Ein weiteres Beispiel zur Vererbung

Hintergrund: Sie sollen eine Software für die Personalabteilung eines Unternehmens entwickeln. Zu Beginn machen Sie sich Gedanken über die Datenstrukturen. Das Unternehmen hat drei Typen von Mitarbeitern: Verkäufer, Arbeiter und Manager.

So sollte man es nicht machen:

```
class Verkaeufer {
	final double Basisgehalt = 1800.0;
	private int PersNr;
	private String Name;
	private int Plz;
	private String Ort;
	private String Strasse;
	// Konstruktoren, Zugriffsmethoden
	double berechneGehalt() {
		return 2*Basisgehalt;
		}
	}

class Arbeiter {
	final double Basisgehalt = 1800.0;
	private int PersNr;
	private String Name;
	private int Plz;
	private String Ort;
	private String Strasse;
	// Konstruktoren, Zugriffsmethoden
	double berechneGehalt() {
		return Basisgehalt;
		}
	}

class Manager {
	final double Basisgehalt = 1800.0;
	private int PersNr;
	private String Name;
	private int Plz;
	private String Ort;
	private String Strasse;
	// Konstruktoren, Zugriffsmethoden
	double berechneGehalt() {
		return 8*Basisgehalt;
		}
	}
```

Auffallend ist, dass die Klassen Verkaeufer, Arbeiter und Manager fast den gleichen Inhalt haben. Lediglich die Methode berechneGehalt() unterscheidet sich. In der Softwareentwicklung ist man stets bemüht, den gleichen Code nicht mehrfach zu

implementieren. Der Grund dafür ist einfach: Sollten Sie z.B. irgendwann später feststellen, dass Sie ein weiteres Attribut hinzufügen müssen (z.B. den Geburtstag eines Mitarbeiters), müssen Sie dieses in unserem Beispiel in drei verschiedenen Klassen machen. Wenn die Software komplexer ist, kann es sein, dass Sie an sehr viel mehr Stellen eine Änderung vornehmen müssen. Dies ist aufwendig und fehleranfällig, da es leicht passieren kann, dass Sie an einer Stelle die Änderung vergessen.

So sollte man es machen:

Fassen Sie die gemeinsamen Attribute und Methoden in einer Basisklasse zusammen und vererben Sie anschließend diese Basisklasse an die Klassen Verkaeufer, Arbeiter und Manager. Sollten Sie jetzt zu einem späteren Zeitpunkt feststellen, dass Sie ein weiteres Attribut (Geburtstag) benötigen, müssen Sie nur an einer Stelle (der Basisklasse) etwas ändern.

```
class Mitarbeiter {                          // Basisklasse
        final double Basisgehalt = 1800.0;
        private int PersNr;
        private String Name;
        private int Plz;
        private String Ort;
        private String Strasse;
        // Konstruktoren, Zugriffsmethoden
        }

class Verkaeufer extends Mitarbeiter {
        double berechneGehalt() {
                return 2*Basisgehalt;
                }
        }

class Arbeiter extends Mitarbeiter {
        double berechneGehalt() {
                return Basisgehalt;
                }
        }

class Manager extends Mitarbeiter {
        double berechneGehalt() {
                return 8*Basisgehalt;
                }
        }
```

Konstruktoren

Da Konstruktoren nicht mitvererbt werden, müssen Sie diese für die abgeleiteten Klassen Verkaeufer, Arbeiter und Manager selbst implementieren. Zum Beispiel sähe die Klasse Arbeiter dann folgendermaßen aus:

```
class Arbeiter extends Mitarbeiter {
        Arbeiter() {
                super();
                }

        Arbeiter(int pn, String n, int plz, String o, String s {
                super(pn, n, plz, o, s);
                }

        double berechneGehalt() {
                return Basisgehalt;
                }
        }
```

super ist dabei synonym für den Namen der Basisklasse. Zur Erinnerung: this ist synonym für den Namen der aktuellen Klasse.

private oder protected

Sie können in einer abgeleiteten Klasse auf alles, was in der Basisklasse *private* ist, nicht direkt zugreifen. Im Herdt-Heft wird als Alternative *protected* genannt. Auf ein Attribut der Basisklasse, das anstelle von *private* als *protected* definiert ist, können Sie direkt in den abgeleiteten Klassen zugreifen. Allerdings können Sie auch in allen anderen Klassen im selben Ordner darauf direkt zugreifen. Deshalb empfehle ich, auf *protected* zu verzichten und für Attribute *private* zu verwenden. Der Zugriff in einer abgeleiteten Klasse erfolgt dann, wie üblich, über die Zugriffsmethoden.

Vererbung via Klassenattribute

Wann erbt eine Klasse von einer anderen, wann ist ein Objekt einer Klasse ein Attribut einer anderen Klasse?

Attribut:

```
class Rechteck {
        private Punkt linksOben;
        private int Breite;
        private int Laenge;
        . . .
```

Vererbung:

```
class Arbeiter extends Mitarbeiter {
        . . .
```

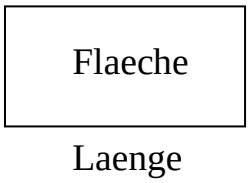
Ist-Beziehung $\Rightarrow$ Vererbung
(Ein Arbeiter ist ein Mitarbeiter)

Hat-Beziehung $\Rightarrow$ Attribut
(Ein Rechteck hat eine linke obere Ecke vom Datentyp Punkt, oder anders gesagt: Ein Rechteck ist kein Punkt)

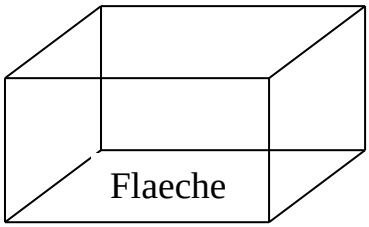
Ein weiteres Indiz, wann eine Hat-Beziehung vorliegt, ist, wenn es mehrere Objekte geben kann. Zum Beispiel kann ein Flugzeug nicht von Triebwerk erben. Ein Flugzeug ist kein Triebwerk, sondern hat eines und kann darüber hinaus auch mehrere Triebwerke haben.

Das Beispiel im Herdt-Heft könnte auf den ersten Blick etwas verwirren. Da erbt ein Quader (Cuboid) von einem Rechteck (Rectangle). Ein Quader hat aber mehrere Flächen vom Datentyp Rechteck. Dies lässt auf eine Hat-Beziehung schließen. Eine andere Möglichkeit der Interpretation wäre: Ein Quader ist ein Rechteck mit einer von Null verschiedenen Höhe.

Rechteck:



Quader:



Rechteck + Höhe

Von daher ist das Beispiel etwas grenzwertig, zeigt aber sehr gut, dass die Dinge in der realen Welt nicht immer eindeutig sind. Soll der Quader also von Rechteck erben oder lieber ein Objekt vom Datentyp Rechteck als Attribut haben? Diese Frage lässt sich nicht eindeutig beantworten. Die Antwort auf diese Frage hängt davon ab, wozu man im weiteren Verlauf der Software den Quader nutzen will. Ist es wichtig, die einzelnen Quaderflächen gezielt anzusprechen, sollte man eher auf eine Hat-Beziehung zurückgreifen. Hat man weitere geometrische Objekte, die auf einem Rechteck basieren (z.B. vom Datentyp Pyramide), und will man auf diesen Objekten Berechnungen anstellen (Berechnung des Volumens, …), ist aber an den Einzelflächen nicht direkt interessiert, käme eher eine Vererbungsbeziehung in Frage. Wir werden diesen Punkt in der übernächsten Woche, wenn wir uns mit Polymorphismus beschäftigen, näher beleuchten.