Grundlagen der Objektorientierung :

Objektorientierung ermöglicht (unter anderem), daß man Software, die von einem selbst, meist aber von Anderen geschrieben ist, komfortabel nochmal verwenden kann.

Diese Idee ist nicht neu, wird aber in der objektorientierten Programmierung konsequenter als früher durchgeführt.

Die Quelle besteht aus Programmen und dazugehörigen Variablen.
Die Programme nennt man **Methoden**, die Variablen heißen **Attribute**.

Bildlich vorgestellt, macht man beim Benutzen der Quellstruktur so etwas wie einen Stempelabdruck in sein eigenes Programm.

Der Stempel heißt **Klasse**, der Abdruck (auch Instanz der Klasse genannt) heißt **Objekt**. Viele solcher Klassen sind in einer **Klassenbibliothek** zusammengefasst.

Beispiel :

Ein Zulieferer für Autotüren liefert Türen und dazu eine Klassenbibliothek für die Funktionssoftware des kleinen Prozessors, der in die Tür eingebaut ist. Der Autobauer bindet diese dann in seine Fahrzeugsoftware ein.

Eine mögliche Klasse wäre *Fensterbedienung*, darin enthalten die Methoden zum Öffnen und Schließen des Fensters, mit Endanschlagüberwachung, Maximalkraftbegrenzung usw.

Mögliche Attribute wären vielleicht die Zustände der Endschalter, der Wert des Motorstroms, und weitere sinnvolle technische Werte.

Der Programmierer wird den Stempel *Fensterbedienung* wahrscheinlich vier mal benutzen. Er erzeugt ein Objekt *Fahrertür*, ein Objekt *Beifahrertür* und so weiter.

Die Methoden und Attribute der einzelnen Instanzen sind nun so aufrufbar :

Fahrertür.Motorstrom beinhaltet den Strom am Fensterheber der Fahrertür.
Beifahrertür.Motorstrom den Strom am Hebermotor der Beifahrertür.

Wichtig : Das sind verschiedene Speicherzellen, die sich nicht überschreiben !

Selber schreiben :

1) Definitionsdatei für eine Klasse : Die header-Datei

Fahrzeugtuer.h

```
class fahrzeugtuer                    (so soll die Klasse heißen)
{
  public :                            (public-Elemente sind von außen zugänglich,
                                       als private definierte Elemente nicht)

    unsigned char strom;              (public, also außen zugreifbares Attribut)

    fahrzeugtuer();                   (der sogenannte Konstruktor. Läuft jedes
                                       mal, wenn die Klasse instanziert wird )

    void fensterrauf();               (eine Methode, also ein Programm)
    void fensterrunter();             (noch eine ..)
}
```

Bemerkungen :

In der Informatikwelt sind public-Objekte verpönt. Es bietet strukturelle Vorteile, wenn Attribute von außen nicht zugänglich (gekapselt) sind. Sie brauchen dann aber Methoden, die den Zugriff ausführen (get/set). In der Geräteprogrammierung wird das deshalb oft mit public gemacht.

Der Konstruktor läuft bei der Bildung der Instanz im Programm. Damit kann man z.b. Anfangswerte setzen. Wenn man das nicht braucht, kann man ihn weglassen, der Compiler erzeugt dann einen Standardkonstruktor.

2) Konstruktor und Methoden : Die cpp-Datei

fahrzeugtuer.cpp

```
#include "fahrzeugtuer.h"                    (Definitionsdatei einbinden)

fahrzeugtuer:: fahrzeugtuer()                (Der Konstruktor. Hier wird
{                                             für den Strom ein Anfangswert
   strom = 0;                                 gesetzt)
}

void fahrzeugtuer:: fensterrauf()            (Eine Methode, hier Setzen eines
{                                             Ausgangsbits)
   PORTB |= (1<<PB7);
}

void fahrzeugtuer:: fensterrunter()
{
   PORTB |= (1<<PB7);
}
```

3) Benutzen der Klasse im eigenen Programm :

Main.cpp

```
#include " fahrzeugtuer.h "                  (Klasse einbinden)

fahrzeugtuer fahrertuer;                     (Objekt erzeugen)
fahrzeugtuer beifahrertuer;                  (Objekt erzeugen)
fahrzeugtuer hintenlinkstuer;                (Objekt erzeugen)
fahrzeugtuer hintenrechtstuer;               (Objekt erzeugen)

int main(void)                               (hier geht das Hauptprogramm los)

{
    while(1)                                 (in einer Schleife..)
    {
        if (lenkrad.taster14 == 1)           (prüfen ob das Attribut taster14 1 ist)
        {
                fahrertuer. fensterrauf();   (wenn ja : Methode fensterrauf an
        }                                     Fahrertür ausführen)
    }
}
```

4) Mit dem Arduino ausprobieren

Definitionsdatei der Klasse led :

led.h

```
class led
{
   public:

      led();
      void an();
      void aus();

};
```

Code in der Klasse led.cpp :

```
#include "led.h"
#include <avr/io.h>

led::led()
{
   DDRB |= (1<<PB7);     //(LED an PB7: Vorsicht , stimmt nicht bei allen Typen !)
}

void led::an()
{
   PORTB |= (1<<PB7);
}

void led::aus()
{
   PORTB &= ~(1<<PB7);
}
```

Bemerkung :

PORTB |= (1<<PB7) setzt das Bit 7 von Port B = 1
Oder man schreibt : **PORTB = PORTB | 128**
Man könnte auch : **PORTB = 128** schreiben, wenn einem die restlichen Bit von PORTB egal sind (werden =0)

Ähnlich : **PORTB &= ~(1<<PB7)** setzt das Bit wieder 0
Geht auch mit : **PORTB = PORTB & 127**

Hauptprogramm : main.cpp

```
#include "led.h"
#include "util/delay.h"

led licht;

int main(void)
{
        while(1)
          {

                licht.an();
                _delay_ms(100);
                licht.aus();
                _delay_ms(100);

          };
          return 0;
}
```