Abend 5
Vererbung

Ziel, Inhalt

- Wir sind ab heute in der Lage Klassenhierarchien zu verstehen und Objekte von abgeleiteten Klassen als Objekte der Basisklasse anzuwenden

Vererbung

Übung Vererbung

Um die Vererbung ein wenig besser in den Griff zu bekommen versuchen wir ein kleines Beispiel zu implementieren.

Vorbereitung

Als erstes erstellen wir ein neues Projekt. In diesem Projekt werden wir zwei Klassen definieren und in einem *main* Objekte dieser Klasse erzeugen. Durch Typumwandlungen werden wir sehen wie wir ein Objekt einer abgeleiteten Klasse als Objekt der Basisklasse verwenden.

Klasse Auto

Erstelle eine Klasse *Auto*. Wie immer erstellst du dafür eine Header und eine Quellcode-Datei (.h und .cpp). Die Klasse enthält folgende Datenelemente : Kennzeichen, Hersteller, Farbe. Als Methode ist sicher der Konstruktor mit Parametern wichtig. Definiere ihn so, dass alle Datenelemente mit dem Konstruktor gesetzt werden. Definiere auch eine Methode *display,* welche die Datenelemente auf ein *std::ostream*-Objekt ausgibt, das der Methode als Argument übergeben wird.
Definiere zusätzlich noch den == operator (Gleichheit) und den Kopierkonstruktor.

Klasse BMW, oder was immer dir einfällt

Definiere nun die Klasse *BMW*. Die Klasse *BMW* leitet von der Klasse *Auto* ab. Selbstverständlich darfst du auch eine Klasse *Renault, Opel* oder so definieren. Diese Klasse hat zusätzlich zu den Datenelementen der Basisklasse ein Datenelement für die Typenbezeichnung und eines das angibt, ob das Fahrzeug einen Reihensechszylinder (gilt nur für BMW) hat. Für eine andere Marke wählst du am besten eine andere Eigenschaft aus, die spezifisch für diese Marke ist. Definiere einen Konstruktor, der aussieht wie der Konstruktor der Klasse Auto. Nur den Hersteller muss man nicht mehr als Argument angeben. Ruf im Konstruktor der Klasse *BMW* den Konstruktor der Klasse *Auto* auf, was in der Initialisierungsliste geschehen muss. Hier ein kleines Beispiel mit einer Klasse *A* und *B*.

```
class A
{
public:
    // Konstruktor mit Parameter
    A(int data) : m_data(data)
    {
    }

private:
    int m_data;
};

class B : public A
{
public:
    // Konstruktor
    B();
};

// Der B Konstruktor muss den
// Konstruktor von A explizit
// aufrufen
B::B()
  :A(10) // Konstruktoraufruf
         // in der Initialisierungsliste
{
}

int main()
{
    A einA(5);

    B einB;

    return 0;
}
```

Redefiniere auch für deine Klasse (*BMW*) den operator == und schreibe einen Kopierkonstruktor. Definiere auch hier eine Methode zur Ausgabe wie in der Basisklasse. Um diese Methode zu implementieren kannst du die Methode der Basisklasse aufrufen (mit Sichtbarkeitsoperator Auto::ausgabe(...)), oder du kannst direkt auf die Datenelemente der Basisklasse zugreifen, du musst hierfür aber vorher das nächste Kapitel lesen.
Bei der Implementation wirst du möglicherweise versuchen ein Datenelement der Basisklasse zu setzen oder zu lesen. Der Zugriff auf diese Datenelemente ist jedoch nicht möglich, da sie *private* sind. Probier es aus, der Compiler wird Fehler melden. Lies das nächste Kapitel!

Das Schlüsselwort protected

Zugriff auf Datenelemente der Basisklasse

Eine abgeleitete Klasse kann nicht auf die privaten Datenelemente der Basisklasse zugreifen. Sie kann auch nicht private Methoden der Basisklasse aufrufen. Es gibt jedoch die Möglichkeit in der Basisklasse Datenelemente für abgeleitete Klassen verfügbar zu machen.
Bis jetzt kannten wir nur die Zugriffsspezifizierer *public* und *private*. Wenn wir Datenelemente oder Methoden in einem *protected* Teil definieren, sind sie in der abgeleiteten Klasse sichtbar.

```
class A
{
public:
    // Konstruktor
    A() {}

protected:
    int m_proData;
private:
    int m_data;
};

class B : public A
{
public:
    // Konstruktor
    B();
};

B::B()
{
    // Zugriff auf Protected
    // Datenelement OK
    m_proData = 30;
    // Zugriff auf Private
    // Datenelemente verboten
    m_data = 10; // Kompilerfehler!
}
```

Es gilt also die Datenelemente der Klasse Auto *protected* zu machen, die in der abgeleiteten Klasse von nutzen sein könnten.

Test der neuen Klassen

main

Schreibe nun ein main, in dem du Objekte der Klasse *Auto* und der abgeleiteten Klasse erzeugst. Teste zuerst die Klasse *Auto*, insbesondere den operator == und den Kopierkonstruktor.
Im zweiten Schritt erzeugst du Objekte der abgeleiteten Klasse (z.B. *BMW*) und probierst auch hier alle Methoden, natürlich auch den operator == und den Kopierkonstruktor.

Austauschbarkeit der Objekte

Probier nun aus, ob du einem Objekt der Klasse *Auto* ein Objekt der abgeleiteten Klasse zuweisen kannst und umgekehrt. Welche Klasse von den beiden kann an Stelle der anderen verwendet werden?

Antwort:

Kannst du auch erklären wieso? (schau auch im Buch ab Seite 553)

Antwort:

Zeiger auf Objekte

Definiere nun Zeiger auf die Klasse *Auto* und auf die abgeleitete Klasse und weise ihnen die Adresse eines Objektes der einen oder anderer Klasse zu. Auch das funktioniert nur für die eine Klasse immer.
Erzeuge auch mit *new* das eine oder andere Objekt und versuche auch hier herauszufinden, was der Zeiger auf das Objekt für einen Datentypen hat.
Du kannst auch einen Zeiger auf den einen Datentypen gleichsetzen mit einem Zeiger des anderen Datentypen. Rufe jeweils die Methode *ausgabe* auf, einmal mit dem einen Zeiger (*Auto**) das andere mal mit dem anderen Zeiger (*BMW**). Welche Methode wird wann ausgeführt?