Programmierkurs Java

UE 11 – Klassen und Objekte III

Dr.-Ing. Dietrich Boles

Gliederung

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

Klassendefinition (zweite Erweiterung)

```
<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)
```

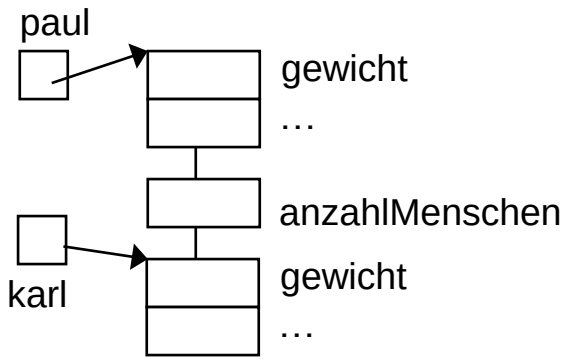
Klassen-Attribute

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

```
class Mensch {
  int gewicht; ...
  static int anzahlMenschen = 0;  // Klassenattribut

  Mensch(boolean mann, int gr, double gew) {
    ...
    anzahlMenschen = anzahlMenschen + 1;
  }
} ...

Mensch paul = new Mensch(true, 181, 72.0);
Mensch karl = new Mensch(true, 192, 86.5);
```



Klassen-Methoden (1)

- 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 Mensch {
  static int anzahlMenschen = 0;  // Klassenattribut
  ...
  static int getAnzahlMenschen() {  // Zweck 1
    return anzahlMenschen;
  }
} ...
Mensch paul = new Mensch(true, 181, 72.0);
Mensch karl = new Mensch(true, 192, 86.5);
int anzahl = Mensch.getAnzahlMenschen(); // anzahl == 2
```

Klassen-Methoden (2)

```
class Math{  // Zweck 2
  static double abs(double value) { ... }
  static double max(double v1, double v2) { ... }
  static double pow(double value, double power) { ... }
  static double sqrt(double value) { ... }
  ...
} ...

double i1 = IO.readDouble("Zahl 1: ");
double i2 = IO.readDouble("Zahl 2: ");
IO.println("Betrag von " + i1 + "=" Math.abs(i1));
double pyt = Math.sqrt(Math.pow(2, i2) + Math.pow(2, i2));
```

Konstanten

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

```
class Mensch {
    final static boolean MANN = true;
    final static boolean FRAU = false;
    ...
    Mensch(boolean geschlecht, int gr, double gew) {
        ...
    }

} ...

Mensch paul = new Mensch(Mensch.MANN, 181, 72.0);
Mensch maria = new Mensch(Mensch.FRAU, 165, 55.0);
```

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

```
class Mensch {
  boolean maennlich;
  int groesse; // in cm
  double gewicht; // in kg

  Mensch(boolean maennlich, int groesse, double gewicht) {
   this.maennlich = maennlich;
   this.groesse = groesse;
   this.gewicht = gewicht;
 }

  boolean hatOrdentlichesGewicht() {
    return this.gewicht <= this.berechneNormalgewicht() &&
           this.gewicht >= this.berechneIdealgewicht();
} }
```

Methoden / Überladen von Methoden (1)

- 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 Bruch {
  int zaehler;
  int nenner;

  Bruch() {
    this.zaehler = 0;
    this.nenner = 1;
  }

  Bruch(int zaehler, int nenner) {
    this.zaehler = zaehler;
    this.nenner = nenner;
  }
```

Methoden / Überladen von Methoden (2)

```
  void mult(int zahl) {
    this.zaehler *= zahl;
  }

  void mult(Bruch bruch) {
    this.zaehler *= bruch.zaehler;
    this.nenner *= bruch.nenner;
  }
  ...
} ...

Bruch bruch1 = new Bruch();
Bruch bruch2 = new Bruch(3, 4);
bruch1.mult(78);
bruch2.mult(bruch1);
```

Subobjekte

- Subobjekt = Attribut von Klassentyp

```
class Konto {
  int    nummer;  double saldo;
  Konto(int n) { nummer = n; saldo = 0.0; }
  void einzahlen(double b) { saldo += b; }
}
class Person {
  static int nr = 0;
  String name;
  Konto sparbuch;      // Subobjekt!
  Person(String n) {
    name = n;
    sparbuch = new Konto(nr++);
    sparbuch.einzahlen(100.0);  // "Geburtsgutschrift"
  }
}
```

Delegation

- Delegation = Weiterreichen von Nachrichten an Subobjekte

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

Objektarrays

```
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);
  }
}
```

Objektreferenzen als Parameter

```
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);
```

Objektreferenzen als Funktionsrückgabewerte

```
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

- Default-Konstruktor = Konstruktor ohne Parameter

```
class Bruch {
  int zaehler;
  int nenner;

  Bruch() {
    this.zaehler = 0;
    this.nenner = 1;
  }
  ...
}

Bruch bruch1 = new Bruch();
```

Copy-Konstruktor

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

```
class Bruch {
  int zaehler, nenner;

  Bruch(int z, int n) { this.zaehler = z; this.nenner = n; }
  Bruch(Bruch ex) {
    this.zaehler = ex.zaehler;
    this.nenner = ex.nenner;
  } // alle Attribute von this und ex sind wertegleich
}
...
Bruch b1 = new Bruch(3, 4);
Bruch b2 = new Bruch(b1);
```

Gleichheit

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

```
class Bruch{
  int zaehler, nenner;
  Bruch (int z, int n) { ... }
  Bruch(Bruch ex) { ... }

  boolean equals(Bruch anderer) { // Objektgleichheit
    return this.zaehler == anderer.zaehler &&
           this.nenner == anderer.nenner;
} } ...

Bruch b1 = new Bruch(3, 4);
Bruch b2 = new Bruch(b1);
boolean refGleichheit = b1 == b2;          // false
boolean wertGleichheit = b1.equals(b2);    // true
```

main-Funktion

```
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
$
```

Zusammenfassung

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