# Abend 3, 3. Semester
# Vererbung

## Ziel, Inhalt

- Wir lernen heute Klassen zu erweitern, indem wir Klassen erzeugen, die sich wie bereits bestehende verhalten, und diese ersetzen

# Vererbung

Bereits mehrmals haben wir die Ähnlichkeit vom *cin*-Objekt mit einem *ifstream*-Objekt oder vom *cout*-Objekt und *ofstream*-Objekten benutzt. Das *cout*-Objekt ist ein Objekt der Klasse *ostream*, eine Klasse, die offenbar fast gleich ist wie die Klasse *ofstream*. Tatsächlich ist die Klasse *ofstream* eine spezielle *ostream* Klasse. In C++ und anderen Objektorientierten Sprachen ist es möglich Klassen zu erzeugen, indem man bereits bestehende Klassen mittels Vererbung erweitert.

## Verwendung von Vererbung

Damit sich eine Klasse grundsätzlich wie eine andere Klasse verhält brauchst Du nur von der bereits bestehenden Klasse abzuleiten. Hier ein kleines Beispiel:

```
class Fahrzeug
{
    public:
        // Konstruktor
        Fahrzeug();
        // Destruktor
        ~Fahrzeug();

        // Methode mit Datentypen, die an
        // einem anderen Ort definiert sind
        void fahre(Richtung r, Geschwindigkeit g);

    private:
        Punkt           m_ort;
        Richtung        m_richtung;
        Geschwindigkeit m_geschwindigkeit;
};

class Auto : public Fahrzeug
{
    public:
        // Konstruktor
        Auto();
        // Destruktor
        ~Auto();

        // spezielle Methode von einem Auto
        void hupe();
    private:
        Hubraum m_hubraum;
};
```

Ein *Auto* ist ein spezielles *Fahrzeug* und erbt alle Datenelemente und Methoden von *Fahrzeug*.Überall dort wo ein Objekt der Klasse *Fahrzeug* gebraucht wird, kann auch ein Objekt der Klasse *Auto* verwendet werden.

**Begriffe**

- **Ableitung :** Die Klasse *Auto* leitet von der Klasse *Fahrzeug* ab.
- **Spezialisierung :** Die Klasse *Auto* ist eine Spezialisierung der Klasse *Fahrzeug*.
- **Basisklasse :** Die Klasse *Fahrzeug* ist die Basisklasse der Klasse *Auto*.

## Beispiel Fahrzeug

Wir schreiben eine kleine Funktion *BewegeFahrzeug:*

```
void BewegeFahrzeug(Fahrzeug& f)
{
    f.fahre(34, 50);
}
```

Ein *main* könnte so aussehen:

```
int main()
{
    Fahrzeug einFahrzeug;
    Auto     einAuto;

    BewegeFahrzeug(einFahrzeug);
    BewegeFahrzeug(einAuto);

    // hupe funktioniert nur bei Auto
    einAuto.hupe();

    // fahre funktioniert bei jedem
    // Fahrzeug
    einAuto.fahre(22, 30);
    einFahrzeug.fahre(12, 12);

    return 0;
}
```

Wenn wir ein Objekt der Klasse *Auto* erzeugen, erzeugen wir auch ein Objekt der Klasse *Fahrzeug*, also alle Datenelemente von der Klasse *Fahrzeug* werden erzeugt. Zusätzlich enthält ein Auto ein Datenelement *m_hubraum*.

**Zugriff auf Datenelemente**

Obwohl ein Objekt der Klasse *Auto* auch die Datenelemente von einem *Fahrzeug* hat, kann in einer Methode der Klasse Auto nicht auf diese *private*-Datenelemente zugegriffen werden. Das heisst in der Methode *hupe()* können wir nicht zum Beispiel auf das Datenelement *m_geschwindigkeit* zugreifen. Falls wir eine Basisklasse schreiben und

zulassen wollen, dass abgeleitete Klassen auf Datenelemente zugreifen, werden wir zum *private*-Attribut noch ein anderes, das *protected*-Attribut kennenlernen. Dazu später mehr!

**Mehrstufige Vererbung**

Es ist möglich von einer bereits abgeleiteten Klasse weiter abzuleiten :

```
class Sportwagen : public Auto
{
// ....
};
```

Auch ein Objekt der Klasse Sportwagen ist auch ein Fahrzeug. Zusätzlich ist ein Sportwagen auch ein Auto. Man kann also für ein Objekt der Klasse Sportwagen auch die Methode *hupe()* aufrufen. Die Klasse *Fahrzeug* ist so eine indirekte Basisklasse der Klasse *Sportwagen*.

## Redefinition von Elemenfunktionen und Datenelementen

Ein Objekt der Klasse *Auto* hat also auch eine Methode *fahre()*. Es ist möglich diese Methode zu redefineren um in der abgeleiteten Klasse zusätzlichen oder anderen Code auszuführen. Wir könnten zur Klasse *Fahrzeug* eine Methode *display()* hinzufügen. Wenn wir diese Methode in der Klasse Auto auch definieren, spricht man von Redefinition.