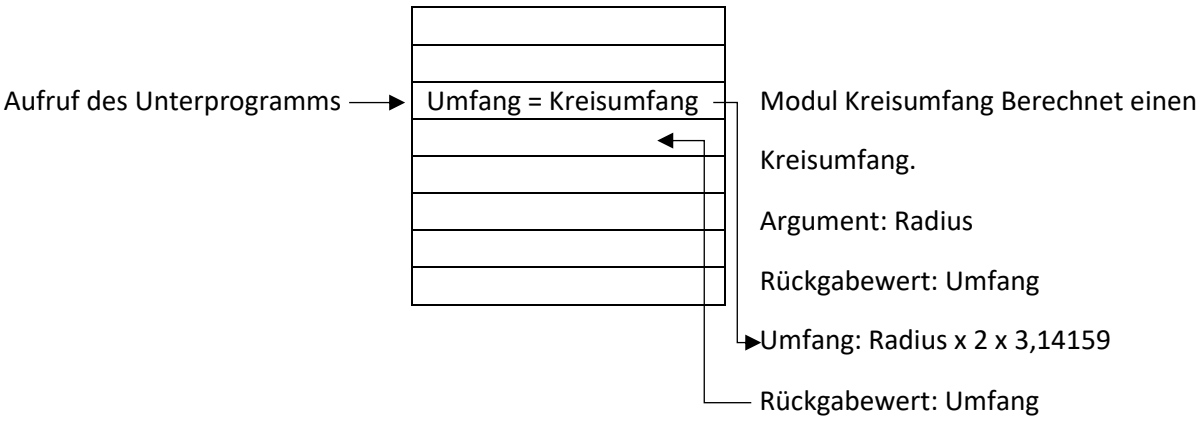
SEOS (Scheringer)

Inhalt

Statische Methoden (Unterprogrammtechnik)



Verschiedene Bezeichnungen für Unterprogramme:

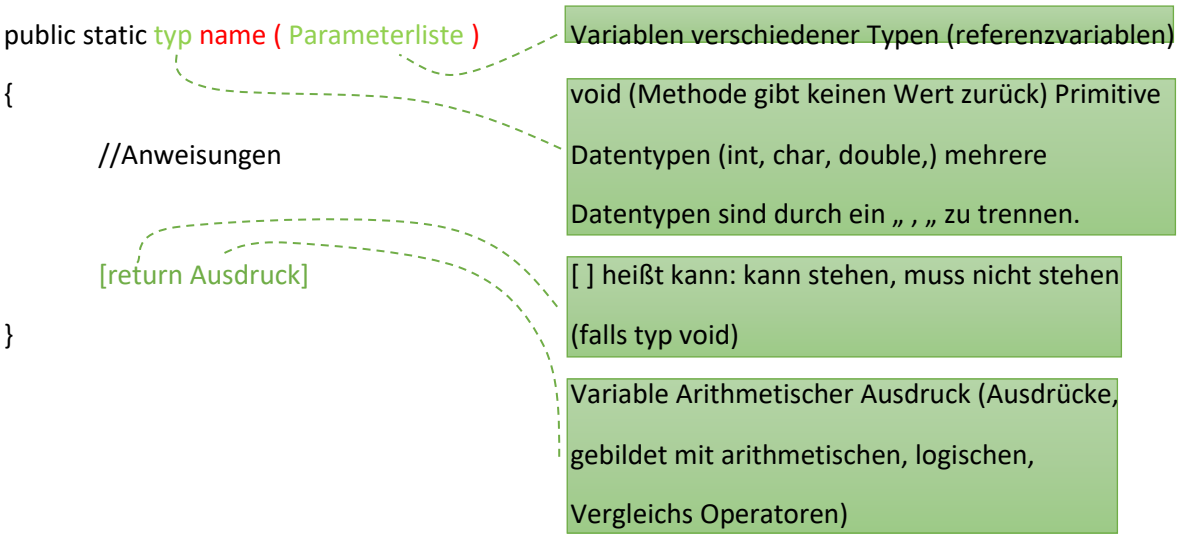
- Methode (Java)
- Prozedur (C, C++, …)
- Funktion ⟶ Ist eine Methode (Prozedur) die einen berechneten wert zurückgibt

Voraussetzungen:

- Müssen in selber Klasse (Class) wie, Hauptprogramm sein.
- Um ein Unterprogramm nutzen zu können, muss derselbe Datentyp wie im Unterprogramm benutzt werden.

```
package java2w1;

public class Javaapp1d {

    public static void main(String[] args) {

        int x[] = {1, 2, 3, 4, 5, 6};

        System.out.println();

        for (int y : x) {
            System.out.print(" " + y);
        }

        modifizieren(x);
        System.out.println();
        for (int y : x) {
            System.out.print(" " + y);
        }
    }

    public static void modifizieren(int x[]) {
        for (int i = 0; i < x.length; i++) {
            x[i] = x[i] * 2;
        }
    }
}
```

Struktur einer statischen Methode



Aufruf einer Methode

[Variable=] name (Prameterliste);

(vom Passenden Typ)

- Mehrere Parameter sind durch Kommata zu trennen
- Variablen
- Konstanten
- Arithmetische Ausdrücke
- Return- Wert einer Methode (kann leer sein)

Parameterlisten müssen zu der Methode passen, heißt Länge und Datentypen müssen mit einander Kompatibel sein.

CALL BY VALUE: Übergabe von Werten von z.B. Variablen

Aufrufen des Programms:

```
double x = 2.56;
double y = -4.81;
double z;
z = mult (x, y);
```

Aufgerufene Methode:

```
public static double multi (double a, double b, double c)
double c;
c = a x b;
```

return;

Call by Reference

Int [ ] arr = {24,9,7,3,10,12}

arr Array arr

Aufrufen

40000 (adresse)

des Progr. Prüfen (arr)

4	9	7	3	10	12

Index 0 1 2 3 4 5

public static void pruefen (int [ ] x)

x[2]=10; //ist identisch mit

//arr[2]=10;

X[4]=x[1]+x[3]

Ternärer Operator

Variable = (Bedingung)? Wert1 : Wert2;

2 Operanden

qs + = z % 10;

Ist dasselbe wie

qs = qs +z % 10;

If (Bedingung)

Variable = Wert1;

Else

Variable = Wert2;

Der ternäre Operator

kann eine if-else-Verzweigung ersetzen und weist meist einer Variablen einen Wert in Abhängigkeit vom Ergebnis einer Bedingungsprüfung zu.

Er ist der einzige Operator, der mit drei Operanden arbeitet, wird oft auch *Bedingungsoperator* genannt und besitzt folgenden allgemeinen Aufbau:

Bedingung ? wert1 : wert2

Die Variablenzuweisung erfolgt dann in der Form

variable = Bedingung ? wert1 : wert2

Bedingung muss immer ein boolscher Ausdruck sein. Er entscheidet über die Wertzuweisung. Ist er true , so wird der Wert nach dem Fragezeichen zugewiesen, ansonsten der Wert nach dem Doppelpunkt.

```
int k = i == 10 ? 12 : 5;
```

Ist i mit 10 initialisiert, so wird k der Wert 12 zugewiesen, im anderen Fall der Wert 5.
wert1 und wert2 müssen Werte repräsentieren, sodass folgendes **nicht** geht, da die Methode print() keinen Rückgabewert besitzt:

```
i == 10 ? System.out.print(12) : System.out.print(5);  // falsch
```

Richtig ist hingegen:

```
System.out.print(i == 10 ? 12 : 5);
```

Im Falle, dass die Variable i mit 10 belegt ist, wird 12 ausgegeben, ansonsten 5.
Hier zeigt sich auch die Stärke des Bedingungsoperators: Er kann Anweisungen sehr kurz fassen, allerdings - wie in solchen Fällen häufig - auf Kosten der Lesbarkeit, besonders wenn die Anweisungen komplexer werden.

```
package Aufgaben;


import java.util.Scanner;


public class Aufgabe6_1 {


    public static void main(String[] args) {
        Scanner eingabe = new Scanner(System.in);


        String w1 = "erfolg";
        String w2 = "gefolg";


        vergleich(w1, w2);
    }


    public static void vergleich(String v1, String v2) {
        boolean flag;


        flag = v1.equals(v2);


        if (flag) {
            System.out.println("Strings sind gleich!");
        } else {
            System.out.println("Strings sind nicht gleich!");
        }
    }
}
```

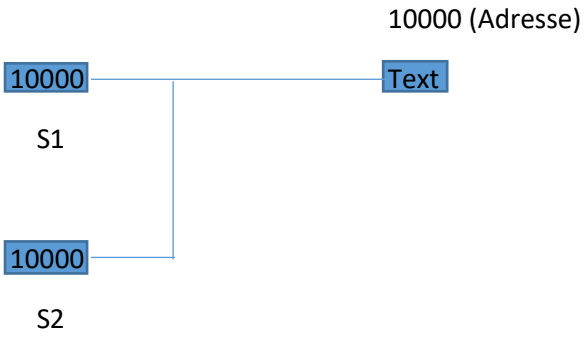
```
package Aufgaben;


import java.util.Scanner;


public class Aufgabe6_11 {


    public static void main(String[] args) {
        Scanner eingabe = new Scanner(System.in);


        String w1 = "erfolg";
        String w2 = "gefolg";


        boolean b = vergleich(w1,w2);


        if (b){
            System.out.println("Strings sind gleich!");
        }else{
            System.out.println("Strings sind nicht gleich!");
        }
    }
    public static boolean vergleich(String v1, String v2) {
       return v1.equals(v2);
    }
}
```

String erzeugen

String s = “Text“ Wird bereits vom Compiler während der Übersetzung des Programms Ausgewertet.

String s = new String(“Text“) Wird erst zur Laufzeit des Programms Ausgeführt (Operators new!)

Compiler Optimiert (String)



String Formatierung zusammengefasst

int x = 10;

double y = 4.217;

char c = ‘A‘

String s = “Text“

System.out.format(“%3d _ _ %10.2f\n%c\n%-20s“, x,y,c,s)

Ausgabe

Objektorientierte Programmierung

1. Statische Methoden
2. String, String Methoden
3. Objektorientierte Softwareentwicklung
    a. Objektorientierte Analyse (OOA)
    b. Objektorientiertes Design (OOD)
    c. Objektorientiertes Programmieren (OOP)

Erste Überlegung: Die reale Welt Beinhaltet Objekte Beispiel: Person, Auto, Haus, Konto, …

Nächste Überlegung: Ein Reales Objekt hat Eigenschaften und Funktionen Bsp.: Objekt Konto

Welche Eigenschaften Und Funktionen Berücksichtigt werden Muss, hängt vom Blick-Winkel auf die Objekte ab.

Eigenschaften: Kontonummer
Name (Inhaber)
Kontostand

Funktion: Konto Anlegen
Geld einzahlen / Geld Auszahlen

Nächste Überlegung: Wie kann man die Struktur eines bestimmten Objekttyps beschreiben?

!!! Mit Hilfe von Klassen !!!

Klassen (Baupläne für Objekte) → Können mittels UML- Diagrammen beschrieben werden (UML = Unified Modeling Language); sofern die UML-Diagramme noch einfach sind, gehören sie zu OOA.

Beispiel:

```
Public class Konto
{
private int kontonr;
private String name;
private double kontostand;
konto( )
{ }
// Akzessoren
Public int getKontonr()
{
Return Kontonr;
}
Public void set Kontonr(int Kontonr)
{
This kontonr = kontonr;
}
```

Member Variablen -> (Attribute) zum Speichern der Eigenschaften Realer Objekte

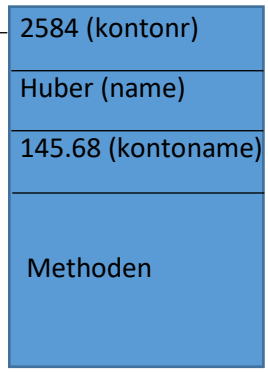
Konstruktor, wird zum erzeugen IT- Technischen Objekten gebraucht

Erläuterungen:

var1 Beginnt ab Adresse 50000

Konto var1 = new Konto(2584,“Huber“,145.68); 50000

Konto var2 = new Konto(2710,“Maier“,-25.17 Var1



Var1 ist eine Referenzvariable die auf Daten auf einer anderen Klasse zeigt, sie speichert nichts!

Var2 zeigt wiederum auf eine andere Speicheradresse und hat z.B. nicht wie var1 Adresse 50000, sondern zeigt auf 60000.

- Zugriff auf Attribute eines Objekts (Attribute sind in unserem Bsp.: kontonr, name, kontonr,)

➔ Falls Attribute den Zugriffsmodifikator haben, nur über Methoden, die in der selber Klasse wie die Attribute erklärt sind.

➔ Hätten die Attribute den Zugriffsmodifikator Public, dann ist ein Zugriff von außen möglich!

- Zugriff auf Methoden einer Klasse mit dem .-Operator, sofern die Methoden den Zugriffsmodifikator Public haben
- Konstruktoren müssen immer den Namen der Class haben. Der Compiler unterscheidet verschiedene Konstruktoren anhand der Listen der formalen Parameter.
- Namensregeln für Getter- und Setter- Methoden einhalten!
- Über Methoden einer Class, Public vorausgesetzt, werden als Schnittstellen (Interface) der Class bezeichnet.

Wozu dienen Konstruktoren?

Sie bewirken das Änderungen an den Attributen vorgenommen werden dürfen.

Prinzipien der Objektorientierung

- IT Technische Abbildung der realen Welt, genauer der Objekte der realen Welt

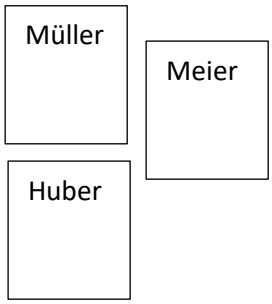
Reale Objekte

Mitarbeiter:

Müller, Meier, Huber

IT- Technisch

Objekte vom Typ Mitarbeiter



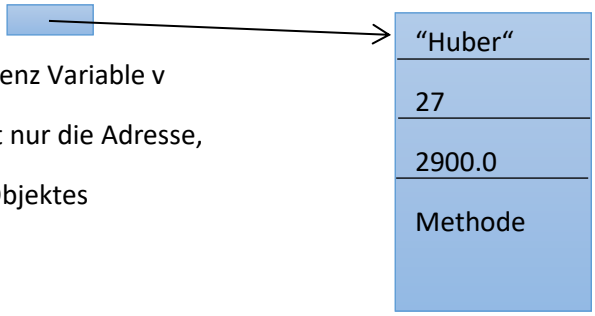
Bauplan für Objekte vom Typ Mitarbeiter

```
public class Mitarbeiter
{
private String name;
private int alter;
private double gehalt;
public Mitarbeiter()
{ }
Public Mitarbeiter(String
Name,int alter,……
        }
this.name = name;
this.alter = alter;
this.gehalt = gehalt;
}
}
```

Erzeugen eines Objekts vom Typ Mitarbeiter

Mitarbeiter v = new Mitarbeiter(“Huber“,27,2900.0);

Im Bsp.



Referenz Variable v

kennt nur die Adresse,

des Objektes

- Wiederverwendbarkeit von Quellcode
- Enge Verknüpfung von Daten und Methoden, mit denen die Daten Bearbeitet werden können (Prinzip der Kapselung)

Erzeugen eines Objekts vom Typ Mitarbeiter

- Ein Objekt, das mit Hilfe einer Klasse erzeugt wurde, bezeichnet man auch als Instanz der Klasse
- Vorgang der Erzeugung einer Instanz wird bezeichnet als Instanziierung

- Referenzarrays

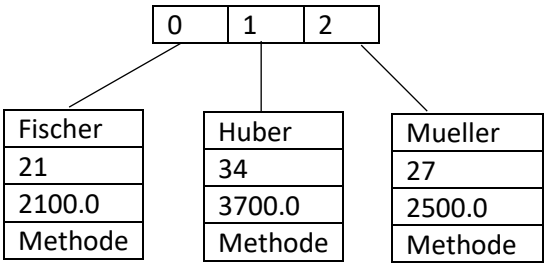
  Mitarbeiter [] m = new Mitarbeiter [3];

  m[0] = new Mitarbeiter(“Fischer“,21,2100.0);

  m[1] = new Mitarbeiter(“Huber“,34,3700.0);

  m[2] = new Mitarbeiter(“Mueller“,27,2500.0)

Arrays dazu: m Index



- Statische Attribute, statische Methoden, statische Konstruktoren

Public class Konto

{

private int knr;

private String inhaber;

private double kstand;

private double zinssatz;

}

Konto k1 = new Konto(1211,……,3.2)

Konto k2 = new Konto(1346,……,3.2)

Konto k3 = new Konto(1541,……,3.2)

Sowas nennt man Klassen Attribut, der Zugriff erfolg über Konto.zinssatz

Situation wenn private static double zinssatz;

Konto k1 = new Konto(1211;“Huber“,1490.0);

Konto k2 = new Konto(1340,“Mueller“,310.0);

Konto k3 = new Konto(1541,“Fischer“,-21.0);

public double got Zins Betrag(int zinstage)

{

double betrag;

betrag = kstand x Konto.zinssatz / 100 x zinssatz / 360

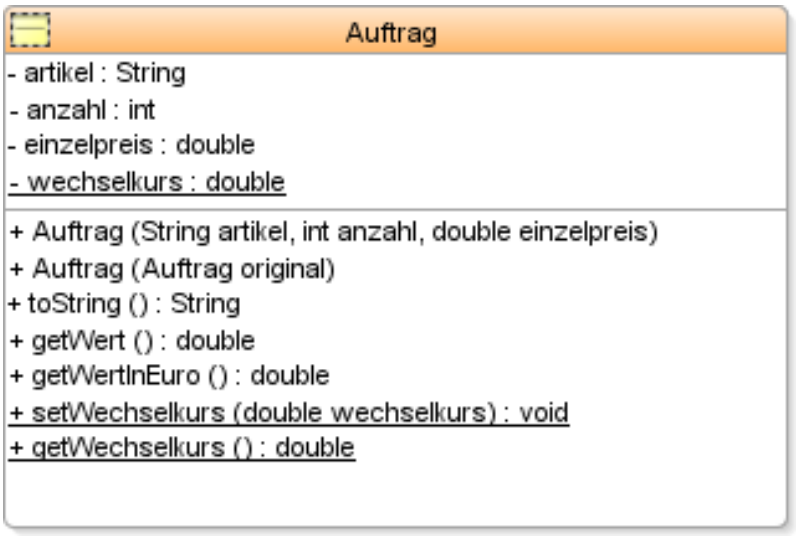
return betrag;

}

Bei dem Zinssatz der bei allen Objekten gleich sein sollen, legt man die Methode mit „static“ fest, um beim ändern des Zinssatzes nicht alle Objekte ändern zu müssen.

Ein Statischer Konstruktor wird (nur einmal) beim Start eines Programms Automatisch ausgeführt

UML Klassen- Diagramm



Erklärung:

- Das - steht für private
- Das + steht für public
- Das unterstrichene bedeutet static

Vererbung

- Begründung Firma XYZ (2.15 CDT)
  es gibt Daten die gespeichert werden sollen
  - Mitarbeiter Name, Vorname, Adresse, Gehalt, Abteilung,
  - Trainer Name, Vorname, Adresse, Honorar, Lister der Themen,
  - Teilnehmer Name, Vorname, Adresse, Lehrgang,

1. Ansatz für eine Klasse

Personen

- Name : String
- Vorname: String
- Adresse: String
- Gehalt: double
- Abteilung: int
- Honorar: double
- Themen: String []
- Lehrgang: String

Methode

2. Ansatz

Mitarbeiter

- Name: String
- Vorname: String
- Adresse: String
- Gehalt: double
- Abteilung: int

Methode

Trainer

- Name: String
- Vorname: String
- Adresse: String
- Honorar: double
- Themen: String

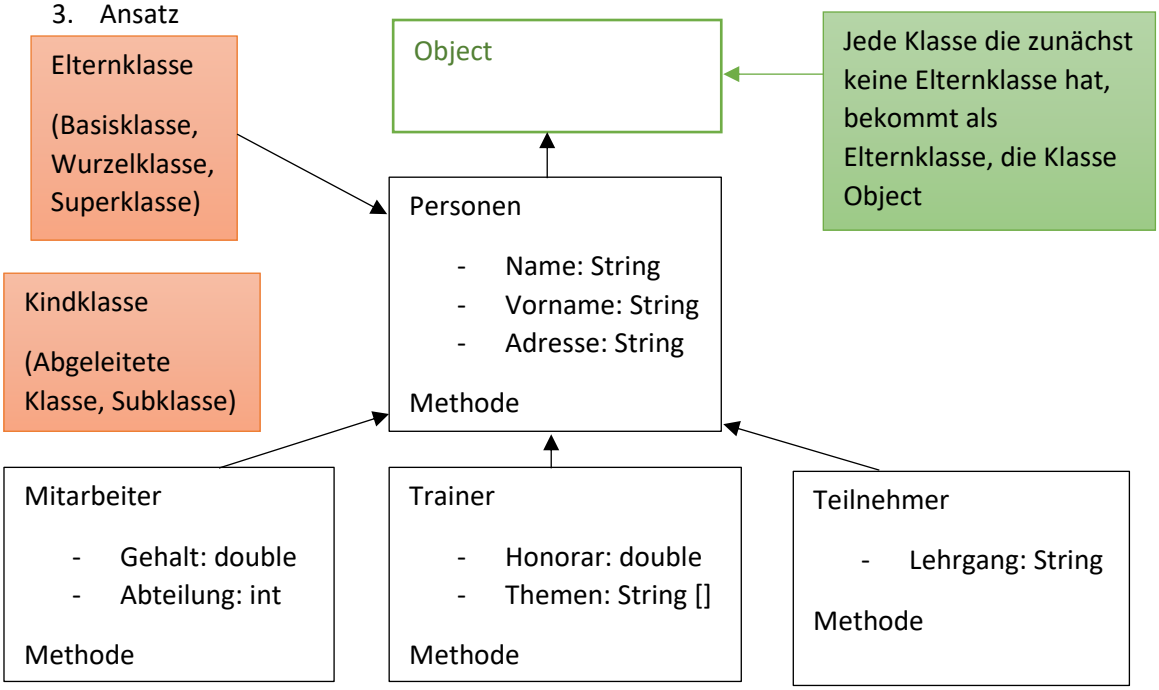
Methode

Teilnehmer

- Name: String
- Vorname: String
- Adresse: String
- Lehrgang: String

Methoden

3. Ansatz



Grundlegende Tatsachen:

- Eine Kind klasse erbt von der Elternklasse alle Attribute und alle Methoden (bis auf …).
- Konstruktoren werden nicht vererbt.
- Statische Attribute werden nicht vererbt.
- Auf Attribute der Elternklasse, die den Zugriffs- Modifikation „private“ haben, kann die Kind klasse nicht direkt zugreifen!
- In Java ist mehrfach- Vererbung nicht erlaubt, das heißt, eine Kind Klasse kann nur von einer Elternklasse erben.
- Vererben über mehrere Ebenen möglich.

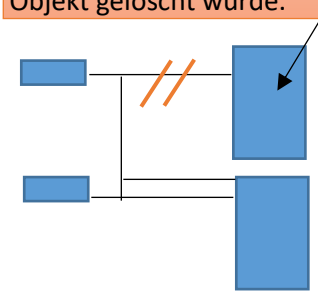
Vererbung, Ausbau der Theorie1

- IS- A- Prinzip, am Beispiel
  - Ein Mitarbeiter ist auch eine Person
  - Ein Trainer ist auch eine Person
  - Ein Teilnehmer ist auch eine Person
- Arbeiten mit Referenzen
  - Mitarbeiter m1 = new Mitarbeiter(…);
  - Mitarbeiter m1 = new Mitarbeiter(…);
- Eine Konsequenz aus dem IS- A- Prinzip
  - Beispie: Personen a;
  - Mitarbeiter b = new Mitarbeiter(…);

    a=b; //ist ein IS- A- Prinzip und funktioniert

    b=a; //ist kein IS- A- Prinzip und funktioniert nicht

Wird durch garbage- Kollektor entsorgt, da der Verwais auf das Objekt gelöscht wurde.



Personen z;

Mitarbeiter.IT x = new Mitarbeiter(…);

Mitarebiter y = new Mitarbeiter(…);

Wenn T2 eine Kindklasse (Enkel-,Urekel- Klasse, …) Der Klasse T1 ist, dann ist möglich:

z=x	Funktioniert	T1 p;
y=x	Funktioniert	T2 q;
y=z	Funktioniert nicht	.
x=y	Funktioniert nicht	.
x=z	Funktioniert nicht	p = q; //IS- A- Prinzip!

- Begriffe
  - Compiletime ⟶ Zeit zu der der Compiler ein Quellprogramm übersetzt
  - Runtime ⟶ Zeit, zu der die JVM ein Programm (Bytecode) ausführt

Spezialisierung:

Die Spezialisierung gibt im UML- Diagramm die Vererbung in Richtung unten an. Sie wird in Richtung unten breiter bzw. Spezieller.

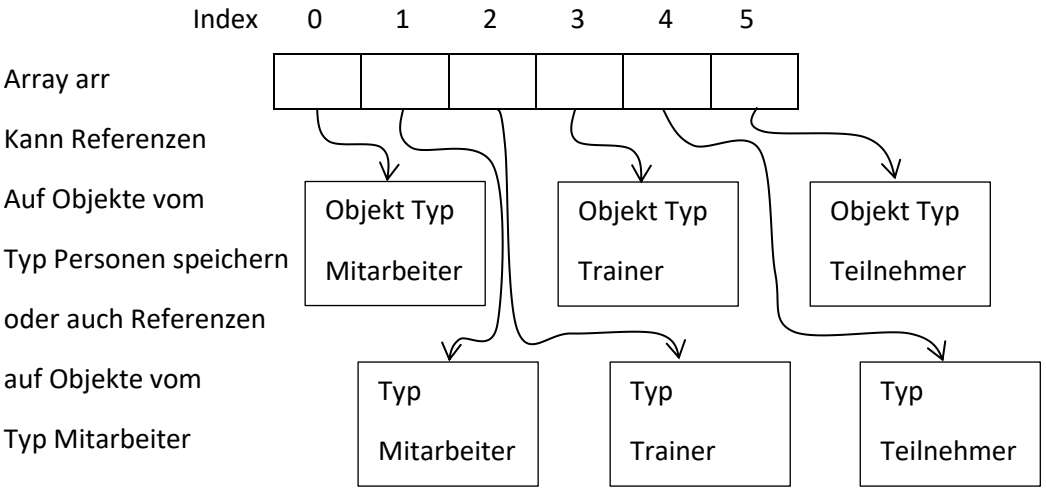
Generalisierung:

Die Generalisierung gibt im UML- Diagramm die Vererbung in Richtung oben an. Sie wird in Richtung oben schmaler, also dünner.

String- Vergleich Merke:

„if(s==t)“ vergleicht nur die Speicheradresse der String Variablen „s“ und „t“. Eine Prüfung auf Gleichheit wird mit „if(s.equals(t))“ erreicht!

Vererbung, Ausbau der Theorie2



Begriffe Upcast, Downcast

Klasse T1 ist Elternklasse von Klasse T2

T1 p;

T2 q;

p=q; //Upcast

q=p; //geht nicht

(T2) p; //Down Cast, geht nur, wenn zur Runtime in p wirklich die Referenz auf ein Objekt
//vom Typ T2 steht!

Vererbung, Ausbau der Theorie3

- Der Compiler baut in Konstruktoren sofern man nicht selbst ein super(…); Programmiert hat, ein super(…); ein.

Aufruf des Standard Konstruktors der Elternklasse!

- Wie kann verhindert werden, dass Objekte vom Typ Personen angelegt werden können
- Überladen und Überschreiben von Methoden
    - Kopf einer Methode :
        - Return- Typ <Name der Methode>(Liste der Parametertypen
        - Beispiel int rechnen (int, int)
        - Double rechnen(double, double)
- String system.out.println(String);
- String system.out.println(int);

- Überschreiben
    - Signatur einer Methode
        - Signatur besteht aus <Name der Methode>(Liste der Parameter)
        - In einer Liste darf es nie zwei Methoden mit derselben Signatur geben!

Beispiel: Klasse Objects toString() @Override

für Override Klasse Personen toString() to String ← selbst geschrieben

Klasse Mitarbeiter ist verdeckt

Der instanceof-Operator

Mit dem `instanceof`-Operator kann zur Laufzeit geprüft werden, ob ein von einem Verweis referenziertes Objekt zuweisungskompatibel zu einer Klasse ist, die im zweiten Operanden von `instanceof` angegeben wird. Der zweite Operand darf kein Verweis sein. `instanceof` hat die Syntax

```
<Verweis oder Verweistyp> instanceof <Verweistyp>
```

`instanceof` liefert `true`, wenn

`instanceof` liefert `false`, wenn

Mit `instanceof` kann zum Beispiel zunächst festgestellt werden, ob zwei Objekte zuweisungskompatibel sind, bevor die eigentliche Zuweisung erfolgt.
Das folgende Beispiel zeigt den Anwendungsfall einer Zuweisung:

```
Point p;

// Zuweisung an p;

if (p instanceof Square)
   Square s = (Square)p;
```

Durch die `if`-Anweisung wird `p` nur dann an `r` zugewiesen, wenn der Wert von `p` zur Laufzeit ein Exemplar von `Square` ist. Falls `p` auf ein Exemplar einer beliebigen Unterklasse von `Square` verweist, ist das Ergebnis von `instanceof` natürlich auch `true`.

In der Standardbibliothek wird `instanceof` fast durchgänig in den Implementierungen der `equals(Object)`-Methode eingesetzt, um vor dem Vergleich der Datenelemente zu prüfen, ob das übergebene Objekt überhaupt zur gleichen Klasse gehört. Hat diese Prüfung Erfolg, kann das zu prüfende Objekt gefahrlos mit einem Cast umgewandelt und die Datenelemente verglichen werden.

```
public boolean equals(Object obj) {
  if (obj instanceof Point) {
    Point pt = (Point)obj;
    return (x == pt.x) && (y == pt.y);
  }
  ...
}
```

Copy-Konstruktor

Dazu ein Beispiel: Die Klasse `Player` bekommt einen Konstruktor, der einen anderen Spieler als Parameter entgegennimmt. Auf diese Weise lässt sich ein schon initialisierter Spieler als Vorlage für die Attributwerte nutzen. Alle Eigenschaften des existierenden Spielers können so auf den neuen Spieler übertragen werden. Die Implementierung kann so aussehen:

```
public Personen(Personen p) {

   this.name = p.name;

   this.vorname = p.vorname;

   this.adresse = p.adresse;

}
```