8.1 Vererbung/Einführung

8.1.1 Anbauen ohne Baustelle?

Es gibt eine fertige Klasse `mitglied`, die um das Attribut `int mgart` erweitert werden soll (1: Vollmitglied, 2: Familienangehöriger, 3: Gast). Dabei soll die bestehende Klasse `mitglied` nicht verändert werden (sie läuft mittlerweile stabil) — Was ist zu tun?

In solchen Fällen wird in objektorientierten Programmiersprachen *Vererbung* angewandt. Mit Vererbung kann man eine bestehende Klasse erweitern, ohne sie selbst zu verändern.

8.1.2 Was ist Vererbung?

Wie bei der Aggregation (Klasse-enthält-Objekt) wird bei der Vererbung das bestehende Objekt in einen neuen Zusammenhang eingebettet.

Während es jedoch bei der Aggregation keinen Zugriff auf die privaten Komponenten des eingebetteten Objekts gibt, hat man bei Vererbung diesen Zugriff durchaus – falls der Programmierer der Klasse des eingebetteten Objektes das ausdrücklich zugelassen hat.

Abbildung 1 zeigt das UML-Symbol für die Vererbung und die dazugehörigen Begriffe, wenn die Klasse `mitglied` durch Vererbung zur Klasse `xmitglied` erweitert wird.

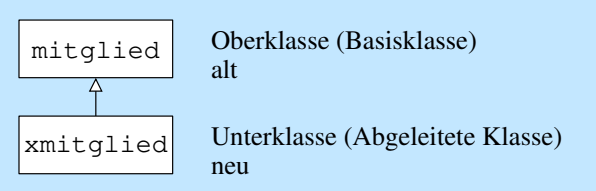

Abbildung 1: Vererbung in UML

8.1.3 Wie sieht Vererbung in C aus?

Im folgenden Programmausschnitt (die Methodendefinitionen fehlen hier aus Gründen der Übersichtlichkeit) wird die Klasse `punkt` zur Klasse `farbpunkt` erweitert. Ein Objekt der Klasse `punkt` soll einen Punkt darstellen, ein Objekt der Klasse `punkt` soll einen Punkt in beliebiger Farbe.

```
1  class punkt
2  {
3  protected:
4      int x, y;
5  public:
6      void set_x(int);
7      void set_y(int);
8      void display(void);
9  };
10 ////////////////////////////
11 class farbpunkt: public punkt
12 {
13 private:
14     int farbe;
15 public:
16     void set_farbe(int);
17     void farbe_display(void);
18 };
19 ////////////////////////////
```

```
20 int main(void)
21 {
22     farbpunkt f;
23     f.set_x(5);
24     f.set_y(6);
25     f.display();
26     f.set_farbe(200);
27     f.farbe_display();
28     return 0;
29 }
```

Gleich in der ersten Zeile der Klassendeklaration von `farbpunkt` wird (ab dem Doppelpunkt) die Vererbung bekanntgegeben:

```
1 class farbpunkt: public punkt
```

In `main()` ist zu sehen, dass man nun in der Klasse `farbpunkt` alle Attribute und Methoden von `punkt` benutzen kann, und das mit der gleichen Schreibweise, als wenn es die eigenen Attribute und Methoden wären.

Damit man auch auf die vormals als `private` deklarierten Attributen `x` und `y` zugreifen kann, sind diese beiden Attribute jetzt als `protected` gekennzeichnet. `protected` bedeutet für ein Element:

- Es ist für die abgeleiteten Klassen zugreifbar.
- Es ist für alle anderen Klassen wie bisher nicht zugreifbar.

Der Zugriffsspezifizierer `protected` hat also nur für abgeleitete Klassen eine besondere Wirkung, ansonsten bewirkt er das Gleiche wie `private`.