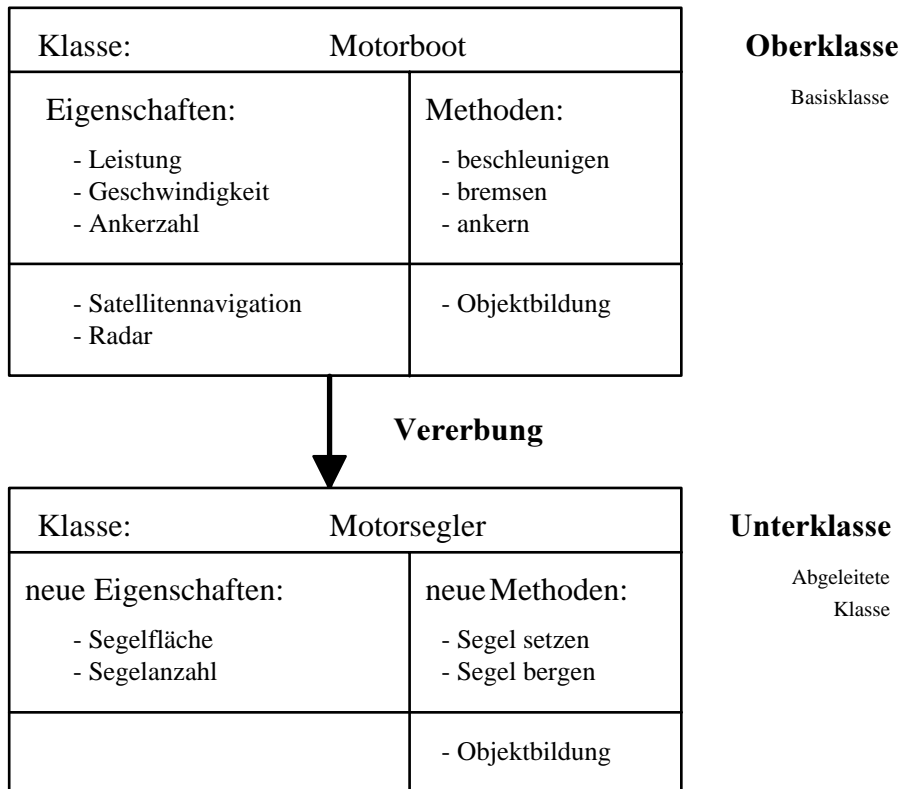


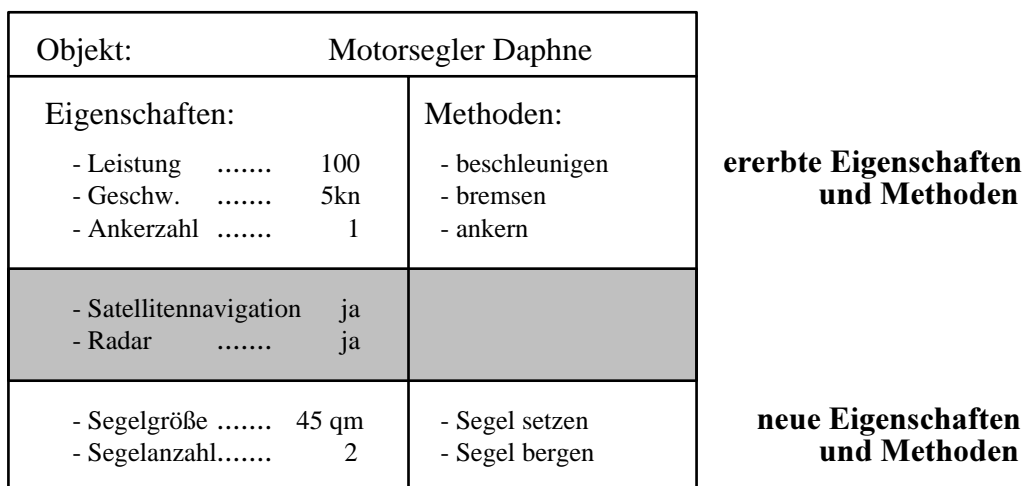
4 Vererbung

4.1 Abgeleitete Klassen

Wie im Kapitel 1 schon erläutert, kann man von einer Klasse eine neue Klasse ableiten.



Ein Objekt der abgeleiteten Klasse erbt damit die Elemente (Eigenschaften und Methoden) der Oberklasse.



4.1.1 Definition einer abgeleitete Klasse

Abgeleitete Klassentypen werden durch **class X: public Y{ ... }**; definiert, wobei X die von Y abgeleitete Klasse ist.

```

/***** Abgeleitete Klassen *****/
/*                               Boote0.h                               */

# include <stdio.h>

class boote                                // Basisklasse
{   int Geschwindigkeit;                 // max. in kn
    int Personenzahl;                    // max. Anzahl
    char *Komfort;
public:
    boote( int, int, char *);           // Konstruktor
    void drucke_daten();
};

class segelschiffe: public boote          // abgeleitete Klasse
{   char *Typ;
    int Segelflaeche;                    // im m^2
public:                                  // Konstruktor
    segelschiffe( int, int, char *, char *, int);
    void drucke_daten();
};

```

4.1.2 Konstruktoren abgeleiteter Klassen

Wird eine Unterklasse erzeugt, so werden automatisch alle vorhandenen Konstruktoren seiner Oberklasse top-down durchlaufen. Benötigt der **Konstruktor** der Basisklasse Parameter, so müssen diese vom Konstruktor der abgeleiteten Klasse weitergeleitet werden. Dies geschieht analog der Member-Objekte.

```

// Konstruktor der Basisklasse
boote::boote( int g, int p, char *k)
{   Geschwindigkeit = g; Personenzahl = p; Komfort = k;
};

// Konstruktor der abgeleiteten Klasse
segelschiffe::segelschiffe( int g, int p, char *k, char *t,
int s)
: boote( g, p, k)
// Parameterübergabe an den Konstruktor der Basisklasse
{   Typ = t; Segelflaeche = s;
};

```

Wird eine Instanz der Klasse **segelschiffe** vereinbart, so müssen sowohl Daten für die eigenen als auch für die ererbten Eigenschaften übergeben werden:

```

segelschiffe Hai( /*Daten fuer boote*/ 5, 6, "luxus",
/*Daten fuer segelschiffe*/ "Ozean", 45);

```

Insbesondere gilt das natürlich auch für Ableitungen über mehrere Stufen:

```

/***** Abgeleitete Klassen *****/
/*                               Bootel.h                               */

# include <stdio.h>

class boote
{   int Geschwindigkeit;
    int Personenzahl;
    char *Komfort;
public:
    boote( int g, int p, char *k)           // Konstruktor
    {   Geschwindigkeit = g; Personenzahl = p; Komfort = k;
};

```

```

};
void drucke_daten()
{ cout << "Hoechstgeschwindigkeit[kn]: "
  << Geschwindigkeit << "\n";
  cout << "Maximalpersonenzahl: "
  << Personenzahl << "\n";
  cout << "Ausstattung: " << Komfort << "\n";
};
};

class segelschiffe: public boote
{ char *Typ;
  int Segelflaeche;
public:
                                     // Konstruktor
  segelschiffe( int g, int p, char *k, char *t, int s)
  : boote( g, p, k)
  { Typ = t; Segelflaeche = s;
  };
  void drucke_daten()
  { boote::drucke_daten();
    cout << "Typ: " << Typ << "\n";
    cout << "Segelflaeche[m^2]: " << Segelflaeche << "\n";
  };
};

class kielboote: public segelschiffe
{ int Kielgewicht;
public:
                                     // Konstruktor
  kielboote( int g, int p, char *k, char *t, int s, int w)
  : segelschiffe( g, p, k, t, s)
  { Kielgewicht = w;
  };
  void drucke_daten()
  { segelschiffe::drucke_daten();
    cout << "Kielgewicht[kg]: " << Kielgewicht << "\n";
  };
};

```

4.1.3 Zugriff auf Klasselemente

Wie das letzte Beispiel zeigt, kann eine Instanz einer abgeleiteten Klasse auf öffentliche Elemente seiner Basisklasse zugreifen (**boote::drucke_daten()**). Der **Zugriff** auf private Elemente der Basisklasse ist nur indirekt durch ihre öffentlichen Methoden möglich, d.h. eigene Methoden, die die privaten Daten manipulieren bzw. die private Methoden aufrufen. Durch das Schlüsselwort **protected**, statt **private**, kann man den Zugriff auf Elemente für alle von der Basisklasse abgeleiteten Klassen öffnen.

```

/***** Abgeleitete Klassen *****/
/*                                     Boote2.h                                     */

# include <stdio.h>

class boote
{ protected:
                                     // oeffentlich fuer alle seine Nachfolger
  int Geschwindigkeit;
  int Personenzahl;
  char *Komfort;
public:
  boote( int g, int p, char *k)
  { Geschwindigkeit = g; Personenzahl = p; Komfort = k;
  };
  void drucke_daten()

```

```

    { cout << "Hoechstgeschwindigkeit[kn]: "
      << Geschwindigkeit << "\n";
      cout << "Maximalpersonenzahl: "
        << Personenzahl << "\n";
      cout << "Ausstattung: " << Komfort << "\n";
    };
};

class segelschiffe: public boote
{
    char *Typ;
    int Segelflaeche;
public:
    segelschiffe( int g, int p, char *k, char *t, int s)
    : boote( g, p, k)
    { Typ = t; Segelflaeche = s;
    };
    void drucke_daten() // direkter Zugriff auf Bootsdaten
    { cout << "Hoechstgeschwindigkeit[kn]: "
      << Geschwindigkeit << "\n";
      cout << "Maximalpersonenzahl: "
        << Personenzahl << "\n";
      cout << "Ausstattung: " << Komfort << "\n";
      cout << "Typ: " << Typ << "\n";
      cout << "Segelflaeche[m^2]: " << Segelflaeche << "\n";
    };
};

```

Bei der bisherigen Definition **class X: public Y{ ... }**; einer abgeleiteten Klassen unter Verwendung des Schlüsselwortes **public** ist jedes öffentliche Element von **Y** auch öffentliches Element von **X**. Lässt man aber das Schlüsselwort **public** weg **class X: Y{ ... }**; , so werden auch die öffentlichen Elemente von **Y** zu privaten Elementen von **X**, auf die von außen nicht zugegriffen werden darf. Damit haben dann die Nachfolger von **X** keine Zugriffsrechte mehr.

```

/***** Abgeleitete Klassen *****/
/*                               Boote3.h                               */

# include <stdio.h>

class boote
{
    int Geschwindigkeit;
    int Personenzahl;
    char *Komfort;
public:
    boote( int g, int p, char *k)
    { Geschwindigkeit = g; Personenzahl = p; Komfort = k;
    };
    void drucke_daten()
    { cout << "Hoechstgeschwindigkeit[kn]: "
      << Geschwindigkeit << "\n";
      cout << "Maximalpersonenzahl: "
        << Personenzahl << "\n";
      cout << "Ausstattung: " << Komfort << "\n";
    };
};

class segelschiffe: boote // implizit privat
{
    char *Typ;
    int Segelflaeche;
public:
    segelschiffe( int g, int p, char *k, char *t, int s)
    : boote( g, p, k)

```

```

    { Typ = t; Segelflaeche = s;
    };
    void drucke_daten()
    { boote::drucke_daten();          // Zugriff hier noch erlaubt
      cout << "Typ: " << Typ << "\n";
      cout << "Segelflaeche[m^2]: "
            << Segelflaeche << "\n";
    };
};

class kielboote: public segelschiffe
{
    int Kielgewicht;
public:
    kielboote( int g, int p,
              char *k, char *t, int s, int w)
    : segelschiffe( g, p, k, t, s)
    { Kielgewicht = w;
    };
    void drucke_daten()
        // kein Zugriff auf Bootsdaten durch boote::drucke_daten();
        // Zugriff auf Bootsdaten indirekt möglich!
    { segelschiffe::drucke_daten();
      cout << "Kielgewicht[kg]: " << Kielgewicht << "\n";
    };
};

```

Soll nur ein Teil der öffentlichen Elemente von **boote** weiterhin als öffentlich erklärt werden, so können diese in **segelschiffe** zu den öffentlichen Elementen aufgenommen werden. Das ist aber nur für Elemente, die in **boote** bereits als öffentliche Elemente definiert sind, möglich. Elemente mit gleichem Namen sind dann aber in **segelschiffe** nicht erlaubt.

```

/***** Abgeleitete Klassen *****/
/*                               Boote5.h                               */

# include <stdio.h>

class boote
{
    int Geschwindigkeit;
    int Personenzahl;
    char *Komfort;
public:
    boote( int g, int p, char *k)
    { Geschwindigkeit = g; Personenzahl = p; Komfort = k;
    };
    void drucke_daten()
    { cout << "Hoechstgeschwindigkeit[kn]: "
          << Geschwindigkeit << "\n";
      cout << "Maximalpersonenzahl: "
          << Personenzahl << "\n";
      cout << "Ausstattung: " << Komfort << "\n";
    };
};

class segelschiffe: boote          // implizit privat
{
    char *Typ;
    int Segelflaeche;
public:
    segelschiffe( int g, int p, char *k, char *t, int s)
    : boote( g, p, k)
    { Typ = t; Segelflaeche = s;
    };
    boote::drucke_daten;          // Bootsmethode bleibt public
};

```

```

/* Methode mit gleichem Namen darf hier nicht existieren! */
/* void drucke_daten()
{ boote::drucke_daten();
  cout << "Typ: " << Typ << "\n";
  cout << "Segelflaeche[m^2]: "
      << Segelflaeche << "\n";
};
*/
};

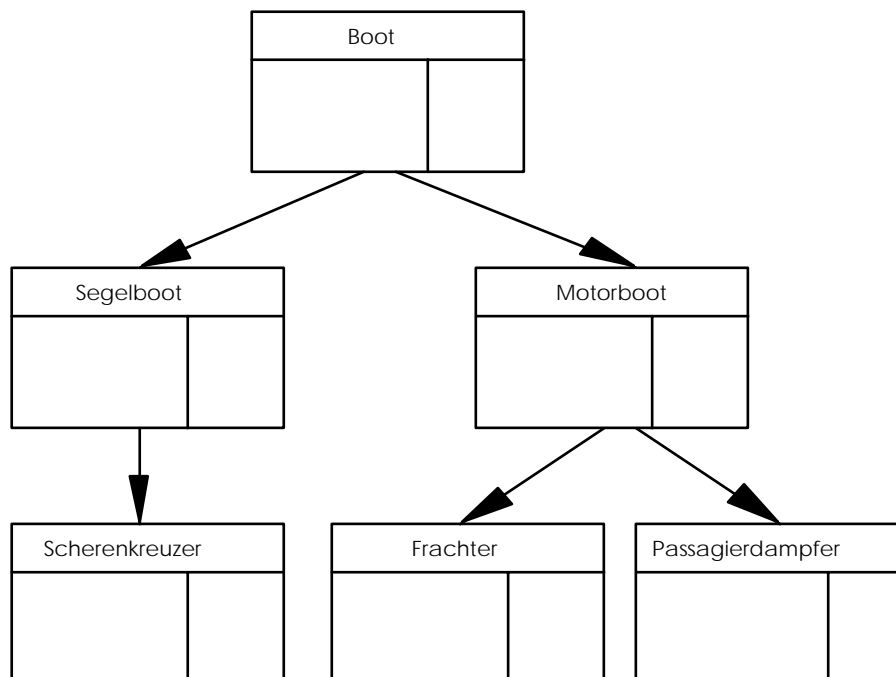
class kielboote: public segelschiffe
{ int Kielgewicht; // in kg
public:
  kielboote( int g, int p,
            char *k, char *t, int s, int w)
  : segelschiffe( g, p, k, t, s)
  { Kielgewicht = w;
  };
  void drucke_daten()
  { // Zugriff auf Bootsmethode jetzt erlaubt
    boote::drucke_daten();
    cout << "Kielgewicht[kg]: " << Kielgewicht << "\n";
  };
};

```

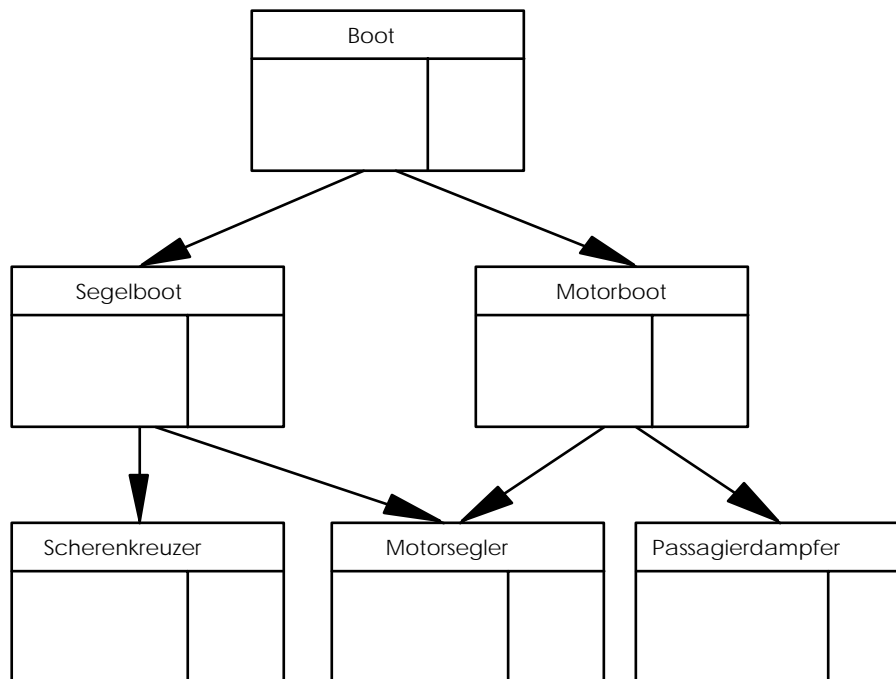
4.2 Klassenhierarchie

In einer größeren **Klassenhierarchie** mit mehreren Ebenen führt die **Extraktion** von gemeinsamen Teilen in eine gemeinsame Oberklasse zur **Wiederverwendbarkeit gemeinsamer Bausteine** und auch zu einer besseren **Lesbarkeit**, da der Betrachter einen Überblick über die von den abgeleiteten Klassen realisierten Daten und Methoden erhält. Diese Oberklasse kann eine **abstrakte Klasse** sein, d.h. eine Klasse, die selbst keine Objekte bildet bzw. bilden kann. Die Instanzen besitzen somit die Elemente der eigenen Klasse, die der "Väter", "Großväter" usw. usf.

Bei der **einfachen Vererbung** werden problemlos die Daten und Methoden der Klassen an ihre Nachfolger vererbt.



Bei der **mehrfachen Vererbung** besitzt eine abgeleitete Klasse mindestens zwei "Väter".



Hier besitzen Instanzen der Klasse **motorsegler** sowohl die Elemente der Klasse **segelschiffe** als auch die der Klasse **motorboote**.

Abgeleitete Klassen mit Mehrfachvererbung werden durch **class X: public Y, Z, ... { ... }**; definiert.

```

/***** Abgeleitete Klassen *****/
/*          Boote6.h          */
/*          Mehrfache Vererbung          */

# include <stdio.h>

class boote
{
  int Geschwindigkeit;
  int Personenzahl;
  char *Komfort;
public:
  boote( int g, int p, char *k)           // Konstruktor
  { Geschwindigkeit = g; Personenzahl = p; Komfort = k;
  };
  void drucke_daten()
  { cout << "Hoechstgeschwindigkeit[kn]: "
    << Geschwindigkeit << "\n";
    cout << "Maximalpersonenzahl: "
    << Personenzahl << "\n";
    cout << "Ausstattung: " << Komfort << "\n";
  };
};

class segelschiffe: public boote           // Einfachvererbung
{
  char *Typ;
  int Segelflaeche;
public:                                     // Konstruktor

```

```

    segelschiffe( int g, int p, char *k, char *t, int s)
    : boote( g, p, k)
    { Typ = t; Segelflaeche = s;
    };
void drucke_daten()
{ boote::drucke_daten();
  cout << "Typ: " << Typ << "\n";
  cout << "Segelflaeche[m^2]: "
        << Segelflaeche << "\n";
};
};

class motorboote: public boote // Einfachvererbung
{ int Hubraum; // in cm^3
public:
  motorboote( int g, int p, char *k, int h)
  : boote( g, p, k)
  { Hubraum = h;
  };
void drucke_daten()
{ boote::drucke_daten();
  cout << "Hubraum[cm^3]: " << Hubraum << "\n";
};
};

// Mehrfachvererbung
class motorsegler: public segelschiffe, public motorboote
{ int Anzahl_Batterie;
public: // Konstruktor
  motorsegler( int g1, int g2, int p,
               char *k, char *t, int s, int h, int b)
  : segelschiffe( g1, p, k, t, s),
    motorboote( g2, p, k, h)
  { Anzahl_Batterie = b;
  };
void drucke_daten() // muss ueberlagert werden
{ cout << "Mit Segel:\n";
  segelschiffe::drucke_daten();
  cout << "Mit Motor:\n";
  motorboote::drucke_daten();
  cout << "Batterienanzahl: "
        << Anzahl_Batterie << "\n";
};
};
};

```

Beim Erzeugen einer Instanz von **motorsegler** erhält diese die **Instanzvariable Geschwindigkeit** von **boote** zweimal, einmal über die Klasse **segelschiffe** und einmal über die Klasse **motorboote**. Dies erscheint hier sinnvoll, denn ein Motorsegler besitzt unter Segeln eine andere Höchstgeschwindigkeit als unter Motor. Darum werden beim Konstruktor zwei Parameter für die Geschwindigkeit **g1** und **g2** angegeben.

Die **Konstruktoren** werden in der Reihenfolge der Angabe der Oberklassen bei der Definition der abgeleiteten Klasse durchlaufen, d. h. zuerst der Konstruktor der Klasse **segelboote** und danach der der Klasse **motorboote**, sowie zuletzt der der Klasse **motorsegler** selbst, unabhängig von der Reihenfolge im Konstruktor selbst.

Da die Methode **drucke_daten()** von beiden Oberklassen an die abgeleitete Klasse **motorsegler** weitergegeben wird, kann es hier zu Konflikten kommen, die man durch **Überlagern** der Methode in der Klasse **motorsegler** umgeht. Der Compiler reagiert in dem Fall der mehrfachen Vererbung einer Methode mit einer Fehlermeldung.

Auch **personenzahl** und **komfort** werden zweimal angelegt, obwohl sie durch den Konstruktor mit demselben Wert belegt werden. Soll eine **Instanzvariable** bei mehrfacher Vererbung nur einmal angelegt werden, so muss eine **virtuelle Basisklasse** gebildet werden (s. später).

```

/***** Abgeleitete Klassen *****/
/*          Boote7.h          */
/*          Mehrfache Vererbung          */

# include <stdio.h>

class boote
{   int Geschwindigkeit;
    int Personenzahl;
    char *Komfort;
public:
    boote( int g, int p, char *k)
    { Geschwindigkeit = g; Personenzahl = p; Komfort = k;
    };
    void drucke_daten()
    { cout << "Hoechstgeschwindigkeit[kn]: "
      << Geschwindigkeit << "\n";
      cout << "Maximalpersonenzahl: "
      << Personenzahl << "\n";
      cout << "Ausstattung: " << Komfort << "\n";
    };
    void set_Personenzahl( int p)           // neue Methode
    { Personenzahl = p;
    };
};

class segelschiffe: public boote
{   char *Typ;
    int Segelflaeche;
public:
    segelschiffe( int g, int p, char *k, char *t, int s)
    : boote( g, p, k)
    { Typ = t; Segelflaeche = s;
    };
    void drucke_daten()
    { boote::drucke_daten();
      cout << "Typ: " << Typ << "\n";
      cout << "Segelflaeche[m^2]: " << Segelflaeche << "\n";
    };
};

class motorboote: public boote
{   int Hubraum;
public:
    motorboote( int g, int p, char *k, int h)
    : boote( g, p, k)
    { Hubraum = h;
    };
    void drucke_daten()
    { boote::drucke_daten();
      cout << "Hubraum[cm^3]: " << Hubraum << "\n";
    };
};

class motorsegler: public segelschiffe, public motorboote
{   int Anzahl_Batterie;
public:

```

```

    motorsegler( int g1, int g2, int p,
                 char *k, char *t, int s, int h, int b)
    : segelschiffe( g1, p, k, t, s),
      motorboote( g2, p, k, h)
    { Anzahl_Batterie = b;
    };
    void drucke_daten()
    { cout << "Mit Segel:\n";
      segelschiffe::drucke_daten();
      cout << "Mit Motor:\n";
      motorboote::drucke_daten();
      cout << "Batterienanzahl: " << Anzahl_Batterie << "\n";
    };
};

/***** Abgeleitete Klassen *****/
/*                                     Boote7.cc                               */

# include <stdio.h>
# include <stream.h>
# include "Boote7.h"

main()
{ cout << "\nMotorsegler DAPHNE:\n";

  motorsegler Daphne( 40, 10, 20, "jux", "neu", 20, 100, 3);
  Daphne.drucke_daten();
  // Daphne.set_Personenzahl( 5);
  // FEHLER: Personenzahl mehrdeutig!
  // Daphne.boote::set_Personenzahl( 5);
  // FEHLER: Personenzahl mehrdeutig!
  Daphne.segelschiffe::set_Personenzahl( 5); // eindeutig!
  Daphne.drucke_daten();
  return 0;
}

```

4.3 Das Schlüsselwort this

In jeder Methode einer Klasse **z** ist der Zeiger **this** als **z *this**; deklariert und so initialisiert, dass er auf die Klasseninstanz zeigt, für die eine Methode aufgerufen wurde.

```

/***** Schluesselwort this *****/
/*                                     this.h                               */

# include <stdio.h>
# include <stream.h>

class dkette
{ dkette *vor; // Vorgaenger
  dkette *nach; // Nachfolger
public:
  void einfuegen( dkette *x);
};

void dkette::einfuegen( dkette *x)
{ x -> nach = nach; // (1)
  x -> vor = this; // (2)
  nach -> vor = x; // (3)
  nach = x; // (4)
}

```

