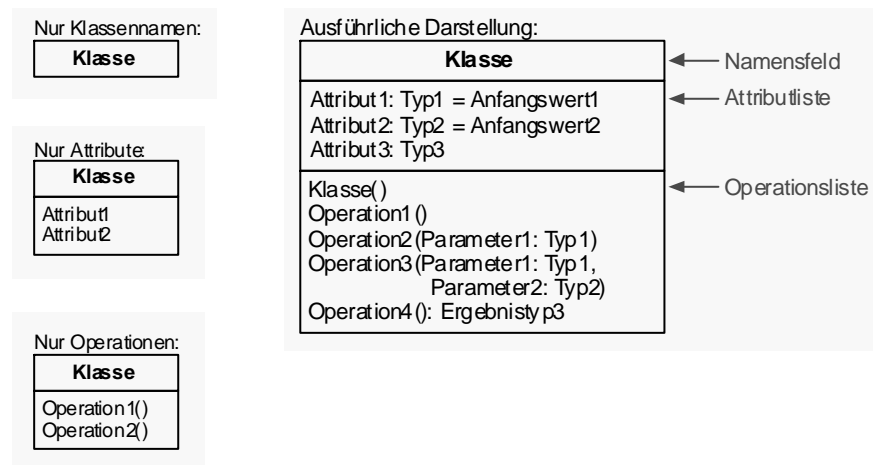


## 2.5.3 Klassen

- ♦ **Klassenname**
  - ◆ Adjektiv (optional) + Substantiv im Singular
  - ◆ Beispiele:
    - Mitarbeiter / Kunde / Öffentliche Veranstaltung
- ♦ **Notation**
  - ◆ In der grafischen Darstellung Angabe von ...
    - Klassenname
    - Attributnamen und ihrer Typen
    - Operationsnamen
  - ◆ Weitere Informationen werden textuell festgelegt.
- ♦ **Klassen-Diagramm**
  - ◆ Darstellung der Klassen und ihrer Beziehungen.

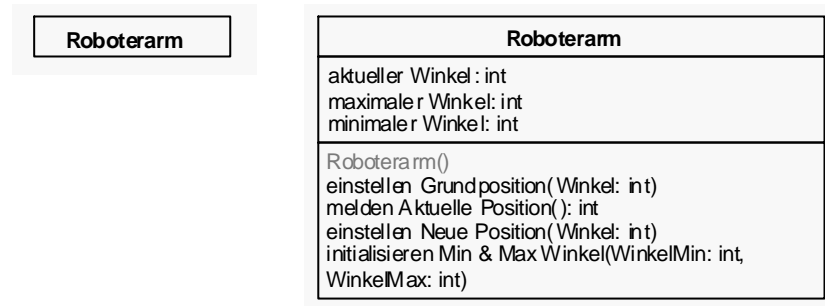
## 2.5.3 Klassen

### ♦ UML-Notation Klasse



## 2.5.3 Klassen

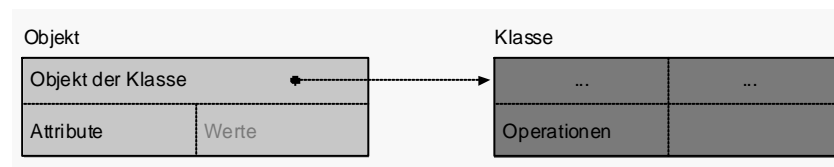
### • Klasse Roboterarm



## 2.5.3 Klassen

### • Objekt kennt seine Klasse

- ◆ Ein Objekt kann i. allg. zur Laufzeit ermitteln, zu welcher Klasse es gehört
- ◆ In Java: `getClass()`
- ◆ Operationen sind der Klasse zugeordnet
  - Da alle Objekte zwar unterschiedliche Attributwerte, jedoch gleiche Operationen besitzen



## 2.5.3 Klassen

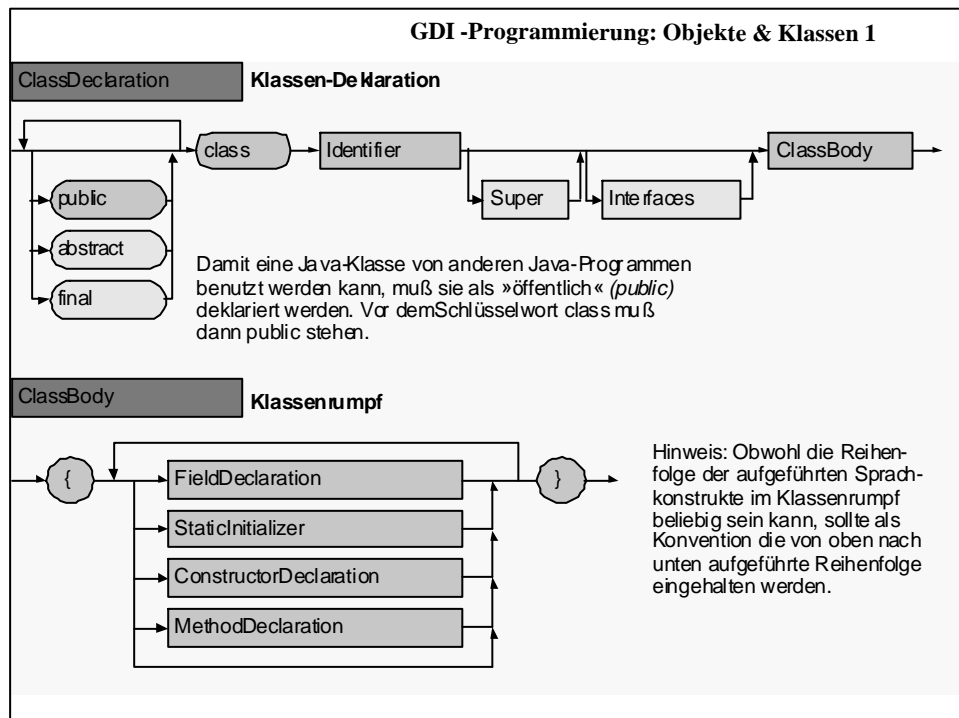
- **Klasse kennt Objekte nicht**
  - ◆ Klasse »weiß« im allgemeinen nicht, welche Objekte sie »besitzt« bzw. welche Objekte von ihr erzeugt wurden
  - ◆ Wenn diese Eigenschaft notwendig ist, muß sie im Einzelfall »von Hand« hinzugefügt werden.

## 2.6 Dann die Praxis: Objekte & Klassen in Java

- **Vorgehensweise**
  1. Überlegung der Objekte und Klassen, bevor ein Java-Programm geschrieben wird
  2. Darstellung in UML-Notation
  3. Schreiben eines Java-Programmgerüsts auf dieser Grundlage
  4. Eine Reihe von Software-Werkzeugen erstellt aus einer UML-Notation automatisch ein Java-Programmgerüst: GO.

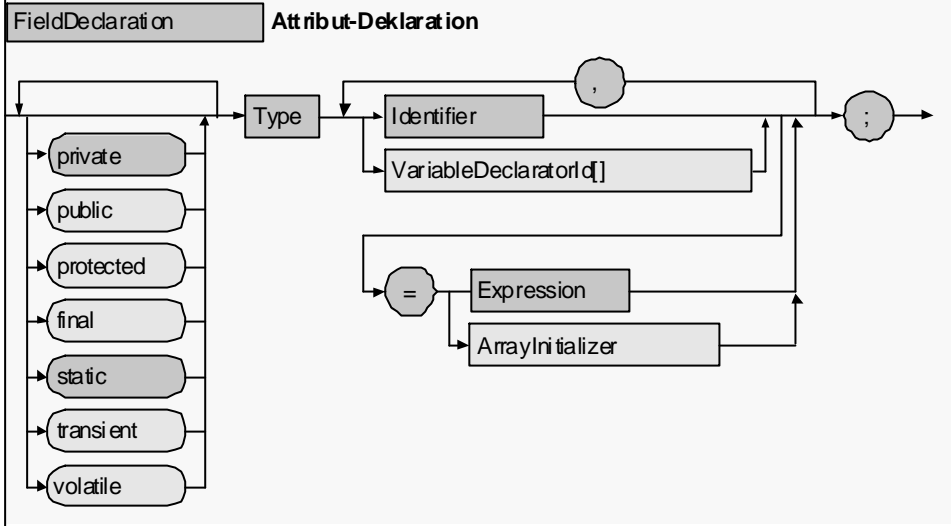
## 2.6.1 Deklaration von Klassen

- **Klassendeklaration (*class declarations*)**
- **3 Teile**
  - ◆ **Klassenname (*class Identifier*)**
  - ◆ **Klassenrumpf (*Class Body*)**
    - **Attributdeklarationen (*Field Declarations*)**
    - **Operationsdeklarationen (*Method Declarations*).**



## 2.6.1 Deklaration von Klassen

## • Java-Syntax für Klassen (2/2)



## 2.6.1 Deklaration von Klassen

- **Attribute**
  - ◆ Müssen durch einen Typ (*type*) spezifiziert werden
- **Vordefinierte Typen**
  - ◆ `String`: Zeichenketten
  - ◆ `int`: ganze Zahlen
  - ◆ `char`: Zeichen
  - ◆ `float`: Gleitkommazahlen
- **Syntax**
  - ◆ Im Gegensatz zur UML stehen in Java die Typangaben vor den Attributnamen
- **Geheimnisprinzip**
  - ◆ `private`.

## 2.6.1 Deklaration von Klassen

### ◊ Beispiele

- ◆ `private String Firmenname;`
- ◆ `private String Adresse;`
- ◆ `private String Telefonnummer;`
- ◆ `private int Auftragssumme;`
- ◆ **Attribute mit gleichem Typ**
  - `private String Firmenname, Adresse, Telefonnummer;`
- ◆ **Attribute mit Initialisierung**
  - `private int Auftragssumme = 0;.`

## 2.6.1 Deklaration von Klassen

### ◊ Objekterzeugung

- ◆ Jede Klasse muß durch Konstruktoren festlegen, wie Objekte von ihr erzeugt werden können

### ◊ Konstruktor

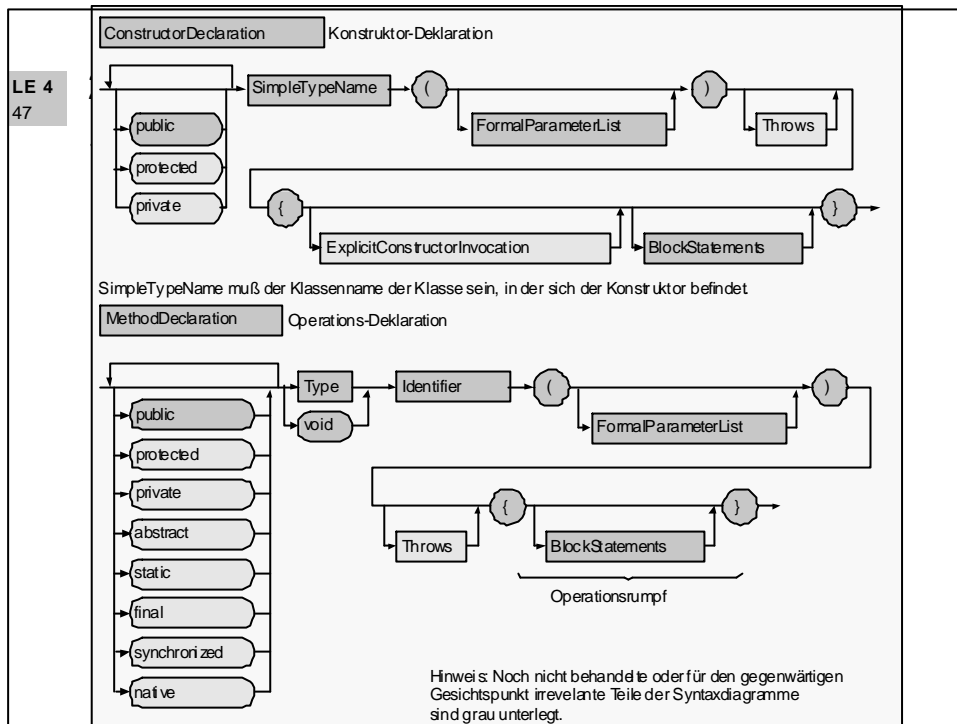
- ◆ Operationsname = Klassenname

### ◊ `public`

- ◆ Muß vor dem Operationsnamen angegeben werden damit von anderen Java-Programmen eine Botschaft an den Konstruktor gesandt werden kann

### ◊ Initialisierung

- ◆ Parameterliste mit Initialisierungswerten kann in runden Klammern aufgeführt werden ( ).



## GDI-Programmierung: Objekte & Klassen 1

LE 4  
48

### 2.6.1 Deklaration von Klassen

#### • Beispiel: Objekterzeugung & Konstruktoren

```
public Kunde (String Name)
//Konstruktor der Klasse Kunde
{
    Firmenname = Name;
    //Das Attribut Firmenname
    //erhält den Wert von Name zugeordnet
}.
```

## 2.6.1 Deklaration von Klassen

- **Operationen**
  - ◆ Ähnlich wie Konstruktoren aufgebaut
  - ◆ Name jedoch frei wählbar
  - ◆ `public`: Von anderen Java-Programmen aufrufbar
  - ◆ Wird eine Operation durch eine Botschaft aktiviert, dann kann die Botschaft Eingabedaten an die Operation übergeben
  - ◆ Eingabedaten
    - Aufzählung der Attributnamen (mit vorangestelltem Attributtyp) in der formalen Parameterliste
  - ◆ Ausgabedaten bzw. Ergebnisdaten
    - Können an Botschaft zurückgegeben werden.

## 2.6.1 Deklaration von Klassen

- Nur ein Wert eines Attributs kann an die auslösende Botschaft übergeben werden
- Der Typ dieses Ergebnisattributs wird vor dem Operationsnamen angegeben
- Übergibt eine Operation keine Ergebnisse, dann steht vor dem Operationsnamen das Schlüsselwort `void`
- **Beispiele**
  - ◆ Operation mit 1 Eingabeparameter und 0 Ausgabeparametern

```
public void setFirmenadresse(String Adresse)
{
    Firmenadresse = Adresse;
}
```

## 2.6.1 Deklaration von Klassen

- ◆ Operation mit 1 Ergebnisparameter und 0 Eingabeparametern

```
◆ public String getFirmenadresse()
{
    return Firmenadresse;
    // Angabe des Ergebnisattributs
}
```

- ◆ Konventionen get, set

- ◆ Wird durch eine Operation ein einzelnes Attribut gelesen, dann sollte die Operation `getAttributname` lauten
- ◆ Wird durch eine Operation ein einzelnes Attribut gesetzt, dann sollte die Operation `setAttributname` heißen.

## 2.6.1 Deklaration von Klassen

- ◆ Fallstudie »Kundenverwaltung«:

- ◆ Klasse Kunde

```
/*Programmname: Kundenverwaltung1
 * Fachkonzept-Klasse: Kunde
 */
public class Kunde
{
    //Attribute
    private String Firmenname, Firmenadresse;
    private int Auftragssumme = 0;
```

## 2.6.1 Deklaration von Klassen

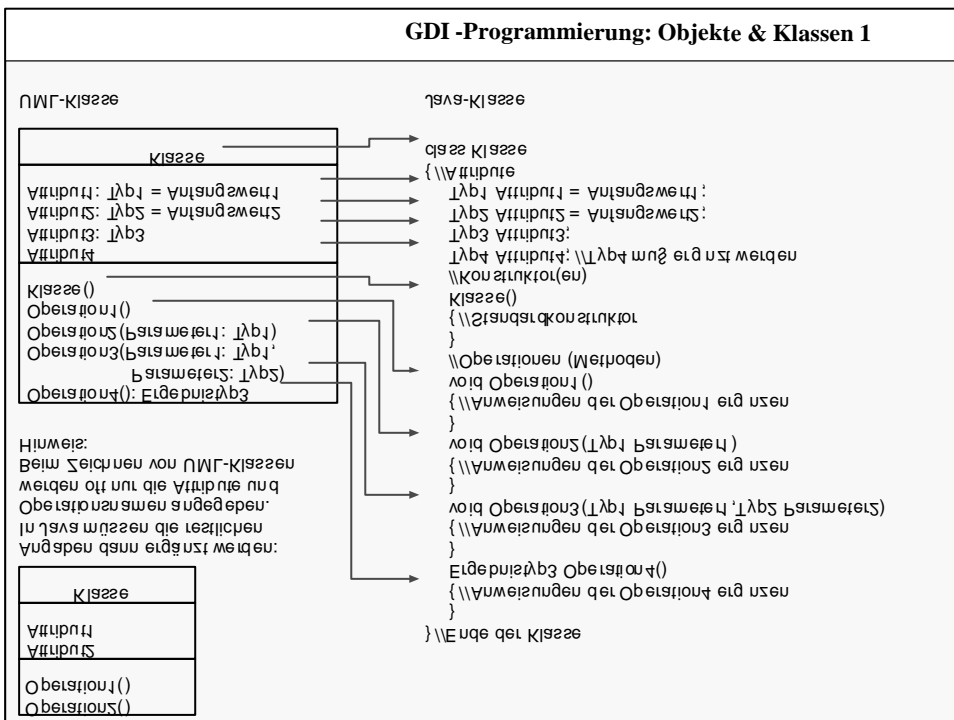
```
//Konstruktor
public Kunde(String Name)
{
    Firmenname = Name;
}
//Kombinierte Schreiboperation
public void aendernAdresse (String Name,
                             String Adresse)
{
    Firmenname = Name;
    Firmenadresse = Adresse;
}.
```

## 2.6.1 Deklaration von Klassen

```
//Schreibende Operationen
public void setFirmenname (String Name)
{
    Firmenname = Name;
}
public void setFirmenadresse (String
                              Adresse)
{
    Firmenadresse = Adresse;
}
public void setAuftragssumme(int Summe)
{
    Auftragssumme = Summe;
}.
```

## 2.6.1 Deklaration von Klassen

```
//Lesende Operationen
public String getFirmenname()
{
    return Firmenname;
}
public String getFirmenadresse()
{
    return Firmenadresse;
}
public int getAuftragssumme()
{
    return Auftragssumme;
}
}.
```



## 2.6.1 Deklaration von Klassen

### • Von UML-Klassen zu Java-Klassen

Folgende Besonderheiten sind zu beachten:

Attributname und Typangabe sind in Java vertauscht, der Doppelpunkt entfällt:

UML `Attribut: Typ = Attributwert`

Java `Typ Attribut = Attributwert;`

Parametername und Typangabe sind in Java ebenfalls vertauscht, der Doppelpunkt entfällt, vor dem

Operationsnamen wird `void` geschrieben:

UML `Operation(Paramater: Typ)`

Java `void Operation(Typ Parameter):`

Hat eine Operation einen Eigenschaftsname, dann wird der Eigenschaftsname vor dem Operationsnamen geschrieben:

UML `Operation(-Eigenschaft)`

Java `Eigenschaft Operation():`

## 2.6.1 Deklaration von Klassen

### • Fachkonzept-Klasse

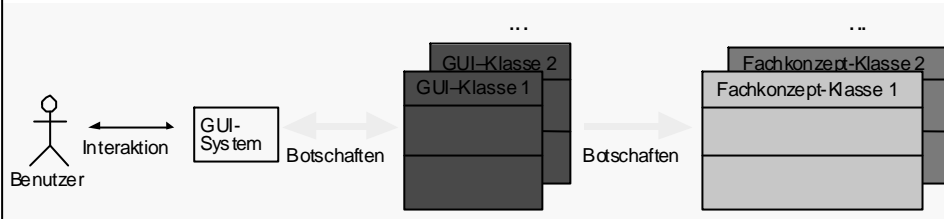
- ◆ Realisierung der im Pflichtenheft festgelegten fachlichen Anforderungen

### • Mehrere Klassen in einer Datei

- ◆ In Java möglich
- ◆ Java-Compiler erzeugt von jeder Klasse, die in einer Datei abgelegt ist, eine gesonderte Datei mit der Endung `.class`.
- ◆ Eine Klasse der Datei muß mit dem Dateinamen übereinstimmen.

## 2.6.2 GUI-Klassen

- **Entwurfsprinzip: Trennung GUI – Fachkonzept**
  - ◆ **Grundlegendes Prinzip beim Entwurf von Software-Systemen**



## 2.6.2 GUI-Klassen

- **Charakteristika**
  - ◆ **Fachkonzept-Klassen dürfen nicht direkt mit GUI-System kommunizieren**
  - ◆ **Zwischen GUI-System und Fachkonzept-Klassen befinden sich GUI-Klassen, die den Fachkonzept-Klassen Botschaften schicken**
  - ◆ **In der Regel kennen die Fachkonzept-Klassen die GUI-Klassen nicht**
  - ◆ **Von den Fachkonzept-Klassen werden keine Botschaften zu den GUI-Klassen geschickt.**

## 2.6.2 GUI-Klassen

- ◊ **Java-Konvention**
  - ◆ Jede Klasse in einer eigenen Datei abspeichern
- ◊ **Namenskonvention**
  - ◆ GUI-Klassen am Ende immer den Zusatz **GUI** anhängen
- ◊ **GUI-Klasse**
  - ◆ Baut die Benutzungsoberfläche auf
    - Bestehend aus Führungstexten, Textfeldern usw.
  - ◆ Liest Eingaben von Benutzern und gibt Ergebnisse aus.

## 2.6.3 Erzeugen und Referenzieren von Objekten

- ◊ **Objekte erzeugen und referenzieren:**
  - 1 Attribute zum Speichern der Referenzen deklarieren
    - Referenz-Attribute
      - Deklaration wie »normale« Attribute
      - Syntax = Syntax für Attributdeklaration
      - Typbezeichnung = Klassenname, auf die die Referenz-Attribute später zeigen sollen
      - Deklaration: noch *keine* Erzeugung der Objekte
  - 2 Objekte mit dem `new`-Operator erzeugen
  - 3 Speicheradressen der erzeugten Objekte müssen in den deklarierten Referenz-Attributen gespeichert werden.

## 2.6.3 Erzeugen und Referenzieren von Objekten

### ♦ Fallstudie »Kundenverwaltung«

#### 1 Deklarieren von Referenz-Attributen

##### a z.B. `ersterKunde`, `zweiterKunde`

##### • Namenskonventionen

- Bezeichnung irgendeines Objekt einer Klasse mit dem Präfix `ein`, `a` oder `erster`, `zweiter` usw. gefolgt vom Klassennamen
- Präfix ist immer kleingeschrieben
  - ☞ Z.B. `einKunde`, `aPerson`, `ersterRoboterarm`, `zweiterRoboterarm`

##### b Typ = Klasse, auf die die Referenzen zeigen

- Hier Klasse `Kunde`:  
`Kunde ersterKunde, zweiterKunde;`

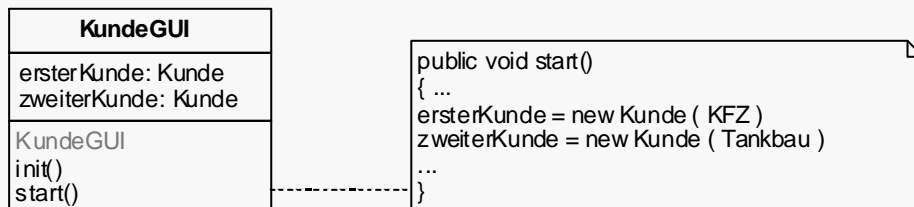
##### c Vor Klassennamen `private`

- z.B. `private Kunde ersterKunde, zweiterKunde;.`

## 2.6.3 Erzeugen und Referenzieren von Objekten

### ♦ Beispiel: Klasse `KundeGUI`

#### ◆ Erzeugung von 2 Kundenobjekten in der Operation `start()`



## 2.6.3 Erzeugen und Referenzieren von Objekten

### 2 Objekte durch Konstruktor-Aufruf erzeugen

- Gewünschte Konstruktor wird mit vorangestellten Operator `new` aufgerufen

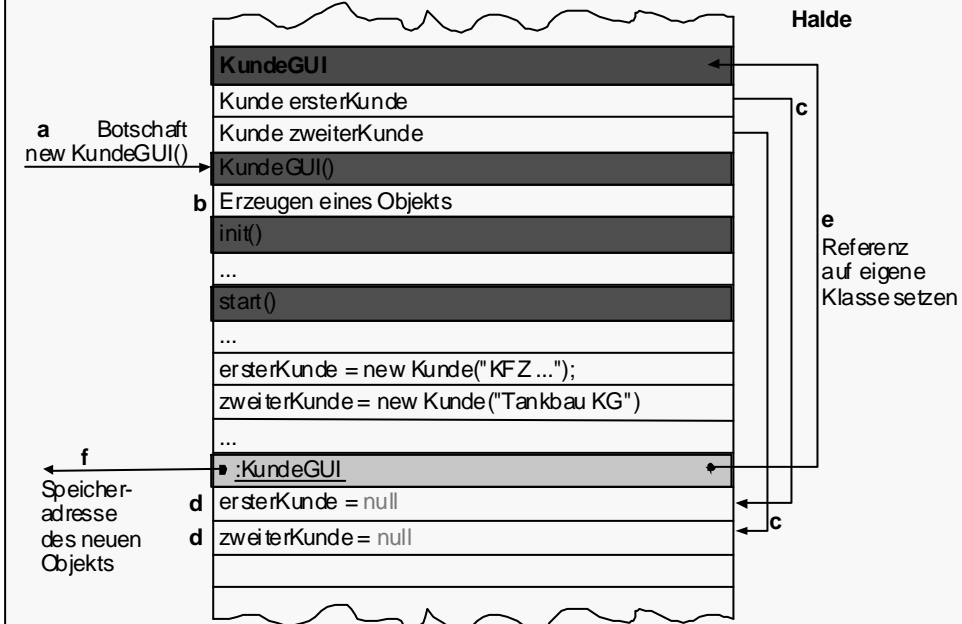
- Z.B.

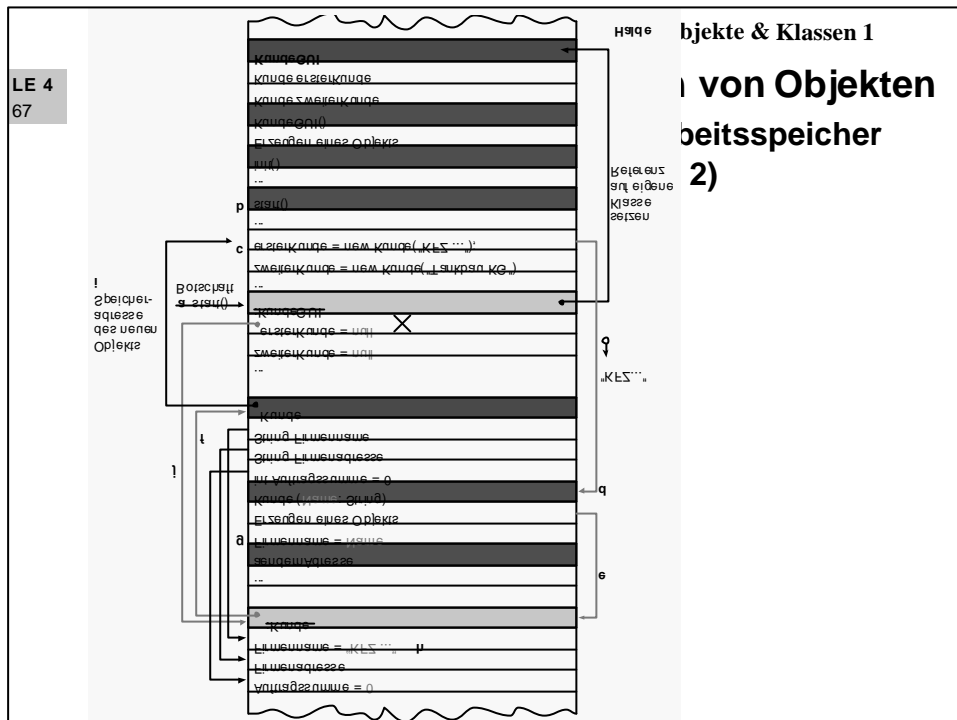
- `new KundeGUI()`
- `new Kunde("KFZ-Zubehoer GmbH")`
- `new Kunde("Tankbau KG")`

- Halde (*heap*)

- ◆ Generell werden alle Klassen und Objekte in einem speziellen Arbeitsspeicherbereich gespeichert.

## 2.6.3 Erzeugen und Referenzieren von Objekten





## 2.6.3 Erzeugen und Referenzieren von Objekten

### 3 Speicheradresse des neu erzeugten Objekts dem Referenz-Attribut zuweisen

- Botschaftssender ist die Anweisung `ersterKunde = new Kunde("KFZ");`;
  - Die zurückgegebene Speicheradresse wird in der Speicherzelle des Referenz-Attributs `ersterKunde` gespeichert
- ◆ Analog: Erzeugung des Objekts `weiterKunde`.

## 2.6.3 Erzeugen und Referenzieren von Objekten

### • UML-Darstellung dynamischer Abläufe

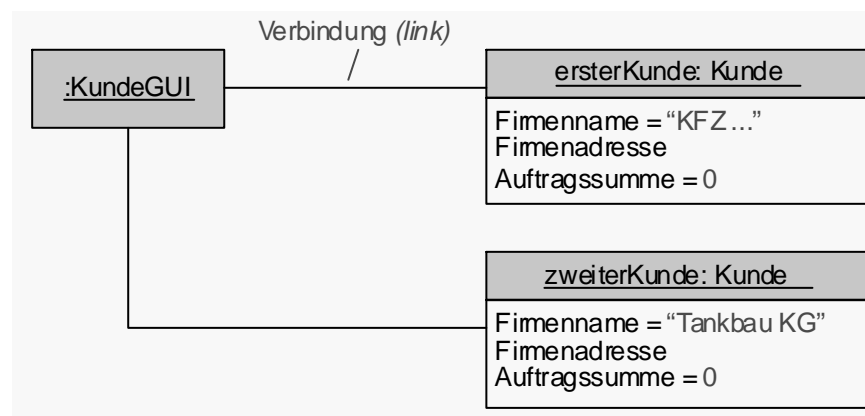
#### ◆ 3 verschiedene Möglichkeiten

- Objektdiagramme (*object diagrams*)
- Kollaborationsdiagramme (*collaboration diagrams*)
- Sequenzdiagramme (*sequence diagrams*).

## 2.6.3 Erzeugen und Referenzieren von Objekten

### • Objektdiagramme (*object diagrams*)

#### ◆ Beispiel



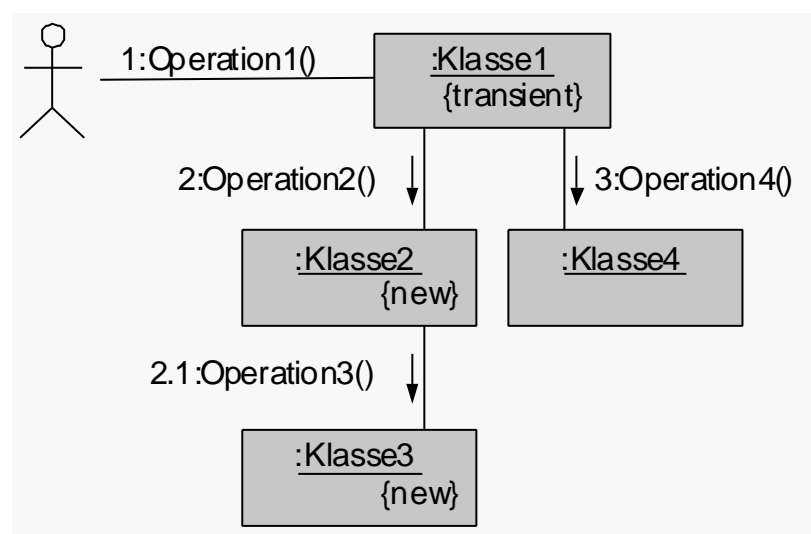
## 2.6.3 Erzeugen und Referenzieren von Objekten

### • Kollaborationsdiagramm (*collaboration diagram*)

- ◆ Besser: Zusammenarbeitsdiagramm
- ◆ Erweiterung des Objektdiagramms um Botschaften
- ◆ Zeigt diejenigen Objekte, die für die Ausführung bestimmter Operationen relevant sind
- ◆ Akteure (Benutzer) können eingetragen werden.

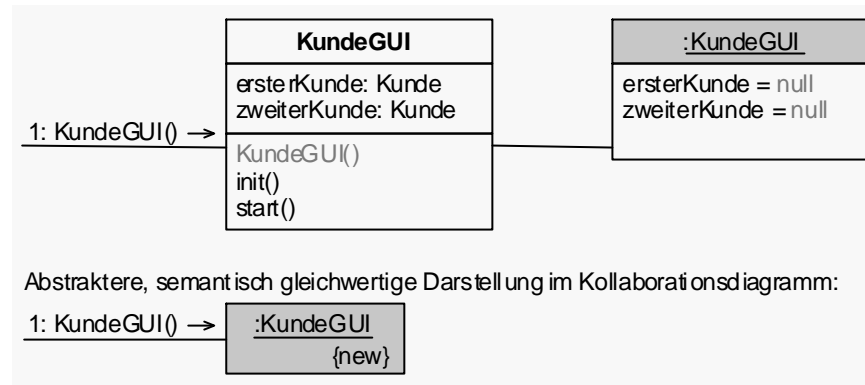
## 2.6.3 Erzeugen und Referenzieren von Objekten

### • Kollaborationsdiagramm



## 2.6.3 Erzeugen und Referenzieren von Objekten

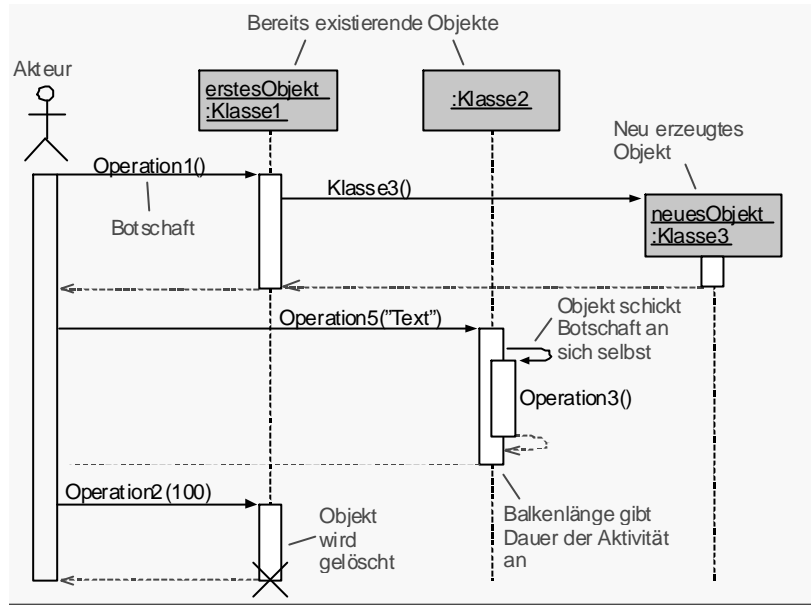
- Ausführliche vs. abstrakte Darstellung der Objekterzeugung im Kollaborationsdiagramm



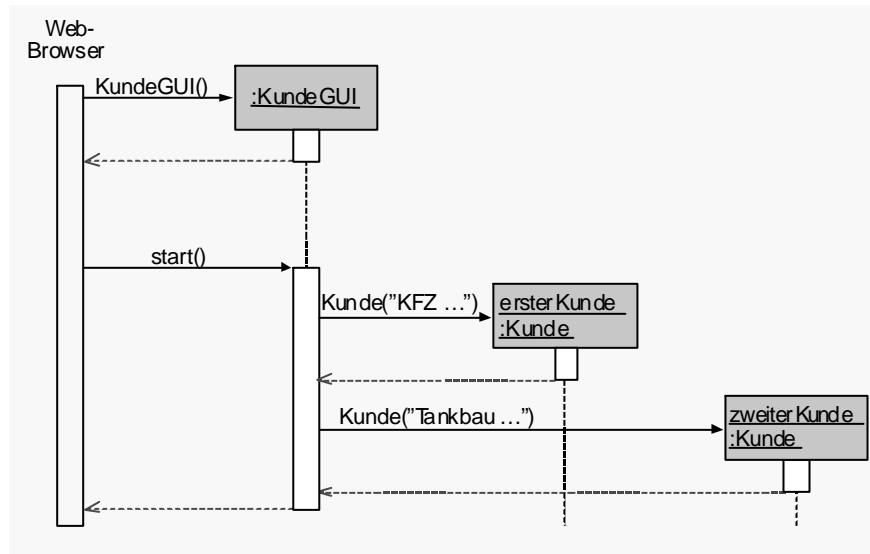
## 2.6.3 Erzeugen und Referenzieren von Objekten

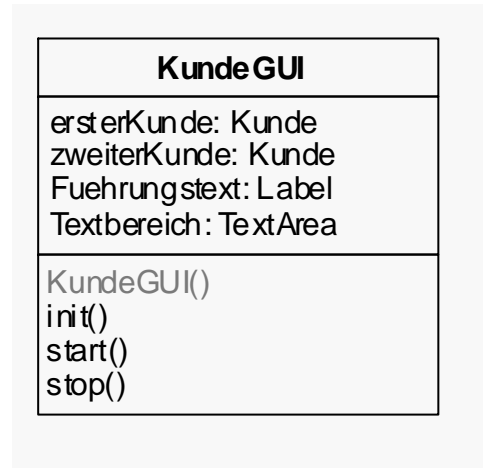
- Sequenzdiagramm
  - ◆ Noch genauere zeitliche Darstellung
  - ◆ Verzicht auf Attributangaben
  - ◆ Objekterzeugungsvorgang wird verkürzt dargestellt.

### 2.6.3 Erzeugen und Referenzieren von Objekten



### 2.6.3 Erzeugen und Referenzieren von Objekten



**2.6.4 Senden v. Botschaft. & Ausf. v. Operationen**♦ **Beispiel Fallstudie »Kundenverwaltung«****2.6.4 Senden v. Botschaft. & Ausf. v. Operationen**♦ **Zugehörige Java-Klasse**

```

/*Programmname: Kundenverwaltung1
 * GUI-Klasse: KundeGUI (Java-applet)
 */
import java.awt.*;
import java.applet.*;
public class KundeGUI extends Applet
{
    //Attribute
    //Deklarieren von zwei Kundenobjekten
    private Kunde ersterKunde, zweiterKunde;
    //Deklarieren der Interaktionselemente
    //Führungstext und Textbereich.

```

**2.6.4 Senden v. Botschaft. & Ausf. v. Operationen**

```
//{{DECLARE_CONTROLS
Label Fuehrungstext;
TextArea Textbereich;
//}}}
//Operationen
public void init()
{ //Erzeugen der Interaktionselemente und
  //Zuordnung von Attributwerten
  //{{INIT_CONTROLS
  setLayout(null);
  setSize(430,270);
  Fuehrungstext = new Label("Kundenverwaltung");
  Fuehrungstext.setBounds(36,24,168,28);
  Fuehrungstext.setFont(new Font("Dialog",
                                Font.BOLD, 12));
```

**2.6.4 Senden v. Botschaft. & Ausf. v. Operationen**

```
add(Fuehrungstext);
Textbereich = new TextArea("",0,0,
                          TextArea.SCROLLBARS_VERTICAL_ONLY);
Textbereich.setBounds(36,60,360,156);
Textbereich.setForeground(new Color(0));
add(Textbereich);
//}}}
}
public void start()
{ //Attribute zum Merken der Ergebn.der Botschaften
  String MerkeText;
  int MerkeZahl;
  //Erzeugen von 2 Obj.durch Aufruf des Konstruktors
  ersterKunde = new Kunde("KFZ-Zubehoer GmbH");
  zweiterKunde = new Kunde("Tankbau KG");
```

LE 4  
81**2.6.4 Senden v. Botschaft. & Ausf. v. Operationen**

```

//Adresse eintragen
ersterKunde.setFirmenadresse("44137 Dortmund,
                             Poststr. 12");

//Adresse ändern
zweiterKunde.aendernAdresse("Tankbau & Partner
                             KG", "44867 Bochum, Suedstr. 23");

//Auftragssumme eintragen
ersterKunde.setAuftragssumme(15000);
//Anzeigen der Attributinhalt im Textbereich
MerkeText = ersterKunde.getFirmenname();
//append hängt den Text an den vorh. Text an
Textbereich.append(MerkeText);
// Mit "\n" wird auf ei. neue Zeile gewechselt
Textbereich.append("\n");
MerkeText = ersterKunde.getFirmenadresse();
Textbereich.append(MerkeText+"\n");

```

LE 4  
82**2.6.4 Senden v. Botschaft. & Ausf. v. Operationen**

```

MerkeZahl = ersterKunde.getAuftragssumme();
//Umwandl. ganze Zahl in Zeichenkette
MerkeText = String.valueOf(MerkeZahl);
Textbereich.append(MerkeText+"\n"+"");
//Anzeigen der Attributinhalt -- Kurzform
Textbereich.append(zweiterKunde.getFirmenname()+
                  "\n");
Textbereich.append(zweiterKunde.getFirmenadresse()+
                  "\n");
MerkeZahl = zweiterKunde.getAuftragssumme();
//Umwandlung ganze Zahl in Zeichenkette
MerkeText = String.valueOf(MerkeZahl);
Textbereich.append("Auftragssumme: " +
                  MerkeText+"\n");
}

```

## 2.6.4 Senden v. Botschaft. & Ausf. v. Operationen

```
public void stop()  
{  
    Textbereich.append("Stop"+"\\n");  
}  
}
```

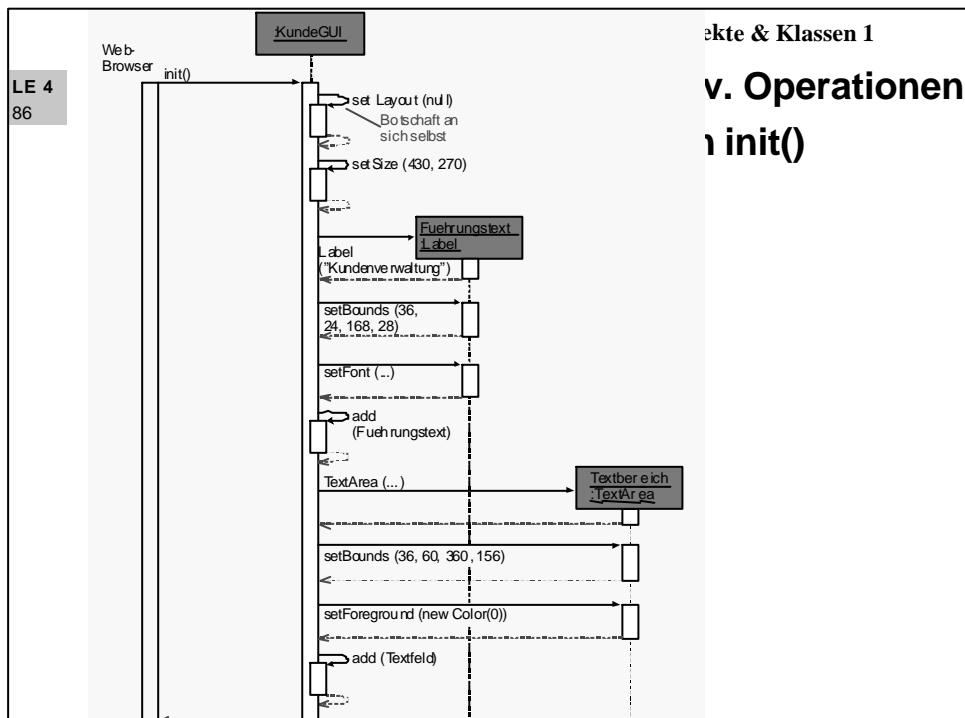
## 2.6.4 Senden v. Botschaft. & Ausf. v. Operationen

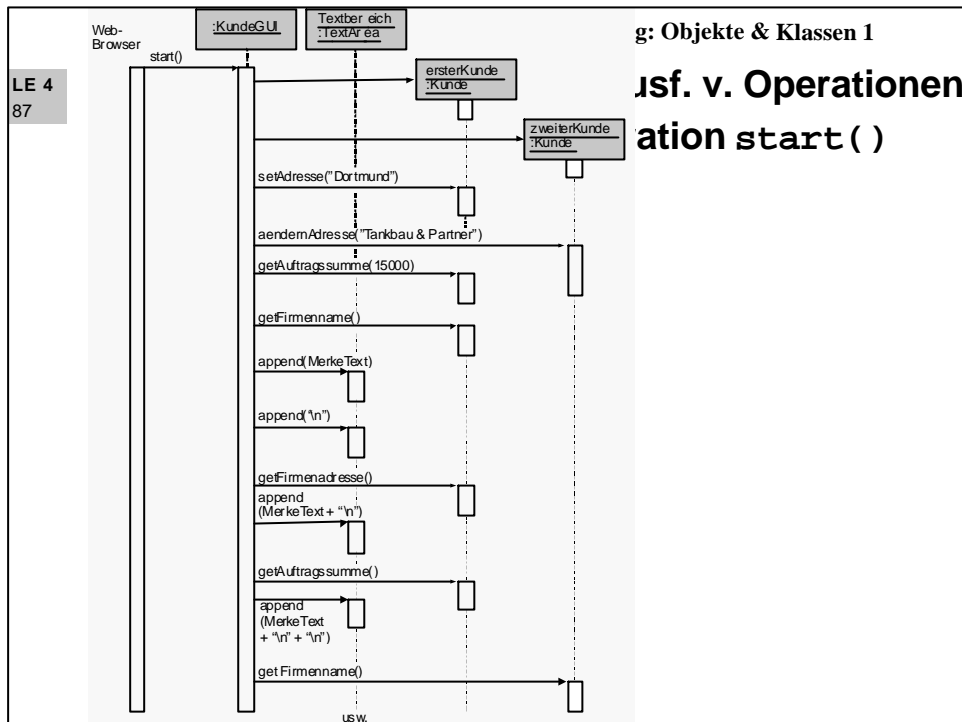
- Ausgabe des Java-Programms  
»Kundenverwaltung«



## 2.6.4 Senden v. Botschaft. & Ausf. v. Operationen

- **Botschaft Objekt .Operation**
  - ◆ **Namen des Objekts, an das die Botschaft gesandt werden soll**
  - ◆ **Punkt und anschl. Name der Operation, die ausgeführt werden soll**
  - ◆ **In Klammern können Parameterwerte an die Operation übergeben werden**
- **Botschaft an sich selbst**
  - ◆ **Nur der Operationsname, u.U. mit Parametern.**





LE 4  
88

## 2.6.5 Löschen von Objekten

♦ **Java:**

- ◆ **Der Programmierer muß sich nicht um das Löschen von Objekten kümmern**
- ◆ **In unregelmäßigen Abständen findet automatisch eine Speicherbereinigung (*garbage collection*) statt**
- ◆ **Die Java-VM prüft, ob es im Halde Speicher Objekte gibt, auf die keine Referenz mehr zeigt**
- ◆ **Diese Objekte werden automatisch gelöscht.**

- ♦ **Danke!**
- ♦ **Aufgaben**
  
- ♦ **Diese Präsentation bzw. Teile dieser Präsentation enthalten Inhalte und Grafiken des Lehrbuchs der Grundlagen der Informatik von Helmut Balzert, Spektrum Akademischer Verlag, Heidelberg 1999**

