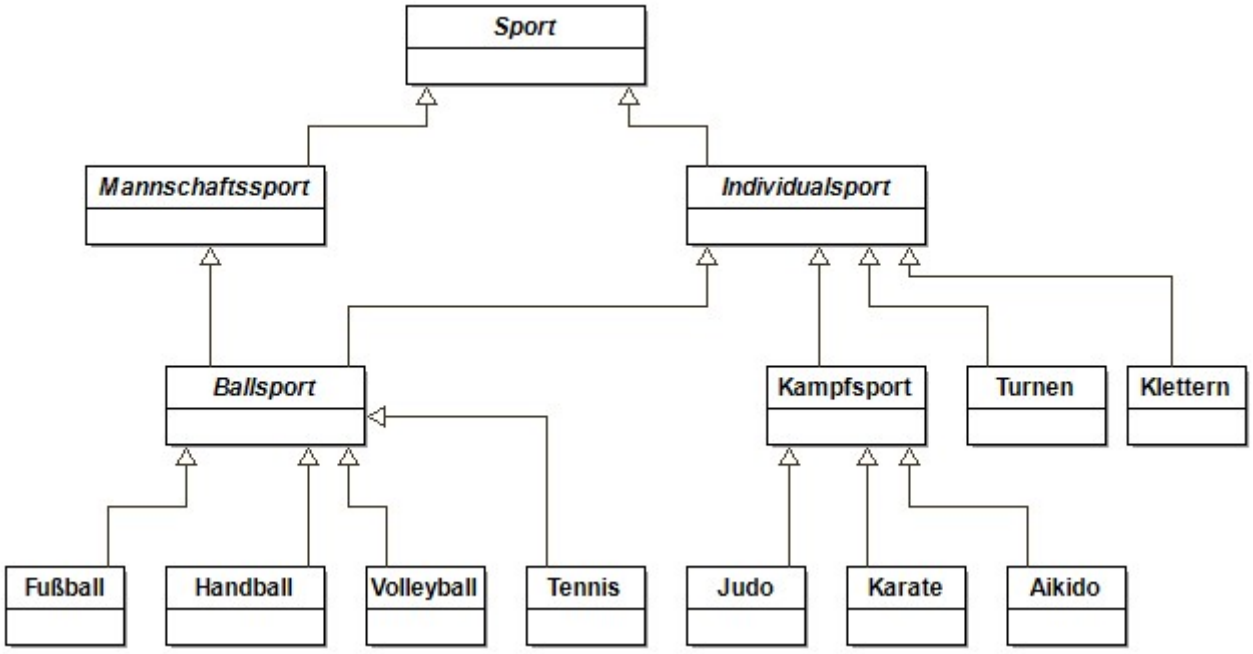
gds2 Gottlieb-Daimler-Schule 2
Technisches Schulzentrum Sindelfingen
mit Abteilung Akademie für Datenverarbeitung

1 Generalisierung und Spezialisierung von Klassen

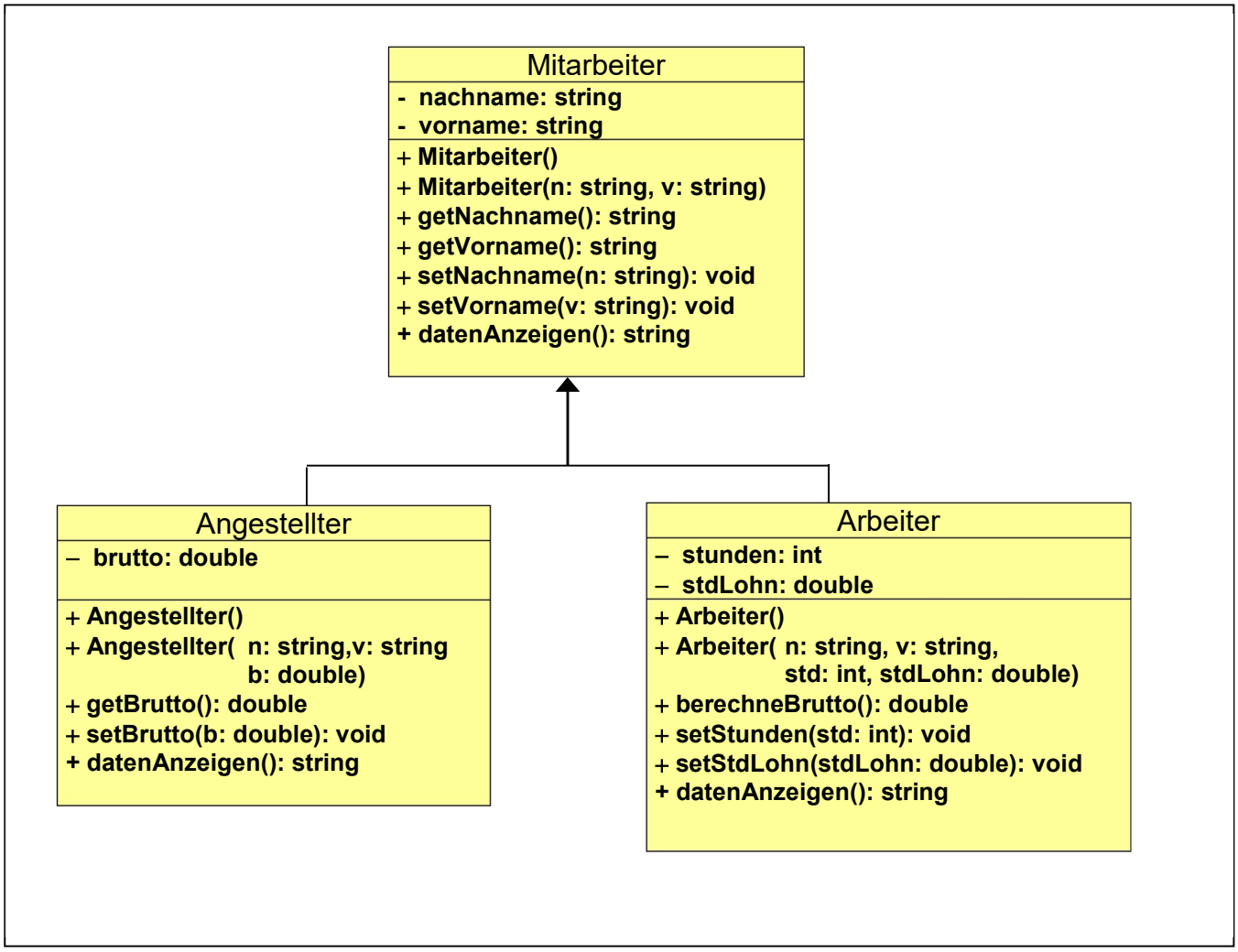
Ziel: Konzept der Generalisierung und Spezialisierung kennenlernen

1.1. Für eine Sofwareanwendung im Sportumfeld soll eine Klassenhirachie konzipiert werden. Folgende Sportarten sind zu berücksichtigen:
Handball, Judo, Fußball, Tennis, Karate, Aikido, Turnen, Klettern, Volleyball



2 Implementierung einer Vererbungshierarchie

Problemstellung: Bei der Gehaltsabrechnungssoftware muss zwischen Arbeitern und Angestellten unterschieden werden. Die gemeinsamen Attribute und Methoden für beide Personengruppen werden in der Basisklasse „Mitarbeiter“ zusammengefasst. Die spezialisierten Klassen werden dann von dieser Basisklasse „abgeleitet“:



2.1. Implementierung der Klassen; Vorgaben:

- Die Standardkonstruktoren setzen die Attribute auf einen Anfangswert: Strings werden mit “- keine Angabe -“ vorbelegt und Zahlen mit 0.
  Die Standardkonstruktoren nutzen die überladenen Konstruktoren (Verkettung)

- Die „überladenen“ Konstruktoren der abgeleiteten Klassen verwenden den „überladenen“ Konstruktor der Basisklasse mit.

- Erzeugen Sie zum Testen im Hauptprogramm jeweils zwei Objekte vom Typ Angestellter und Arbeiter, wobei Sie jeweils ein Objekt mit Hilfe des „überladenen“ Konstruktors initialisieren und das jeweils andere Objekt mit Hilfe der Set-Methoden initialisieren.

- Zeigen Sie dann die Daten der einzelnen Mitarbeiter am Bildschirm an. Nutzen Sie

dazu die Methode `datenAnzeigen()`, die den entsprechenden String erzeugt. Bsp.:

```
Angestellter: Markus Müller verdient Brutto 2500 € pro Monat
```

2.2. Aufrufreihenfolge der Konstruktoren untersuchen

Nutzen Sie den Debug-Modus von Eclipse, um die Aufrufreihenfolge der Konstruktoren zu verfolgen. Setzen dazu jeweils auf die erste Anweisung jedes Konstruktors einen Haltepunkt (Breakpoint). Starten Sie dann das Programm im Debug-Modus.

```
 7 /**
 8  * @author stk
 9  *
10  *         Kurzbeschreibung: Speichert die Daten eines Mitarbeiters
11  */
12 public abstract class Mitarbeiter
13 {
14     private String nachname;
15     private String vorname;
16
17     public Mitarbeiter()
18     {
19         // this.nachname = "- leer -";
20         // this.vorname = "- leer -";
21         // Besser: Verkettung von Konstruktoren
22         this("- leer -", "- leer -");
23     }
24
25     public Mitarbeiter(String n, String v)
26     {
27         this.nachname = n;
28         this.vorname = v;
29     }
30
31     public String getNachname()
35
36     public void setNachname(String n)
40
41     public String getVorname()
45
46     public void setVorname(String v)
50
51     // Die abstrakte Methode macht die Klasse zu einer abstrakten Klasse
52     public abstract String datenAnzeigen();
53 //  {
54 //      return String.format("Mitarbeiter: %s %s\n", vorname, nachname);
55 //  }
56 }
```

```java
public class Angestellter extends Mitarbeiter
{
    private double brutto;

    public Angestellter()
    {
        // Besser: Verkettung von Konstruktoren
        this("- leer -", "- leer -", 0);
        //this.brutto = 0;
    }

    public Angestellter(String n, String v, double b)
    {
        super(n, v); // Aufruf des Konstruktors der Ober(Super)klasse
        this.brutto = b;
    }

    public double getBrutto()

    public void setBrutto(double b)

    public String datenAnzeigen()
    {
        return "Der Angestellte " + this.getVorname() + ", " + this.getNachname() +
        String.format(" verdient Brutto %7.2f € pro Monat\n", this.brutto);
    }
}
```

```java
public class Arbeiter extends Mitarbeiter
{
    private int stunden;
    private double stdLohn;

    public Arbeiter()
    {
        // Besser: Verkettung von Konstruktoren
        this("- leer -", "- leer -", 0, 0.0);
        //this.stunden = 0;
        //this.stdLohn = 0.0;
    }

    public Arbeiter(String n, String v, int std, double stdLohn)
    {
        super(n,v);  // Verkettung mit Konstruktor der Basisklasse
        this.stunden = std;
        this.stdLohn = stdLohn;
    }

    public double berechneBrutto()
    {
        return this.stunden * this.stdLohn;
    }

    public void setStunden(int std)

    public void setStdLohn(double stdLohn)

    public String datenAnzeigen()
    {
        return "Der Arbeiter: " + this.getVorname() + ", " + this.getNachname()
          + String.format(" verdient Brutto %6.2f € pro Monat\n", this.berechneBrutto());
    }
}
```

2.3. Verwaltung von Objekten über Basisklassenreferenzen und „Polymorpher“ Aufruf der Methode `datenAnzeigen()`

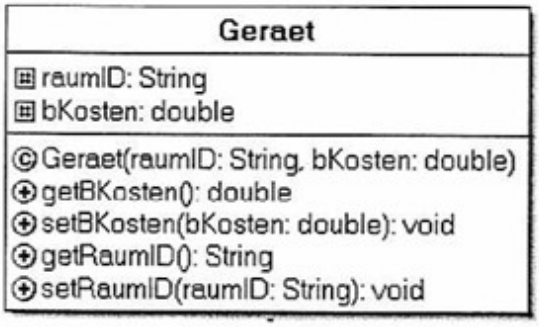
- Definieren Sie in `main()` zusätzlich ein Array der Klasse `Mitarbeiter` für 4 Elemente.
- Weisen Sie den Elementen des Arrays ihre zwei Angestellten und zwei Arbeiter zu.
- Lassen Sie die Daten der Array-Elemente in einer for-Schleife am Bildschirm durch Aufruf der Methode `datenAnzeigen()` ausgeben.

  → Welche Variante der Methode `datenAnzeigen()` wird jetzt jeweils aufgerufen?

```
public class PersonalStartUI
{
    public static void main(String[] args)
    {
        Angestellter an1 = new Angestellter("Fröhlich", "Markus", 2500.0);
        Angestellter an2 = new Angestellter();

        Arbeiter ar1 = new Arbeiter("Sauer", "Stefan", 140, 25.0);
        Arbeiter ar2 = new Arbeiter();

        System.out.println(an1.datenAnzeigen());
        System.out.println(an2.datenAnzeigen());

        an2.setNachname("Frisch");
        an2.setVorname("Michael");
        an2.setBrutto(3000.0);
        System.out.println(an2.datenAnzeigen());

        System.out.println(ar1.datenAnzeigen());
        System.out.println(ar2.datenAnzeigen());

        ar2.setNachname("Schlecht");
        ar2.setVorname("Sebastian");
        ar2.setStdLohn(30.0);
        ar2.setStunden(130);
        System.out.println(ar2.datenAnzeigen());

        Mitarbeiter[] liste = new Mitarbeiter[4];
        // Das Array kann Angestellte und Arbeiter aufnehmen
        liste[0] = an1;
        liste[1] = an2;
        liste[2] = ar1;
        liste[3] = ar2;

        for (int i = 0; i < liste.length; i++)
        {
            // Je nach Objekt-Typ wird die passende Methode aufgerufen
            // Dies ist möglich durch spätes binden
            System.out.println(liste[i].datenAnzeigen());
        }
    }
}
```

2.4. Geräteverwaltung

Für die Geräteverwaltung der Fa. Machheim soll ein Programm entwickelt werden. Zunächst werden nur Rechner verwaltet, später sollen auch andere Geräte (z. B. Kopierer, usw.) integriert werden.



Codieren Sie zunächst die Klasse *Geraet* (vgl. UML-Klassendiagramm) mit der an der Schule eingeführten Programmiersprache.

Eigenschaft	Datentyp	Beschreibung
raumID	String	Raum-Nummer des Gerätes
bKosten	Double	Beschaffungskosten in Euro

Implementieren Sie eine Methode *aktuellerGeraeteWert()*, die den aktuellen Wert eines Gerätes in Abhängigkeit von dem Alter in Jahren ermittelt.
Eingabeparameter der Methode ist das Alter in Jahren und der Rückgabewert der aktuelle Geldwert.

Von folgendem Wertverlust wird ausgegangen:

Wert eines Rechner in Abhängigkeit vom Alter				
1. Jahr	2. Jahr	3. Jahr	4. Jahr	danach
100 %	50 %	25 %	12,5 %	10 %

Ergänzung zum Testen:

- Legen Sie im Hauptprogramm zwei Geräte-Objekte an und initialisieren Sie diese.
- Erzeugen Sie eine Bildschirmausgabe in der Form:

```
Gerät g1 von Raum B243 hat 500 Euro gekostet. Aktueller Wert = 50 Euro
Gerät g2 von Raum D114 hat 450 Euro gekostet. Aktueller Wert = 225 Euro
```

Für die Verwaltung der Rechner wird die Klasse *Geraet* erweitert und eine neue Klasse *Rechner* entworfen.
In der Klasse *Rechner* werden die zusätzlichen Eigenschaften, wie der Rechnername und der Domänenname sowie die entsprechenden Setter- und Getter-Methoden implementiert.
Erweitern Sie das UML-Klassendiagramm entsprechend Aufgabe 2.1.

Eigenschaft	Datentyp	Beschreibung
rName	String	Rechnername (bspw. *pizza*)
rDomain	String	Domänenname (bspw. *machheim.com*)

```
 * @author stk
public class Geraet
{
    protected String raumID;
    protected double bKosten;
    protected LocalDate bDatum;

    public Geraet(String raumID, double bKosten, LocalDate bDatum)
    {
        this.raumID = raumID;
        this.bKosten = bKosten;
        this.bDatum = bDatum;
    }

    public void setRaumID(String raumID)

    public String getRaumID()

    public void setBKosten(double bKosten)

    public double getBKosten()

    public LocalDate getBDatum()

    public double aktuellerGeraeteWert()
    {
        LocalDate heute = LocalDate.now();
        int iAlterJ;
        double[] faktor = { 1.0, 0.5, 0.25, 0.125, 0.1 }; // Lookup Table

        double wert = 0;

        iAlterJ = heute.getYear() - bDatum.getYear();
        if (heute.getMonthValue() < bDatum.getMonthValue()
                || (bDatum.getMonthValue() == heute.getMonthValue()
                && heute.getDayOfMonth() < bDatum.getDayOfMonth()))
            iAlterJ--;

        if (iAlterJ>=4)
        {
            wert = bKosten * faktor[4];
        }
        else
        {
            wert = bKosten * faktor[iAlterJ];
        }

        return wert;
    }
```

Ergänzung um eine abgeleitete Klasse:

- Implementieren Sie auch die abgeleitete Klasse „Rechner“
- Erzeugen Sie im Hauptprogramm zum Testen zusätzlich auch noch zwei Objekte vom Typ Rechner mit entsprechender Bildschirmausgabe.

```
 * @author stk
public class Rechner extends Geraet
{
    private String rName;
    private String rDomain;

    public Rechner(String raumID, double bKosten, LocalDate bDatu
                   String rName, String rDomain)
    {
        super(raumID, bKosten, bDatum);
        this.rName = rName;
        this.rDomain = rDomain;
    }

    public void setRName(String rName)

    public String getRName()

    public void setRDomain(String rDomain)

    public String getRDomain()

    public double aktuellerGeraeteWert()
    {
        LocalDate heute = LocalDate.now();
        int iAlterJ;
        double[] faktor = { 1.0, 0.5, 0.25 };

        double wert = 0;

        iAlterJ = heute.getYear() - bDatum.getYear();
     if (heute.getMonthValue() < bDatum.getMonthValue()
             || (bDatum.getMonthValue() == heute.getMonthValue()
             && heute.getDayOfMonth() < bDatum.getDayOfMonth()))
        iAlterJ--;

        if (iAlterJ >= 3)
        {
            wert = 0;
        }
        else
        {
            wert = bKosten * faktor[iAlterJ];
        }
        return wert;
    }
}
```

```java
 * @author stk
public class GeraeteInventarUI
{
    /**
     * @param args
     * Kurzbeschreibung: Start-UI-Klasse für die Inventarberechnung
     */
    public static void main(String[] args)
    {
        Geraet g1 = new Geraet("B243", 500, LocalDate.of(2016, 2, 18));
        Geraet g2 = new Geraet("D114", 450, LocalDate.of(2019, 3, 7));

        System.out.printf("Gerät g1 von Raum %s hat %7.2f Euro gekostet. "
                          + "Aktueller Wert = %7.2f  Euro\n",
                          g1.getRaumID(), g1.getBKosten(), g1.aktuellerGeraeteWert());

        System.out.printf("Gerät g2 von Raum %s hat %7.2f  Euro gekostet. "
                          + "Aktueller Wert = %7.2f  Euro\n",
                          g2.getRaumID(), g2.getBKosten(), g2.aktuellerGeraeteWert());

        Rechner r1 = new Rechner("B243", 500, LocalDate.of(2016, 2, 18),
                                 "B243_PC01", "gds2.de\n");
        Rechner r2 = new Rechner("B256", 650, LocalDate.of(2019, 4, 17),
                                 "B243_PC01", "gds2.de\n");

        System.out.printf("Rechner r1 von Raum %s hat %7.2f Euro gekostet. "
                          + "Aktueller Wert = %7.2f  Euro\n",
                          r1.getRaumID(), r1.getBKosten(), r1.aktuellerGeraeteWert());

        System.out.printf("Rechner r2 von Raum %s hat %7.2f Euro gekostet. "
                          + "Aktueller Wert = %7.2f  Euro\n",
                          r2.getRaumID(), r2.getBKosten(), r2.aktuellerGeraeteWert());

        Geraet[] liste = new Geraet[4];
        double gesamtwert = 0;

        liste[0] = g1;
        liste[1] = g2;
        liste[2] = r1;
        liste[3] = r2;

        for (int i = 0; i < liste.length; i++)
        {
            // Je nach Objekt-Typ wird die richtige Methode aufgerufen
            gesamtwert = gesamtwert + liste[i].aktuellerGeraeteWert();
        }

        System.out.printf("Gesamtwert des Inventars = %7.2f", gesamtwert);
    }
}
```

Zusatzaufgaben:

- Ergänzen Sie die Basisklasse um ein zusätzliches Attribut

```
# bDatum: LocalDate // Beschaffungsdatum
```

Ergänzen Sie den Konstruktor so, dass beim Erzeugen eines Objektes das Beschaffungsdatum eingetragen wird.

- Erzeugen Sie im Hauptprogramm zum Testen zusätzlich auch noch zwei Objekte vom Typ Rechner mit entsprechender Bildschirmausgabe.
- Definieren Sie im Hauptprogramm ein Array zur Speicherung aller Geräte (inkl. Rechner), so dass Sie den Gesamtwert des Inventars in einer Schleife ermitteln können.

  Achtung: Für die Ermittlung des aktuellen Gerätewertes muss jetzt das Alter nicht mehr an die Methode *aktuellerGeraeteWert()* übergeben werden. Das Alter soll in dieser Methode aus dem aktuellen Datum und dem Beschaffungsdatum ermittelt werden.
- Ändern Sie die Wertermittlung für einen „Rechner“ so ab, dass er im Gegensatz zu anderen Geräten bereits ab dem 4. Jahr mit 0 € gebucht wird.