

Programmierkurs Java

UE 10 – Klassen und Objekte II

Dr.-Ing. Dietrich Boles

- Klassendefinition (Erweiterung)
- Objekte
- Nutzung von Klassen
- Beispiel Gewicht
- Definitionen

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

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

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

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

Anmerkungen:

- Bezeichner: **Klassenname** (neuer Typ!)
- **Klasse = Zusammenfassung von Attributen und Funktionen/Methoden**

- Gültigkeitsbereich: der gesamte Klassenblock
- Lebensdauer: entspricht Lebensdauer des Objektes
- jedes Objekt einer Klasse hat dieselben Attribute
- die Wert der Attribute können jedoch verschieden sein!

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



- spezielle Funktion/Methode zur Initialisierung der Attribute eines Objektes
- Funktionsname = Klassenname
- kein Funktionstyp (auch nicht `void`)
- wird bei der Erzeugung eines Objektes aufgerufen

// Konstruktor

```
Mensch(boolean mann, int groe, double gew) {  
    maennlich = mann;  
    groesse = groe;  
    gewicht = gew;  
}
```



- wie Prozeduren / Funktionen, jedoch ohne `static`
- können auf Attribute und (andere) Methoden zugreifen

```
// Methoden
void wachsen(int cm) {
    groesse += cm;
}

void zunehmen(double kg) {
    gewicht += kg;
}

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

```
double berechneNormalgewicht() {  
    return groesse - 100;  
}
```

```
double berechneIdealgewicht() {  
    if (maennlich) {  
        return berechneNormalgewicht() / 100.0 * 90.0;  
    } else {  
        return berechneNormalgewicht() / 100.0 * 85.0;  
    }  
}
```

```
<objectvar-def> ::= <klassen-bezeichner>  
                    <bezeichner>  
                    { ", " <bezeichner >  
                    ";"
```

Anmerkungen:

- Klassenbezeichner muss Name einer gültigen Klasse sein
- Bezeichner sind Objektvariablen ("Namen für Objekte")
- Objektvariablen sind Referenzvariablen
- Objektvariablen speichern Referenzen auf Objekte

Beispiele:

```
Mensch paul;  
Mensch willi, maria, heike;  
Mensch franz = null; // expl. Initialisierung
```



```
<object-erz> ::= "new" <bezeichner>  
                "(" { <parameter-liste> } ")"
```

Anmerkungen:

- Bezeichner muss Name einer gültigen Klasse sein
- es muss ein "passender" Konstruktor existieren

Semantik:

- Es wird ein Objekt der entsprechenden Klasse "erzeugt" und ggf. ein Konstruktor ausgeführt
- reserviert Speicherplatz für Objektattribute auf dem Heap
- Konstruktor: initialisiert Objektattribute
- liefert Adresse (Referenz) auf den Speicherbereich

Beispiele:

```
new Mensch(true, 181, 72.0)  
new Mensch(false, 169, 55.0)  
new Frau() // Fehler (unbekannte Klasse)  
new Mensch(181) // Fehler (ungültiger Konstruktor)
```

- häufig erfolgt die Objektvariablendefinition und die Objekterzeugung in einer Anweisung
- Zuweisungen möglich
- Typ der Objektvariablen muss gleich Typ des Objektes sein (Ausnahme: Polymorphie!)

Beispiele:

```
Mensch paul = new Mensch(true, 181, 72.5);  
Mensch karl = new Mensch(true, 192, 83.0);  
paul = new Mensch(true, 176, 68.3);  
karl = paul;
```

```
class Tier { . . . }  
Tier mops = new Mensch(false, 168, 54.0); // Fehler!
```

<meth-aufruf> ::= <bezeichner> "." <funktionsaufruf>

Anmerkungen:

- Bezeichner muss gültige Objektvariable sein
- Objektvariable muss auf ein Objekt verweisen
- "passende" Funktion muss in der Klasse des Objektes vorhanden sein

Semantik:

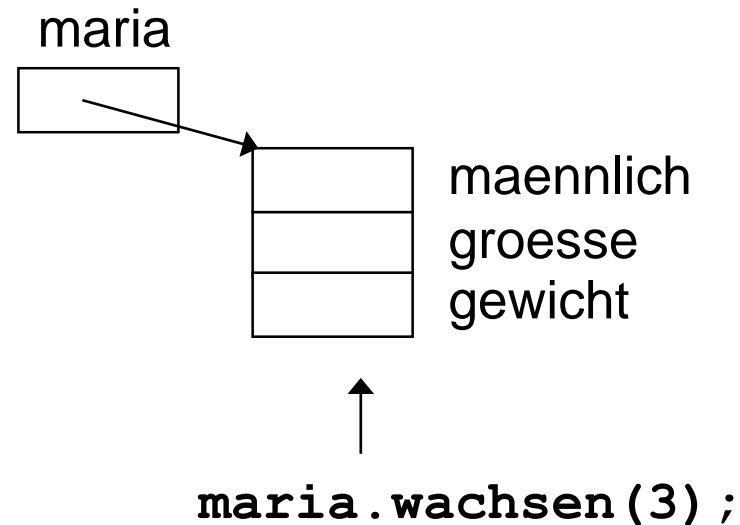
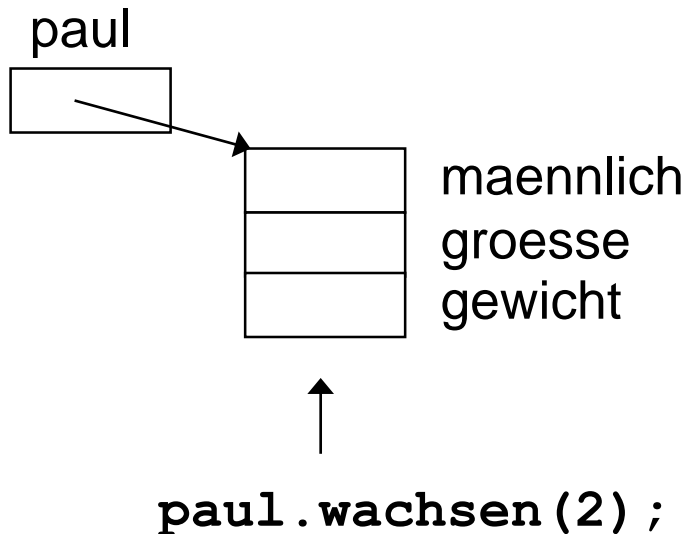
- die Funktion ("für das Objekt") wird aufgerufen
- sie "operiert" auf den Attributen des Objektes

Beispiele:

```
Mensch paul = new Mensch(true, 181, 72.0);  
if (paul.hatOrdentlichesGewicht()) paul.zunehmen(2.3);
```

```
Mensch karl;  
karl.wachsen(2); // Laufzeitfehler (kein Objekt)  
paul.schrumpfen(); // Fehler (fehlende Funktion/Methode)  
paul.wachsen(3.0); // Fehler (fehlende Funktion/Methode)
```

```
Mensch paul = new Mensch(true, 181, 72.0);  
Mensch maria = new Mensch(false, 169, 54.5);  
paul.wachsen(2);  
maria.wachsen(3);  
paul.zunehmen(2.5);  
maria.zunehmen(1.0);
```



Ab jetzt gilt das Prinzip der **Datenkapselung**:

- Von außerhalb einer Klassendefinition darf nur noch auf die Methoden eines Objektes zugegriffen werden, nicht mehr auf die Attribute!
- Ggfls. **Getter/Setter**-Methoden definieren

```
class Mitarbeiter {  
    double gehalt;  
    double getGehalt() { return gehalt; }  
    void setGehalt(double geh) { gehalt = geh; }  
}
```

```
class Firma { ...  
    void streik() {  
        for (Mitarbeiter m : mitarbeiter) {  
            m.setGehalt(m.getGehalt() + 100);  
            m.gehalt += 100; // verboten  
        }  
    }  
}
```

Problemstellung:

Das so genannte Normalgewicht berechnet sich nach der Formel

"Körpergröße (in cm) minus 100". Das Idealgewicht beträgt bei Männern 90% und bei Frauen 85% des Normalgewichts.

Schreiben Sie ein (objektorientiertes) Java-Programm, welches nach Eingabe von Größe, Gewicht und Geschlecht ausgibt, ob ein Mensch zu dick oder zu dünn ist, oder ob er/sie zwischen Ideal- und Normalgewicht liegt.

```
class Mensch {  
  
    // Attribute  
    boolean maennlich;  
    int groesse; // in cm  
    float gewicht; // in kg  
  
    // Konstruktor  
    Mensch(boolean mann, int groe, float gew) {  
        maennlich = mann;  
        groesse = groe;  
        gewicht = gew;  
    }  
}
```

```
// Methoden
void wachsen(int cm) {
    groesse += cm;
}

void zunehmen(float kg) {
    gewicht += kg;
}

boolean hatUebergewicht() {
    return gewicht > berechneNormalgewicht();
}

boolean hatUntergewicht() {
    return gewicht < berechneIdealgewicht();
}

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



```
// Hilfsmethoden
```

```
float berechneNormalgewicht() {  
    return groesse - 100;  
}
```

```
float berechneIdealgewicht() {  
    if (maennlich) {  
        return berechneNormalgewicht() / 100.0f * 90.0f;  
    } else {  
        return berechneNormalgewicht() / 100.0f * 85.0f;  
    }  
}
```

```
public class OOGewicht {
    public static void main(String[] args) {
        boolean isMann = IO.readChar("Maennlich (m/w)?") == 'm';
        int groesse = IO.readInt("Groesse (cm) eingeben: ");
        float gewicht = IO.readFloat("Gewicht (kg) eingeben: ");

        Mensch person = new Mensch(isMann, groesse, gewicht);
        if (person.hatUebergewicht()) {
            IO.println("Alter Fettsack!");
        } else if (person.hatUntergewicht()) {
            IO.println("Na, du Hungerhaken!");
        } else {
            IO.println("Idealer Gewichtsbereich!");
        }
    }
}
```

Klasse:

beschreibt

- Eigenschaften (Attribute)
- Struktur (Subobjekte)
- Verhalten (Methoden)

einer Gruppe von gleichartigen Objekten (→ Datentyp)

Objekte (Instanzen):

- werden durch Klassen beschrieben
- setzen sich demnach zusammen aus
 - Datenelementen (Attribute) → Eigenschaften / Struktur / Zustand
 - den auf den Attributen ausführbaren Operationen → Verhalten
- Objekte einer Klasse haben gleiche Attribute und gleiche Funktionen; sie unterscheiden sich nur in den Werten ihrer Attribute

Attribut (Instanz-Variable):

- Variable, für die in jedem Objekt Speicherplatz reserviert ist
- Menge der Attribute eines Objektes repräsentiert Zustand eines Objektes

Methode:

- realisieren die auf Objekten einer Klasse anwendbaren Operationen

Konstruktor:

- spezielle Methode zur Initialisierung von Objekten

Instantiierung:

- Erzeugung von Objekten

Protokoll einer Klasse:

- Menge der von außen aufrufbaren Methoden einer Klasse

- Klasse: Bauplan für gleichartige Dinge (Objekte)
- Eine Klasse definiert einen neuen Typ
- Methoden: Eine Klasse definiert u. a. Funktionen (Methoden), die für Objekte der Klasse aufgerufen werden können
- Objekte: Von einer Klasse können Objekte erzeugt werden
- Objektvariablen: Der Methodenaufruf erfolgt via der Punktnotation über Objektvariablen, die Referenzen auf Objekte speichern können
- OO-Programm: Aufruf von Methoden von/für Objekte
- Datenkapselung: Zugriff auf Attribute von außerhalb verboten