Erstes Viertel der Vorlesung

Software-Engineering

Was Modellieren

Wie Programm schreiben
kompilieren
korrigieren
interpretieren

Warum verifizieren: Zustandsverfolgung, Verifikation, Induktionsbeweis
oder systematischer Performanztest

Grundbegriffe der Informatik

Vererbung,

Syntax, Produktionen,

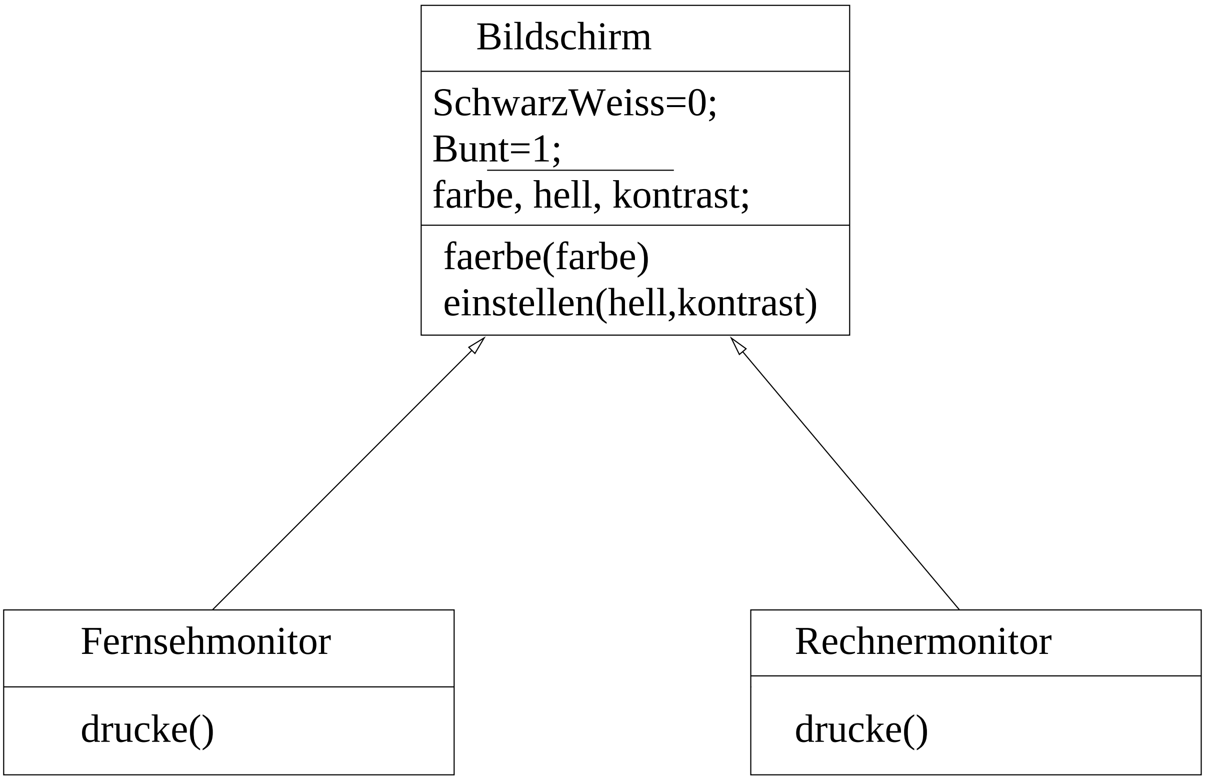
Wert-/ Parameter- -zuweisung/-übergabe

JAVA

Klasse, Behälterklasse, Objekt

Variable, Methode, Konstruktor

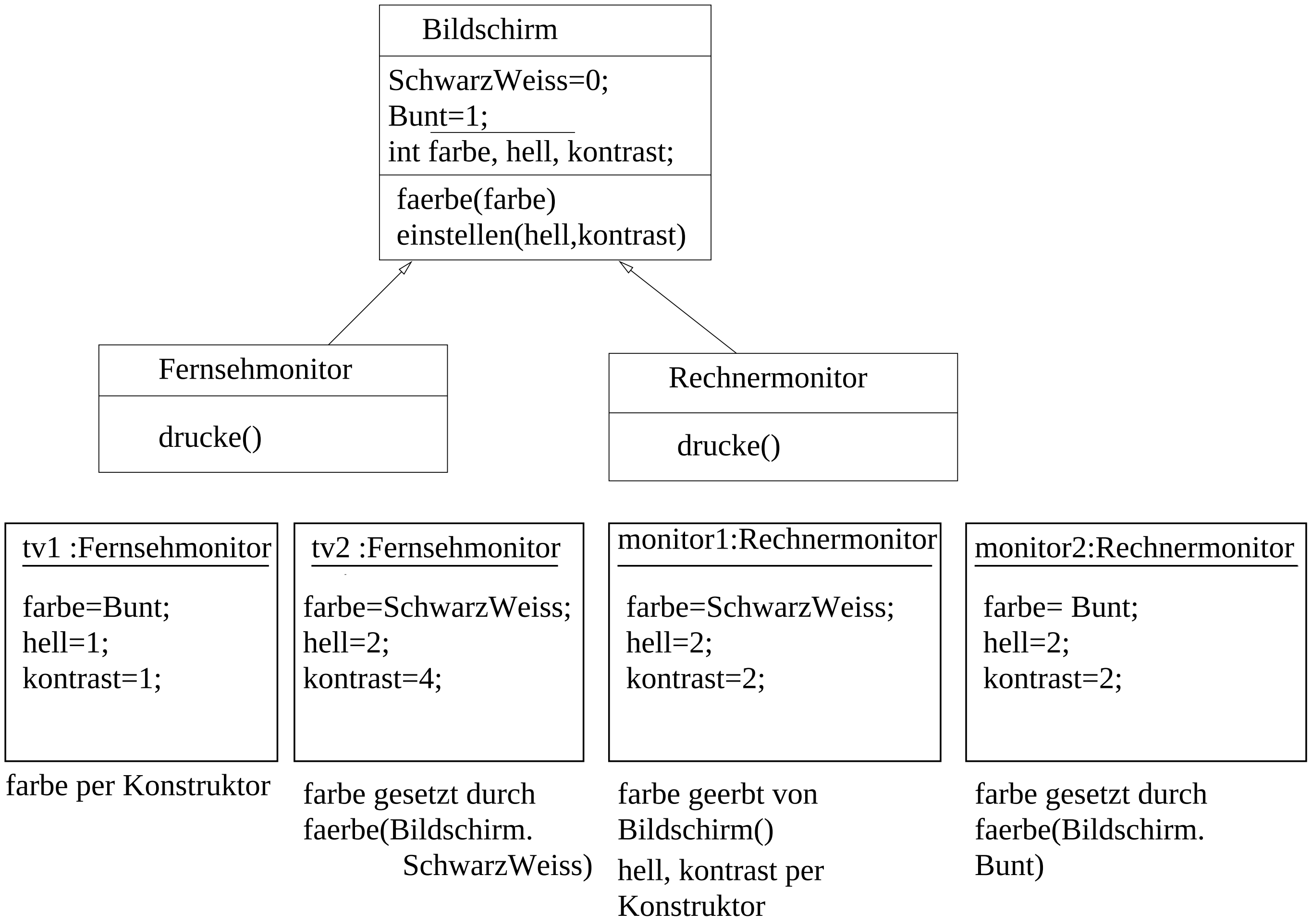
Ein ganz normales Beispiel



```
class Bildschirm {
    static final int SchwarzWeiss=0, Bunt=1;
    int farbe, hell, kontrast;

    public Bildschirm () {
        farbe=SchwarzWeiss;
        hell=3;
        kontrast=3; }

    public void faerbe (int _farbe) {
        farbe=_farbe;

    }

    public void einstellen (int _hell, int _kontrast) {
        hell=_hell;
        kontrast=_kontrast;
    }

}
```

```
class Fernsehmonitor extends Bildschirm {
    public Fernsehmonitor () {
        farbe=Bunt;
        hell=1;
        kontrast=1;
    }

public void drucke () {
        System.out.println ("TV−Farbe: "+farbe);
        System.out.println ("TV−Helligkeit: "+hell);
        System.out.println ("TV−Kontrast: "+kontrast);
    }
}

class Rechnermonitor extends Bildschirm {
    public Rechnermonitor () {
        hell=2;
        kontrast=2;
    }

    public void drucke () {
        System.out.println ("Monitor−Farbe: "+farbe);
        System.out.println ("Monitor−Helligkeit: "+hell);
        System.out.println ("Monitor−Kontrast: "+kontrast);
    }
}
```

```
class FernsehClass {

    public static void main (String[] argv) {
        Fernsehmonitor tv1=new Fernsehmonitor ();
        tv1.drucke ();
        Fernsehmonitor tv2=new Fernsehmonitor ();
        tv2.faerbe (Bildschirm.SchwarzWeiss);
        tv2.einstellen (2,4);
        tv2.drucke ();
        Rechnermonitor monitor1=new Rechnermonitor ();
        monitor1.drucke ();
        Rechnermonitor monitor2=new Rechnermonitor ();
        monitor2.faerbe (Bildschirm.Bunt);
        monitor2.drucke ();
    }
}
```


Bildschirm
SchwarzWeiss=0;
Bunt=1;
int farbe, hell, kontrast;
faerbe(farbe)
einstellen(hell,kontrast)
Fernsehmonitor
drucke()
Rechnermonitor
drucke()
tv1 :Fernsehmonitor
farbe=Bunt;
hell=1;
kontrast=1;
tv2 :Fernsehmonitor
farbe=SchwarzWeiss;
hell=2;
kontrast=4;
monitor1:Rechnermonitor
farbe=SchwarzWeiss;
hell=2;
kontrast=2;
monitor2:Rechnermonitor
farbe= Bunt;
hell=2;
kontrast=2;
farbe per Konstruktor
farbe gesetzt durch faerbe(Bildschirm. SchwarzWeiss)
farbe geerbt von Bildschirm()
hell, kontrast per Konstruktor
farbe gesetzt durch faerbe(Bildschirm. Bunt)

tv1 mit den Werten des Konstruktors:

TV-Farbe: 1

TV-Helligkeit: 1

TV-Kontrast: 1

tv2 mit SchwarzWeiss und eingestellter Helligkeit, Kontrast:

TV-Farbe: 0

TV-Helligkeit: 2

TV-Kontrast: 4

monitor1 mit den Werten des Konstruktors:

Monitor-Farbe: 0

Monitor-Helligkeit: 2

Monitor-Kontrast: 2

monitor2 mit Bunt:

Monitor-Farbe: 1

Monitor-Helligkeit: 2

Monitor-Kontrast: 2

Schleifen

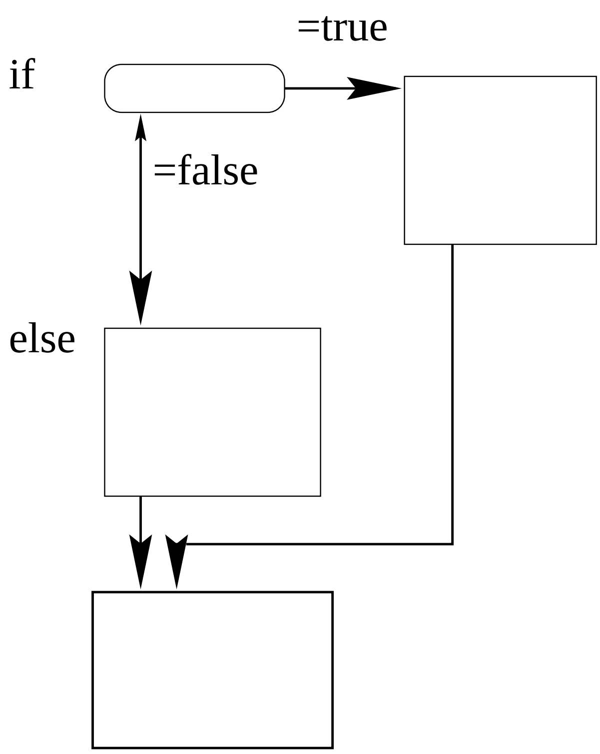
- Anfang, meist durch den Anfangswert einer Laufvariable (d.i. ein Zähler) gegeben,
- Abbruchsbedingung (logische Bedingung),
- nächster Wert der Laufvariable.

```
for (i=1; 10 > i; i++)
      System.out.println(i*i);


while (10 > i) {
    System.out.println(i*i);
    i++;
}


do {
    System.out.println(i*i);
    i++;
} while (10 > i);
```

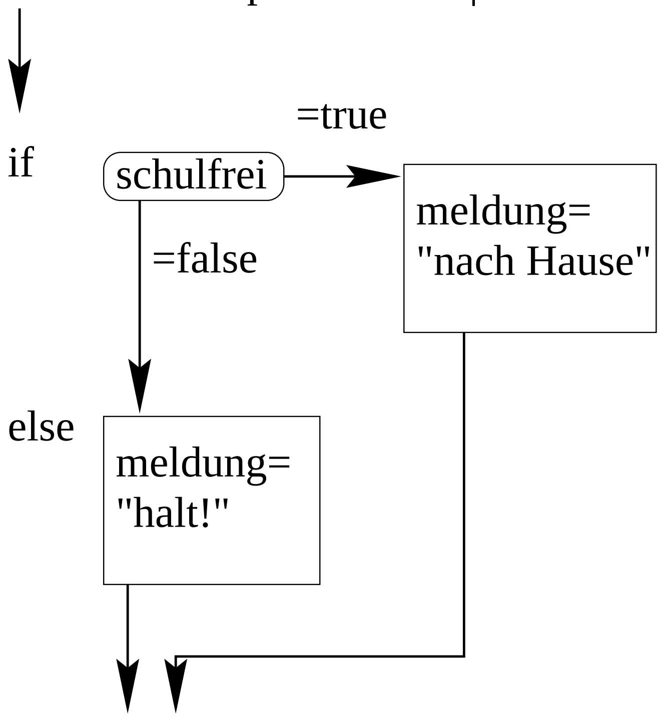
Verzweigung

- *if* (Bedingung) Block1
- *else* Block2 – optional

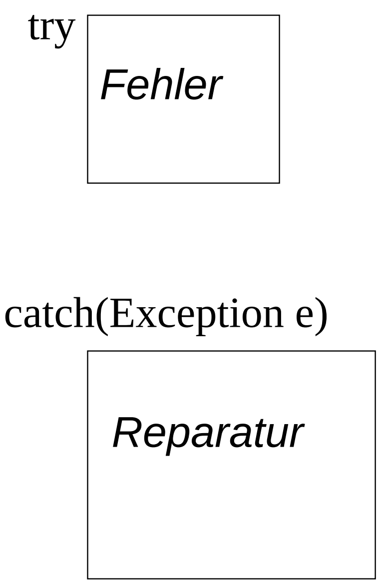
Wenn die Bedingung als Wert `true` hat, wird Block1 ausgeführt. Wenn nicht, wird fortgefahren.



schulfrei= temperatur > 39 | 15 >= temperatur;



Fehlerbehandlung



try{ } Der durch gescheifte Klammern gegebene Block ist der, in dem etwas Unvorhergesehenes passieren kann. Vielleicht erzeugt das JAVA-System zur Laufzeit des Programms ein Objekt einer Unterklasse von **Exception** oder von **Error**. Auch wenn der Block über *break*, *continue* oder *return* verlassen wird, erzeugt JAVA ein Objekte einer Unterklasse von **Exception**.

catch(*SomeException e***){ }** behandelt das Fehlerobjekt, das in dem nächsthöheren Block oder dem nächst zurückliegenden Aufruf erzeugt wurde.

```
class SchulfreiMeldung {                    //Klasse fuer Durchsagen in einer Schule

  public static void main (String argv[]) {
    String meldung;
    boolean schulfrei;
    int temperatur;

    try {                                          // Von try und catch wird die
      temperatur = Integer.parseInt (argv[0]);     //Umwandlung des Parameters
                                                   // vom Typ String in den Basistyp integer
                                                   // umhuellt, um Fehler bei der Eingabe abzufangen.
      schulfrei = temperatur > 39 | 15 >= temperatur;                 //Bedingung
      if (schulfrei)                                                  //bedingte Anweisung
        meldung = "ihr duerft nach Hause gehen";
```

```
            else
                meldung = "halt, hiergeblieben!";

            System.out.println (meldung);
        }
        catch (Exception e)  {
            System.out.println ("Ungueltige Temperaturangabe");
        }
    }
}
```

Aufruf mit

```
java SchulfreiMeldung 40
```

Die Methode **main** hat immer als Argument ein Feld mit Elementen vom Typ **String**.

switch

switch (...)

case n_1:

case n_2:

...

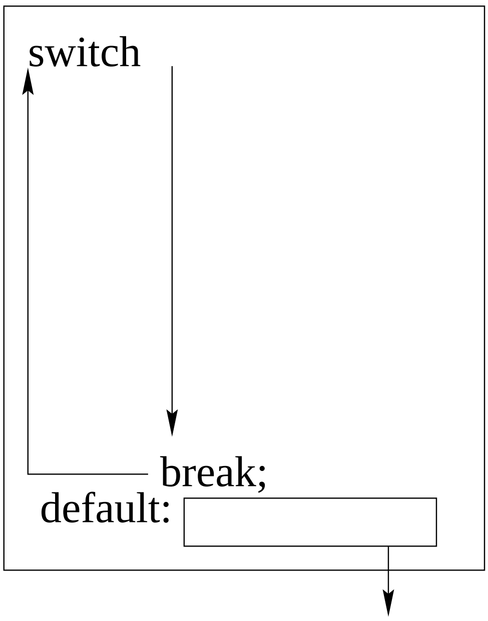
case n_i: Anweisung_i

case n_{i+1}:

...

case n_j: Anweisung_j

default: Anweisungen für restliche Fälle;



Noch eine Verzweigung

```
import AlgoTools.IO;

 class TageProMonat  {

     public static void main (String argv[])  {

         int monat = IO.readInt ("Bitte Monatsnummer [1..12] eingeben: ");
         int jahr = IO.readInt ("Bitte Jahreszahl (vierstellig) eingeben: ");
         int tage = 0;
         boolean fehler = false;

         switch (monat)  {
             case 1:                                             // Wenn Januar,
             case 3:                                             // Maerz,
```

```
case 5:                                                    // Mai,
case 7:                                                    // Juli,
case 8:                                                    // August,
case 10:                                                   // Oktober,
case 12:                                                   // Dezember,
  tage = 31;                                               // dann 31 Tage.
  break ;
case 4:                                                    // Wenn April,
case 6:                                                    // Juni,
case 9:                                                    // September,
case 11:                                                   // November,
  tage = 30;                                               // dann 30 Tage.
  break ;
case 2:                                  // Spezialfall: Februar mit Schaltjahren
  if ( ((jahr % 4 == 0) && !(jahr % 100 == 0)) || (jahr % 400 == 0) )
  tage = 29;                                   // Schaltjahr, dann 29 Tage
```

```
            else
            tage = 28;                          // Kein Schaltjahr, dann 28 Tage
            break ;
          default:                              // Monatsnummer nicht im Interval [1..12]
            System.out.println ("Kein gueltiger Monat!");
            fehler=true;
            break ;
      }

      if (!fehler) {
          System.out.println ("Dieser Monat hat "+tage+"Tage.");
      }
   }
}
```

Felder

Mehrere Daten desselben einfachen Datentyps können zu einem Feld (engl.: array) zusammengefaßt werden. Die Felder sind der Reihe nach nummeriert, beginnend bei 0. Ein Feld wird deklariert durch den Datentyp seiner Elemente und eckige Klammern.

```
int[] feldInt;
char[] feldChar;
boolean[] feldBoolean;
```

Die Länge eines Feldes wird bei der Konstruktion eines neuen Feldes durch eine Zahl in den eckigen Klammern angegeben.

```
feldInt = new int[8];
```

Zugriff auf ein Element des Feldes

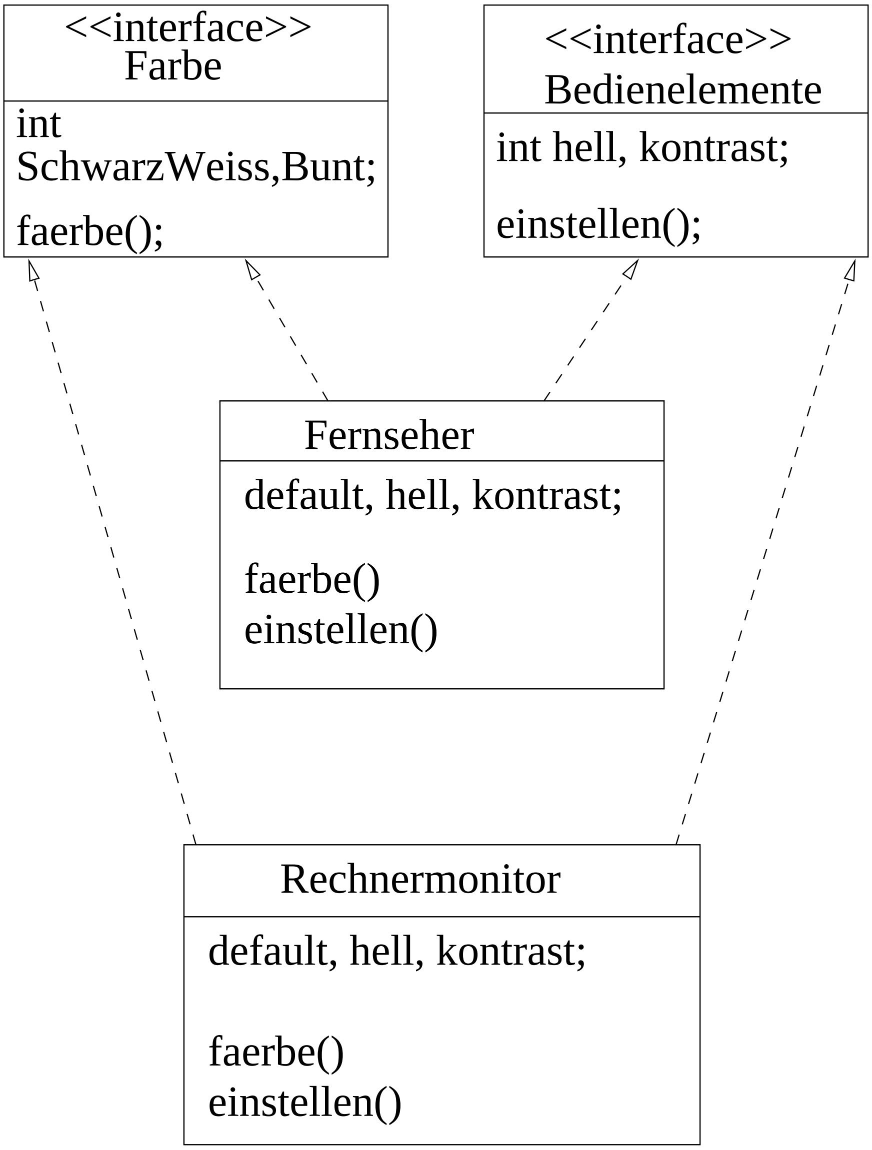
feldInt[2] = 6;
feldBoolean[0] = regnet | !regnet;

Schnittstellen

Eine Klasse, die mit dem Schlüsselwort *interface* anstelle von *class* ausgezeichnet ist.

- Eine Schnittstelle ist eine Klasse, die ausschließlich abstrakte Methoden hat.
- Eine Schnittstelle kann keine Objekte haben.
- Eine Schnittstelle kann von anderen Schnittstellen abstrakte Methoden erben. Das Schlüsselwort ist wie bei Klassen *extends*: Vererbung im Sinne von **A ist ein B**.
- Eine Klasse kann Unterklasse von mehreren Schnittstellen sein. Dies wird durch das Schlüsselwort *implements* angegeben: **A implementiert B**. Dann implementiert sie die abstrakten Methoden all dieser Schnittstellen.

Beispiel


<<interface>>
Farbe
int
SchwarzWeiss,Bunt;
faerbe();
<<interface>>
Bedienelemente
int hell, kontrast;
einstellen();
Fernseher
default, hell, kontrast;
faerbe()
einstellen()
Rechnermonitor
default, hell, kontrast;
faerbe()
einstellen()

... in JAVA

```
interface Farbe {
     int SchwarzWeiss=0, Bunt=1;
     public void faerbe ();
}

interface Bedienelemente {
     int Hell=3, Kontrast=3;
     public void einstellen ();

}
```

Verwendung der Schnittstelle

```
class Fernseher implements Farbe, Bedienelemente {
    int Default,Hell,Kontrast;
    public void faerbe () {
        Default=Bunt;
    }

    public void einstellen () {
        Hell=1;
        Kontrast=1;

    }
    public void drucke () {
        System.out.println ("TV-Farbe: "+Default);
        System.out.println ("TV-Helligkeit: "+Hell);
        System.out.println ("TV-Kontrast: "+Kontrast);
    }
}

class Rechnermonitor implements Farbe, Bedienelemente {
    int Default,Hell,Kontrast;

    public void faerbe () {
        Default=SchwarzWeiss;
    }
    public void einstellen () {
        Hell=2;
```

```
        Kontrast=2;
    }
    public void drucke () {
        System.out.println ("Monitor-Farbe: "+Default);
        System.out.println ("Monitor-Helligkeit: "+Hell);
        System.out.println ("Monitor-Kontrast: "+Kontrast);
    }
}

class InterfaceTest {

    public static void main (String[] argv) {
        Fernseher tv=new Fernseher ();
        tv.faerbe ();
        tv.einstellen ();
        tv.drucke ();
        Rechnermonitor monitor=new Rechnermonitor ();
        monitor.faerbe ();
        monitor.einstellen ();
        monitor.drucke ();
    }
}
```

Ergebnis

```
java InterfaceTest
```

TV-Farbe: 1

TV-Helligkeit: 1

TV-Kontrast: 1

Monitor-Farbe: 0

Monitor-Helligkeit: 2

Monitor-Kontrast: 2

Zugriffskontrolle

- Eine Variable kann nur verwendet werden, wenn ihr Typ (die Klasse, die ihren Wertebereich angibt) zugreifbar ist und sie selbst zugreifbar ist.
- Eine Methode kann nur verwendet werden, wenn sie selbst zugreifbar ist und die Klasse, für deren Objekte die Methode Handlungen bereitstellt.
- Ebenso kann ein Konstruktor nur verwendet werden, wenn er selbst und die Klasse, für die er Objekte erzeugt, zugreifbar ist.

Modifikatoren

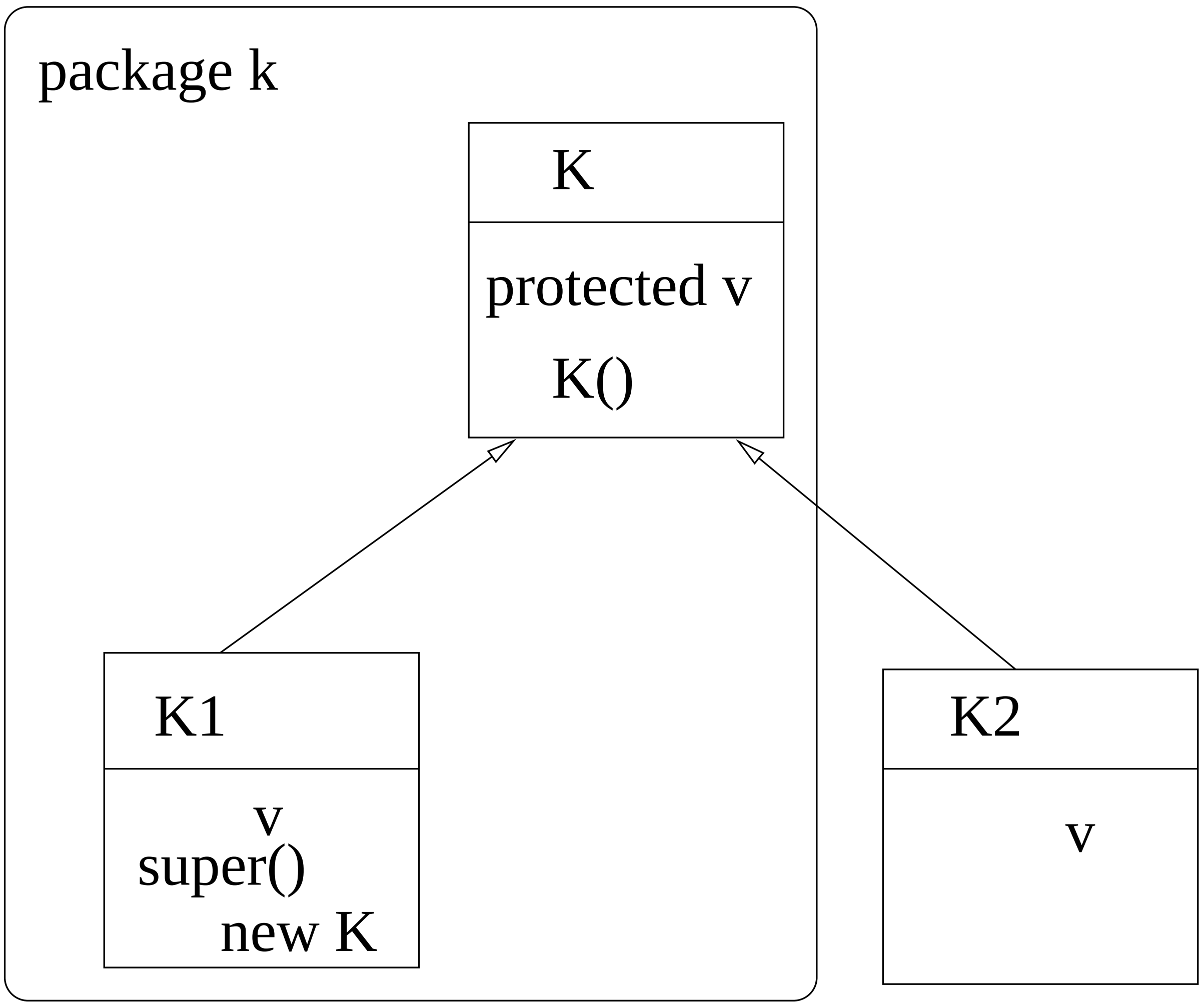
public : weltweit zugreifbar;

ohne Schlüsselwort : von dem Paket aus zugreifbar, in dem die Klassendeklaration steht;

protected : von dem Paket aus zugreifbar, in dem die Klassendeklaration steht, Variablen auch von Unterklassen außerhalb des Pakets aus;

private : nur innerhalb der Klasse zugreifbar, in der die Variable oder Methode deklariert ist. Also auch nicht erblich!

Beispiel



Sichtbarkeit von Variablen

Eine Variable kann

- eine Klasseneigenschaft – geschrieben mit dem Schlüsselwort $static$,
- eine Objekteigenschaft – deklariert am Anfang von $ClassBody$ (ohne $static$),
- ein Unikat – eine Variable von einem einfachen Datentyp,
- eine Hilfsgröße, die wir gerade mal (z.B. in einer Methode oder in einer Schleife) benötigen

ausdrücken.

Eine Klasseneigenschaft ist überall sichtbar, wo die Klasse sichtbar ist. Eine Objekteigenschaft ist ebenfalls überall sichtbar, wo die Klasse sichtbar ist, deren Objekte diese Eigenschaft haben. Ob die Klasse sichtbar (zugreifbar) ist, ergibt sich daraus, in welchem Paket und mit welchem (oder keinem) Modifikator sie deklariert wurde.

Lokale Variablen

gelten nur innerhalb des Blocks, in dem sie stehen.

Eine for-Anweisung wird wie ein Block behandelt.

Innerhalb dieses Geltungsbereichs darf der Name der lokalen Variablen nicht noch einmal auftreten. Beispielsweise darf in dem Block, in dem die lokale Variable deklariert ist, nicht noch eine for-Schleife mit einer Variable gleichen Namens vorkommen.

Außerhalb schon.

```
import ballbeispiel.*; import AlgoTools.IO;
class Mensch { String name, geschlecht; Hausrat hausrat; } ;
class Studierend extends Mensch {
    int semester,monat,jahr;
    public Studierend () {
        super ();
        semester = IO.readInt ("Im wievielten Semester ist Stud? ");
        monat = IO.readInt ("Der wievielte Monat ist jetzt?");
        jahr = IO.readInt ("In welchem Jahr? ");
    }
    public void studieren () {
        for (int i=this.monat; 13 >i;i++) {       //Monate zaehlen
            if ((i!=this.monat) && (i==4 | i==10)) {
                this .semester++;                //Semester zaehlen
                System.out.println (this .name+"ist "+this.jahr
                +"im "+semester+". Semester"); }
        }
        this .jahr++;                                     //Jahre zaehlen
        this .monat=1;
        if (9>semester)            //Studienende noch nicht erreicht?
            studieren ();                          //dann weiterstudieren
        else System.out.println ("Das Diplom!");     //sonst Diplom
    }
}
class Studi {
    private static void main (String argv[]) {
        Studierend stud;
        stud = new Studierend ();
        stud.studieren ();
        System.out.println (stud.name+ "diplomiert "+stud.jahr);
    }
}
```

Probieren Sie einmal aus, was passiert, wenn Sie statt i den Variablennamen $monat$ verwenden!