5.5 Einführung und Klassen/Mehrere Objekte einer Klasse

5.5.1 Einführung

Bisher war in fast allen Beispielen nur von einem Objekt die Rede, das mit Inhalten gefüllt, verändert und ausgegeben wurde. Aber genauso können wir unsere Programme auch mit mehreren Objekten ausstatten.

Und so wie bisher die Records können auch Objekte als Parameter und als Rückgabewerte dienen. Und zwar sowohl bei Funktionen als auch bei Methoden derselben oder anderer Klassen.

5.5.2 Mehrere Objekte

Die Arbeit mit mehreren Objekten ist von `main()`-Seite aus völlig problemlos. Man muss nur beachten, dass man eben nur über öffentliche Methoden an die Inhalte kommt:

```
1 #include <stdio.h>
2 #include "pkwtyp.h"
3 int main(void)
4 {
5     pkwtyp cheftauto("BI-XY_490", 42345);
6     pkwtyp servicewagen("BI-AB_491", 212314);
7     pkwtyp lieferfahrzg("BI-CD_492", 89216);
8     int gesamtkilometer;
9     gesamtkilometer=cheftauto.getkmstand()+servicewagen.getkmstand()
10                    +lieferfahrzg.getkmstand();
11     printf("%d\n", gesamtkilometer);
12     return 0;
13 }
```

Zeile 2 Hier wird die Klasse `pkwtyp` eingebunden (siehe voriges Kapitel). Alternativ kann an dieser Stelle die komplette Klassendefinition stehen so wie in den Kapiteln davor.

Zeile 9+10 Hier werden aus den drei Objekten die gewünschten Attribute herausgeholt und weiterverarbeitet.

Man sieht, dieses Beispiel enthält nichts Neues.

5.5.3 Funktion erhält Objekt(e) als Parameter

Jede Funktion kann ein oder mehrere Objekte als Parameter bekommen. Aber auch hier ist der Zugriff auf die Inhalte nur über öffentliche Methoden möglich:

```
1 #include <stdio.h>
2 #include "pkwtyp.h"
3 int ermittle_gesamt_km(pkwtyp p1, pkwtyp p2, pkwtyp p3)
4 {
5     int erg;
6     erg=p1.getkmstand()+p2.getkmstand()+p3.getkmstand();
7     return erg;
8 }
9 int main(void)
10 {
11     pkwtyp cheftauto("BI-XY_490", 42345);
12     pkwtyp servicewagen("BI-AB_491", 212314);
13     pkwtyp lieferfahrzg("BI-CD_492", 89216);
14     int gesamtkilometer;
```

```

15     gesamtkilometer=ermittle_gesamt_km(chefauto , servicewagen , lieferfahrzeug);
16     printf ("%d\n", gesamtkilometer);
17     return 0;
18 }

```

Zeile 3 Hier beginnt eine Funktion, die drei Objekte des Typs `pkwtyp` als Parameter nimmt. Man beachte: Es ist keine Methode innerhalb der Klasse, sondern eine eigenständige Funktion.

Zeile 6 Weil `ermittle_gesamt_km()` eine eigenständige Funktion ist, kommt man an den Inhalt des `private`-Attributs `kmstand` nur über den Aufruf der `public`-Methode `getkmstand()`.

Ein anderes Beispiel wäre die Vergleichsfunktion `hatmehrkmstand()`:

```

1  int hatmehrkmstand(pkwtyp a, pkwtyp b)
2  {
3      if(a.getkmstand()>b.getkmstand()) return 1;
4      if(a.getkmstand()<b.getkmstand()) return -1;
5      return 0;
6  }

```

Der Aufruf in `main()` wäre dann:

```

1     x=hatmehrkmstand(chefauto , servicewagen);
2     if(x>0)
3         printf ("Chef-Auto_hat_mehr_km_gefahren\n");

```

Dieses zweite Beispiel soll im nächsten Kapitel anders realisiert werden.

5.5.4 Methode erhält Objekt(e) als Parameter

Nun soll der Vergleich aus dem vorigen Kapitel in einer *Methode* programmiert werden. Der Vergleich soll also – was den Quelltext betrifft – nicht mehr in einer Funktion (also außerhalb der Klasse `pkwtyp`) stattfinden, sondern in einer Methode der Klasse `pkwtyp` (also innerhalb dieser Klasse).

Zur Erinnerung: Eine Methode ist eine Funktion, die an eine Klasse und damit auch an ein Objekt gebunden ist. Deshalb braucht eine Methode das Objekt selbst nicht als Parameter: Statt

```

1     printpkw(chefauto); // Aufruf einer Funktion

```

reicht es,

```

1     chefauto.printpkw(); // Aufruf einer Methode

```

zu schreiben. Den sonst nötigen Parameter braucht die Methode nicht, denn sie kennt ja die Attribute ihres Objekts.

Genauso ist es hier: Auch hier brauchen wir das Objekt, mit dem die Methode aufgerufen wird, nicht als Parameter. Damit sieht dann der Aufruf so aus:

```

1     x=chefauto.hatmehrkmstand(servicewagen);

```

Die Methode könnte dann so realisiert werden:

```

1  struct pkwtyp
2  {
3      ...
4  public:
5      int hatmehrkmstand(pkwtyp b)

```

```

6   {
7       if (kmstand > b.getkmstand ()) return 1;
8       if (kmstand < b.getkmstand ()) return -1;
9       return 0;
10  }
11 };

```

Zeile 5 Jetzt ist nur noch ein Parameter nötig. Das Objekt `a` aus dem vorigen Beispiel ist jetzt das Objekt, mit der die Methode aufgerufen wird. Das Objekt `b` ist nun der einzige Parameter.

Zeile 7 Im Vergleich zum vorigen Beispiel heißt es hier

```

1   if (kmstand > b.getkmstand ()) return 1;

```

statt

```

1   if (a.getkmstand () > b.getkmstand ()) return 1;

```

Das hat zwei Gründe:

- a) Erstens gibt es `a` nicht in dieser Methode (es wurde ja nur *ein* Parameter übergeben), stattdessen sind in der Methode die Attribute und Methoden dieses Objektes sichtbar. Damit könnte hier stehen:

```

1   if (getkmstand () > b.getkmstand ()) return 1;

```

- b) Zweitens ist innerhalb des Objektes der Zugriff auf *private*-Elemente möglich, damit kann hier stehen:

```

1   if (kmstand > b.getkmstand ()) return 1;

```

Und jetzt kommt ein entscheidender Vorteil des Methodenaufrufs gegenüber einem normalen Funktionsaufruf ins Spiel: Weil die Methode zur Klasse `pkwtyp` gehört, kann sie auf die `private`-Elemente des anderen Objekts (Parameter `b`) zugreifen. Merke: **Ein Objekt hat Zugriff auf die `private`-Daten aller anderen Objekte derselben Klasse**. Anders gesagt: Der Zugriffsschutz greift nur außerhalb der Klasse. In C++ bedeutet `private` somit nur Klassen-privat, nicht Objekt-privat.

Und was nützt das? Die Methode kann direkt auf alle Attribute des Parameter-Objekts zugreifen, welches es (typisch für C und C++) als Kopie bekommen hat:

```

1   struct pkwtyp
2   {
3       ...
4   public:
5       int hatmehrkmstand (pkwtyp b)
6       {
7           if (kmstand > b.kmstand) return 1;
8           if (kmstand < b.kmstand) return -1;
9           return 0;
10      }
11 };

```

Zeile 7 Man sieht: Jetzt kommt man auch bei `b` ohne Aufruf einer `get`-Methode aus.

Das ist besonders für Fälle wichtig, in denen es keine `get`- oder `set`-Methoden für einzelne Attribute gibt.

5.5.5 Funktion gibt Objekt zurück

Jede Funktion kann auch ein Objekt als Rückgabetyt ausgeben. Wie bei normalen Funktionen üblich ist auch hier der Zugriff auf die Inhalte nur über öffentliche Methoden möglich. Hier ein Beispiel¹:

```

1 #include <stdio.h>
2 #include "pkwtyp.h"
3 pkwtyp erstelle_neu(pkwtyp altfahrzeug)
4 {
5     pkwtyp erg;
6     erg.set_kmstand=0;
7     erg.set_kennz(altfahrzeug.getkennz());
8     return erg;
9 }
10 int main(void)
11 {
12     pkwtyp chefauto("BI-XY_490", 42345);
13     pkwtyp neufahrzeug;
14     neufahrzeug=erstelle_neu(chefauto);
15     neufahrzeug.printpkw();
16     return 0;
17 }
```

5.5.6 Methode gibt Objekt zurück

Dasselbe Ziel soll jetzt mit einer Methode erreicht werden. Beim Aufruf einer Methode braucht man das Objekt selbst nicht als Parameter übergeben, es kann deshalb wegfallen:

```
1     neufahrzeug=chefauto.erstelle_neu();
```

Die Methode könnte dann so aussehen:

```

1 struct pkwtyp
2 {
3     ...
4 public:
5     pkwtyp erstelle_neu(void)
6     {
7         pkwtyp erg;
8         erg.set_kmstand(0);
9         erg.set_kennz(kennz);
10        return erg;
11    }
12 };
```

Zeile 7 Hier wird für den Rückgabewert ein *neues* Objekt der Klasse *pkwtyp* angelegt. In dieser Methode haben wir also auch zwei Objekte – einmal das Objekt, mit dem die Methode aufgerufen wurde und dazu das Objekt *erg*.

Zeile 8–9 Hier werden brav die set-Methoden für das Objekt *erg* aufgerufen.

¹bei dem ein neues Fahrzeug das alte Kennzeichen des zu verschrottenen Chefautos bekommt

Wie schon oben genannt, kann eine Methode auf die `private`-Elemente der anderen Objekte der Klasse zugreifen. Das umfasst in diesem Beispiel auch das Objekt `erg`². Damit kann die Methode auch so geschrieben werden:

```

1 struct pkwtyp
2 {
3     ...
4 public:
5     pkwtyp erstelle_neu(void)
6     {
7         pkwtyp erg;
8         erg.set_kmstand=0;
9         strcpy(erg.kennz, kennz); // string.h nicht vergessen ...
10        return erg;
11    }
12 };

```

Zeile 8-9 Hier werden die Attribute des Objekts `erg` direkt verändert. Funktioniert, weil `erg` in derselben Klasse ist wie diese Methode.

5.5.7 Fazit

Wenn man in einer Funktion oder Methode mit mehreren Objekte einer Klasse gleichzeitig arbeitet, dann haben Methoden den großen Vorteil, dass man mit ihnen an die `private`-Elemente all dieser Objekte herankommt, mit Funktionen dagegen nicht³.

Allerdings gibt es in C++ die sogenannten Friend-Funktionen. Damit kann man auch einzelnen Funktionen die Möglichkeit geben, auf `private`-Elemente der Objekte einer Klasse direkt zuzugreifen. Diese Möglichkeit wird allerdings nur selten benutzt, weil sie der Idee der Zugriffsbeschränkung entgegensteht.

In der Praxis kommen daher oft Methoden einer Klasse vor, die ein Objekt derselben Klasse als Parameter haben und auf diese Art mit zwei Objekten umgehen. Deshalb ist es notwendig, mit dieser Art der Programmierung vertraut zu werden (siehe folgender Quelltext von `pkwtyp5.cpp`).

```

1 #include <stdio.h>
2 #include <string.h>
3 struct pkwtyp
4 {
5 private:
6     char kennz[20];
7     int kmstand;
8 public:
9     pkwtyp(void)
10    {
11        kennz[0] = '\0';
12        kmstand=0;
13    }
14    pkwtyp(const char *knz, int kms)
15    {
16        kennz[0] = '\0';
17        strncat(kennz, knz, 19);
18        kmstand=kms;
19    }

```

²Es wird zwar in der Methode angelegt, ist aber trotzdem ein fremdes Objekt.

³Dass man mit Methoden einen Parameter einspart, ist dagegen eher unbedeutend.

```
20 void printpkw(void)
21 {
22     printf("%s,_%d\n", kennz, kmstand);
23 }
24 int hatmehrkmstand(struct pkwtyp b)
25 {
26     if(kmstand>b.kmstand) return 1; // geht, weil gleiche Klasse
27     if(kmstand<b.kmstand) return -1; // geht, weil gleiche Klasse
28     return 0;
29 }
30 struct pkwtyp erstelle_neu(void)
31 {
32     struct pkwtyp erg;
33     erg.kmstand=kmstand; // geht, weil gleiche Klasse
34     erg.kennz[0]='\0'; // geht, weil gleiche Klasse
35     strncat(erg.kennz, kennz,19); // geht, weil gleiche Klasse
36     return erg;
37 }
38 };
39 /*****/
40 int main(void)
41 {
42     struct pkwtyp chefauto("BI-XY_490", 42345);
43     struct pkwtyp servicewagen("BI-AB_491", 212314);
44     struct pkwtyp lieferfahrzeug("BI-CD_492", 89216);
45     struct pkwtyp neuwagen;
46     if(chefauto.hatmehrkmstand(servicewagen))
47     {
48         neuwagen=chefauto.erstelle_neu();
49     }
50     else
51     {
52         neuwagen=servicewagen.erstelle_neu();
53     }
54     printf("Neuwagen:_");
55     neuwagen.printpkw();
56     return 0;
57 }
```