

## **Teil**

# **Objektorientierte Programmierung**

## **Unterrichtseinheit 24**

### **Klassen und Objekte (Fortgeschrittene Konzepte)**

**Dr. Dietrich Boles**

- Klassendefinition
- Klassenattribute
- Klassenmethoden
- Konstanten
- this
- Methoden
- Subobjekte
- Delegation
- Objektarrays
- Objektreferenzen
- Default-Konstruktor
- Copy-Konstruktor
- Gleichheit
- main-Funktion
- Beispiel Buchhandlung
- Zusammenfassung

```
<klassen-def> ::= ["public"] "class" <bezeichner> "{"  
                { <attr-def>          | <static-attr-def>  
                  <methoden-def> | <static-meth-def>  
                  <konstruktor-def>  
                }  
                "}"
```

**<attr-def> ::= <variablen-def>**

**<static-attr-def> ::= "static" <variablen-def>**

**<methoden-def> ::= <funktionen-def> (ohne static)**

**<static-meth-def> ::= <funktionen-def> (mit static)**

**<konstruktor-def> ::= <methoden-def> (ohne Funktionstyp)**

- Attribute mit vorangesetztem Schlüsselwort **static**
- für jede Klasse existiert nur eine Instanz eines Klassen-Attributs
- alle Objekte der Klasse haben Zugriff darauf

## Beispiel:

```
class Hamster {  
    static int anzahlHamster = 0; // Klassenattribut
```

```
    Hamster(int r, int s, int b, int k) {
```

```
        ...
```

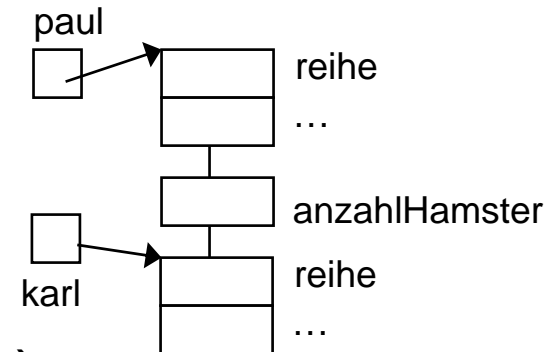
```
        anzahlHamster = anzahlHamster + 1;
```

```
    }
```

```
} ...
```

```
Hamster paul = new Hamster(1,1,Hamster.OST,1);
```

```
Hamster karl = new Hamster(2,2,Hamster.OST,2);
```



- Methode mit Schlüsselwort **static**
- Zweck 1: "Operationen" auf Klassen-Attributen
- Zweck 2: Realisierung von Funktionen ohne Objektbezug
- Aufruf: **<Klassenname>.<Funktionsaufruf>**
- Klassenmethoden haben keinen Zugriff auf Instanz-Attribute und Instanz-Methoden!

```
class Hamster {  
    static int anzahlHamster = 0;    // Klassenattribut  
  
    static int getAnzahlHamster() {    // Zweck 1  
        return anzahlHamster;  
    }  
} ...  
Hamster paul = new Hamster(1,1,Hamster.OST,1);  
Hamster karl = new Hamster(2,2,Hamster.OST,2);  
int anzahl = Hamster.getAnzahlHamster(); // anzahl == 2
```

```
class Hamster { // Zweck 2
    static Hamster getStandardHamster() { ... }
    ...
}

class Territorium { // Zweck 2
    static int getAnzahlReihen() { ... }
    static int getAnzahlSpalten() { ... }
    static int getAnzahlKoerner(int reihe, int spalte) { ... }
    ...
}

void main() {
    Hamster paul = Hamster.getStandardHamster();
    if (Territorium.getAnzahlKoerner(
        paul.getReihe(), paul.getSpalte()) > 0)
        paul.nimm();
}
```

- `final-static`-Attribut = Konstante
- kann nachträglich nicht mehr verändert werden
- Vereinbarung: ausschließlich Großbuchstaben!

```
class Hamster {  
    final static int NORD = 0;  
    final static int OST = 1;  
    final static int SUED = 2;  
    final static int WEST = 3;  
    ...  
}  
  
void main() {  
    Hamster paul = new Hamster(1,1,Hamster.OST,1);  
    ...  
    if (paul.getBlickrichtung() == Hamster.OST) {  
        paul.linksUm();  
    }  
}
```

- Schlüsselwort; zwei Einsatzzwecke:
  - stellt das Objekt heraus, für das ein Attribut oder eine Methode aufgerufen wird
  - Zugriff auf Attribute bei gleichnamigen lokalen Parametern

## Beispiel:

```
class Hamster {  
    int reihe, spalte, blickrichtung, anzahlKoerner;  
  
    Hamster(int reihe, int spalte, int b, int k) {  
        this.reihe = reihe;  
        this.spalte = spalte;  
        this.blickrichtung = b;  
        this.anzahlKoerner = k;  
    }  
    void vor() {  
        if (!this.vornFrei()) // Programmabbruch  
            ...  
    } }  
}
```



- für alle Attribute, deren Wert von außen ermittelt oder manipuliert werden soll (aber auch nur für die!) sollten entsprechende `get`- bzw. `set`-Methoden definiert werden
- Vereinbarung: `get` / `set` + Attribut-Name mit großem Anfangsbuchstaben
- Zugriff auf die Attribute von außerhalb der Klassendefinition ausschließlich über diese Methoden

## Beispiele:

```
class Hamster {  
    int reihe;  
    int getReihe() { return this.reihe; }  
} ...
```

```
Hamster paul = new Hamster(1,1,Hamster.OST,3);  
int r = paul.reihe;           // verboten  
int r = paul.getReihe();
```

- zwei oder mehrere Methoden bzw. Konstruktoren einer Klasse können denselben Namen besitzen, wenn
  - sie eine unterschiedliche Anzahl an Parametern besitzen oder
  - wenn sich die Parametertypen an entsprechender Stelle unterscheiden

```
class Hamster {  
    ...  
    Hamster(int r, int s, int b, int k) {...}  
    Hamster(Hamster exHamster) {  
        this.reihe = exHamster.reihe;  
        this.spalte = exHamster.spalte;  
        this.blickrichtung = exHamster.blickrichtung;  
        this.anzahlKoerner = exHamster.anzahlKoerner;  
    }  
}
```

# Methoden / Überladen von Methoden (2)

```
void vor() {...}
void vor(int anzahl) { // moeglich
    while (this.vornFrei() && anzahl > 0) {
        this.vor();
        anzahl--;
    }
}
...
}

void main() {
    Hamster paul = new Hamster(1,2,Hamster.OST, 3);
    Hamster karl = new Hamster(paul);
    paul.vor();
    karl.vor(5);
}
```

- Subobjekt = Attribut von Klassentyp

## Beispiel:

```
class Konto {
    int    nummer;  double saldo;
    Konto(int n) { nummer = n; saldo = 0.0; }
    void einzahlen(double b) { saldo += b; }
}

class Person {
    String name;
    Konto sparbuch;      // Subobjekt!
    Person(String n) {
        name = n;
        sparbuch = new Konto(4711);
        sparbuch.einzahlen(100.0);  // "Geburtsgutschrift"
    }
}
```

- Delegation = Weiterreichen von Nachrichten an Subobjekte

## Beispiel:

```
class Gehirn {  
    void denken() {...}  
}  
  
class Mensch {  
    Gehirn hirn;                // Subobjekt  
    Mensch() {  
        hirn = new Gehirn();  
    }  
    void denken() {  
        hirn.denken();          // Delegation  
    }  
}
```

```
class Konto {
    int    nummer;    double saldo;
    Konto(int n) { nummer = n; saldo = 0.0; }
    void einzahlen(double b) { saldo += b; }
}

class Bank {
    int blz;    // Bankleitzahl
    Konto[] konten;
    Bank(int zahl) {
        blz = zahl;
        konten = new Konto[1000];
        for (int i=0; i<1000; i++)
            konten[i] = new Konto(i);
    }
    void einzahlen(int nummer, double betrag) {
        konten[nummer].einzahlen(betrag);
    } }
}
```

```
class Konto {  
    int    nummer; double saldo;  
    Konto(int n) { nummer = n; saldo = 0; }  
    void einzahlen(double betrag) { saldo += betrag; }  
    void abheben(double betrag) { saldo -= betrag; }  
    void ueberweisen(Konto aufKonto, double betrag) {  
        this.abheben(betrag);  
        aufKonto.einzahlen(betrag);  
    }  
} ...
```

```
Konto k1 = new Konto(4711);  
Konto k2 = new Konto(4712);  
k1.einzahlen(200.0);  
k1.ueberweisen(k2, 100.0);
```

```
class Konto {
    int    nummer;  double saldo;
    Konto(int n) { nummer = n; saldo = 0.0; }
    void einzahlen(double betrag) { saldo += betrag; }
}

class Bank {
    Konto[] konten;
    Bank() {
        konten = new Konto[1000];
        for (int i=0; i<1000; i++) konten[i] = new Konto(i);
    }
    Konto liefereKonto(int nummer) {
        return konten[nummer];
    } } ...

Bank olb = new Bank();
Konto k = olb.liefereKonto(3);
k.einzahlen(200.0);
```



- Default-Konstruktor = Konstruktor ohne Parameter

```
class Hamster {  
    Hamster() { // Default-Konstruktor  
        Hamster s = Hamster.getStandardHamster();  
        this.reihe = s.getReihe();  
        this.spalte = s.getSpalte();  
        this.blickrichtung = s.getBlickrichtung();  
        this.anzahlKoerner = s.getAnzahlKoerner();  
    }  
}  
...  
Hamster paul = new Hamster();  
paul.vor();
```

- Konstruktor mit Objekt vom selben Typ als Parameter
- Zweck: "Clonen" von Objekten

```
class Hamster {  
    Hamster(Hamster ex) { // Copy-Konstruktor  
        this.reihe = ex.reihe;  
        this.spalte = ex.spalte;  
        this.blickrichtung = ex.blickrichtung;  
        this.anzahlKoerner = ex.anzahlKoerner;  
    } // alle Attribute von this und ex sind wertegleich  
}  
...  
Hamster paul = new Hamster(3, 4, Hamster.OST, 7);  
Hamster paulsZwillingsBruder = new Hamster(paul);
```

- Unterschied: Referenzgleichheit und Objektgleichheit
- `equals`-Methode (wird später noch etwas korrigiert!)

```
class Hamster {  
    boolean equals(Hamster anderer) { // Objektgleichheit  
        return this.reihe == anderer.reihe &&  
            this.spalte == anderer.spalte &&  
            this.blickrichtung == anderer.blickrichtung &&  
            this.anzahlKoerner == anderer.anzahlKoerner;  
    }  
} ...
```

```
Hamster paul = new Hamster(3, 4, Hamster.OST, 7);  
Hamster zwilling = new Hamster(paul);  
boolean refGleichheit = paul == zwilling ;           // false  
boolean wertGleichheit = paul.equals(zwilling);      // true
```

```
public static void main(String[] args) { ... }
```

- Klassenmethode
- besitzt eine Klasse X eine solche main-Funktion und wird der Java-Interpreter mit `java X` aufgerufen, so startet das Programm automatisch mit dem Aufruf der Funktion `X.main`
- `args`: Übergabeparameter der Shell

```
class Echo {  
    public static void main(String[] args) {  
        for (int i=0; i<args.length; i++)  
            System.out.println(args[i]);  
    }  
}
```

```
$ java Echo hello world  
hello  
world  
$
```

```
$ java Echo moin  
moin  
$
```

Demo

- Szenario:
  - Buchhandlung beschafft Bücher
  - Bücher können gekauft werden
  
- Klassen
  - Buch
  - Buchhandlung
  - BuchSzenario (Hauptprogramm)

Demo

# Beispiel Buchhandlung (2)

```
class Buch { // repraesentiert ein Buch
    int isbn;
    String titel;

    Buch(int isbn, String titel) {
        this.isbn = isbn;
        this.titel = titel;
    }

    int getIsbn() {
        return this.isbn;
    }

    String getTitel() {
        return this.titel;
    }
}
```

# Beispiel Buchhandlung (3)

```
class Buchhandlung { // realisiert einfache Buchhandlung
    String name;
    Buch[] buecher;
    int next;

    Buchhandlung(String name, int maxGroesse) {
        this.name = name;
        this.buecher = new Buch[maxGroesse];
        for (int i=0; i<maxGroesse; i++)
            this.buecher[i] = null;
        this.next = 0;
    }

    void buchEintragen(Buch buch) {
        if (this.next < this.buecher.length)
            this.buecher[this.next++] = buch;
    }
}
```

## Beispiel Buchhandlung (4)

```
Buch buchVerkaufen(int isbn) {  
    for (int i=0; i<this.next; i++) {  
        if ((this.buecher[i] != null) &&  
            (buecher[i].getIsbn() == isbn)) {  
            Buch buch = buecher[i];  
            buecher[i] = null;    // Buch wird ausgetragen  
            return buch;  
        }  
    }  
    return null;  
}
```

```
boolean buecherVorhanden() {  
    for (int i=0; i<this.next; i++)  
        if (this.buecher[i] != null) return true;  
    return false;  
}  
}
```



# Beispiel Buchhandlung (5)

```
class BuchSzenario {
    public static void main(String[] args) {
        // Buchhandlung erzeugen und initialisieren
        String name = IO.readString("Name der Buchhandlung: ");
        int anzahlBuecher = IO.readInt("Max Anzahl Buecher: ");
        Buchhandlung handlung =
            new Buchhandlung(name, anzahlBuecher);

        // Buecher bestellen
        IO.println("Einkauf");
        int anzahlBestell =
            IO.readInt("Anzahl an Buchbestellungen: ");
        for (int i=0; i<anzahlBestell; i++) {
            Buch buch =
                new Buch(IO.readInt("ISBN: "),
                    IO.readString("Titel: "));
            handlung.buchEintragen(buch);
        }
    }
}
```

## Beispiel Buchhandlung (6)

```
// Buecher verkaufen
IO.println("Buchverkauf");
while (handlung.buecherVorhanden()) {
    int isbn = IO.readInt("ISBN: ");
    Buch buch = handlung.buchVerkaufen(isbn);
    if (buch == null) {
        IO.println("Das Buch haben wir leider nicht!");
    } else {
        IO.println("Hier haben Sie das Buch mit dem Titel: " +
            buch.getTitel());
    }
}

// Ausverkauft
IO.println("Ausverkauft!");
}
```

- static-Elemente: Klassen-spezifische Elemente
- this: das "betroffene" Objekt
- Subobjekt: Attribut vom Klassentyp
- Klassen sind Referenztypen!
- static-main-Funktion: Start eines Java-Programms