4
Vererbung, Polymorphie

Jörn Loviscach

Versionsstand: 21. März 2014, 22:57

Die nummerierten Felder sind absichtlich leer, zum Ausfüllen beim Ansehen der Videos:
http://www.j3L7h.de/videos.html

Bitte hier notieren, was beim Bearbeiten unklar geblieben ist

1 Vererbung

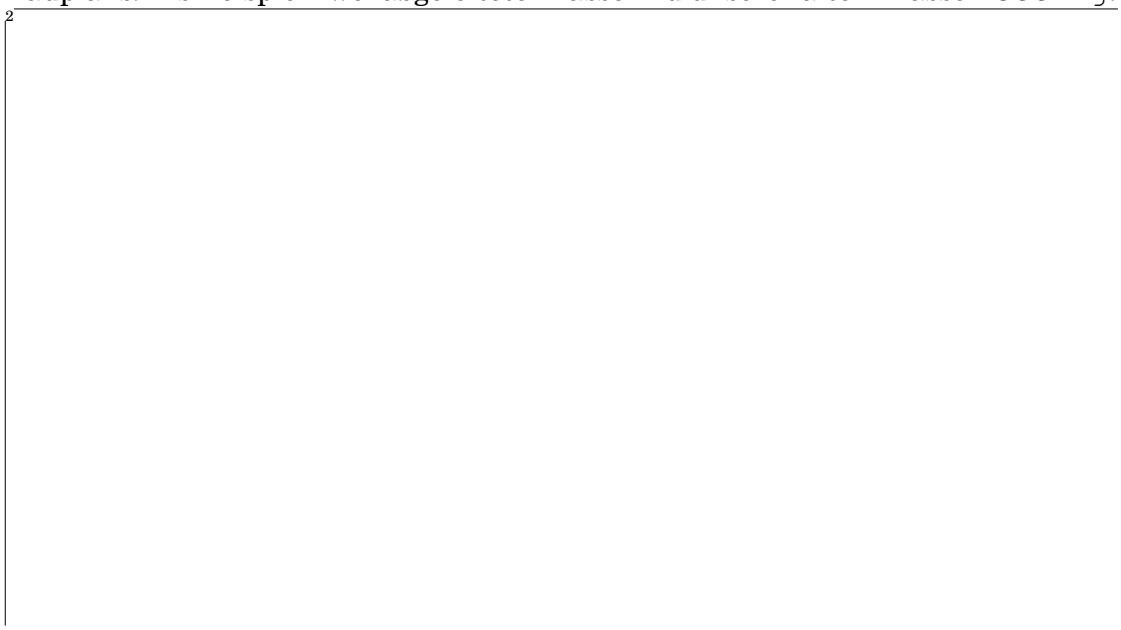
Die Datenkapselung [encapsulation] war das erste große Prinzip der Objektorientierung:

[1]

Klassen sind Baupläne für Objekte – solche Kombinationen aus Daten und Funktionen. Jedes Objekt ist eine Instanz einer Klasse.

Nun kommt das zweite große Prinzip: Vererbung [inheritance]. Eine (Kind-)Klasse [child class] kann von einer anderen (Mutter-)Klasse [parent class, base class] erben [to inherit]; sie heißt dann auch von dieser Klasse abgeleitet [derived]. Die abgeleitete Klasse kann alles Öffentliche, was die Mutterklasse kann – und kann selbst Neues hinzufügen. Die Kindklasse erhält sozusagen eine Kopie des Bauplans. Als Beispiel zwei abgeleitete Klassen zu unserer alten Klasse `Meeting`:

2

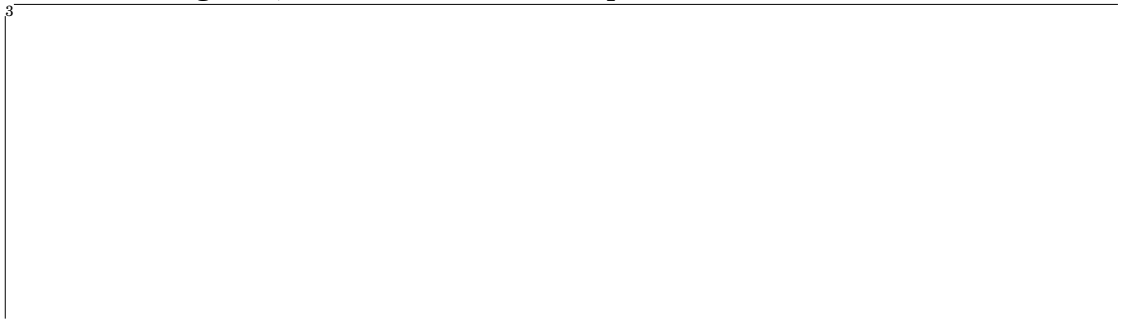


In C++ muss man hinter den Doppelpunkt noch `public` schreiben. In Java schreibt man `extends` statt des Doppelpunkts.

Statische Attribute werden für die Kindklasse *nicht* neu angelegt. Es gibt also zum Beispiel weiterhin nur einen einzigen Zähler für die Anzahl an `Meeting`s – den in der Klasse `Meetings`. Demo im Debugger.

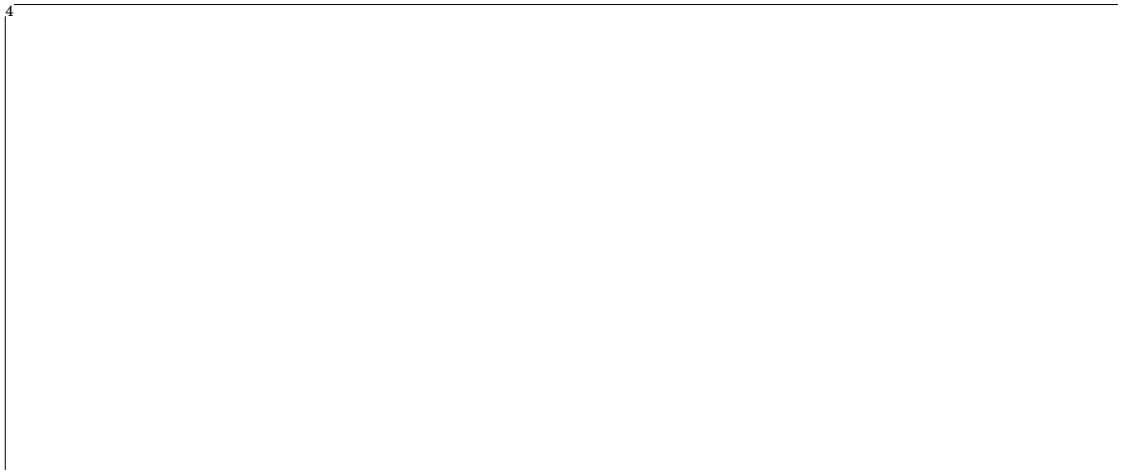
Aus der Kindklasse sind nur die `public`-Elemente der Mutterklasse verfügbar. Man kann aber allen Kind- und Kindeskind-Klassen Zugriff auf Elemente der Mutterklasse geben, wenn man die dort als `protected` deklariert:

3



Microsoft Visual Studio zeigt auf Wunsch ein Klassendiagramm. Dort lassen sich auch per Mausklick Änderungen an den Klassen vornehmen, die dann automatisch in den Quelltext übernommen werden. Das Diagramm in Visual Studio entspricht allerdings nicht genau dem Klassendiagramm nach UML-Standard:

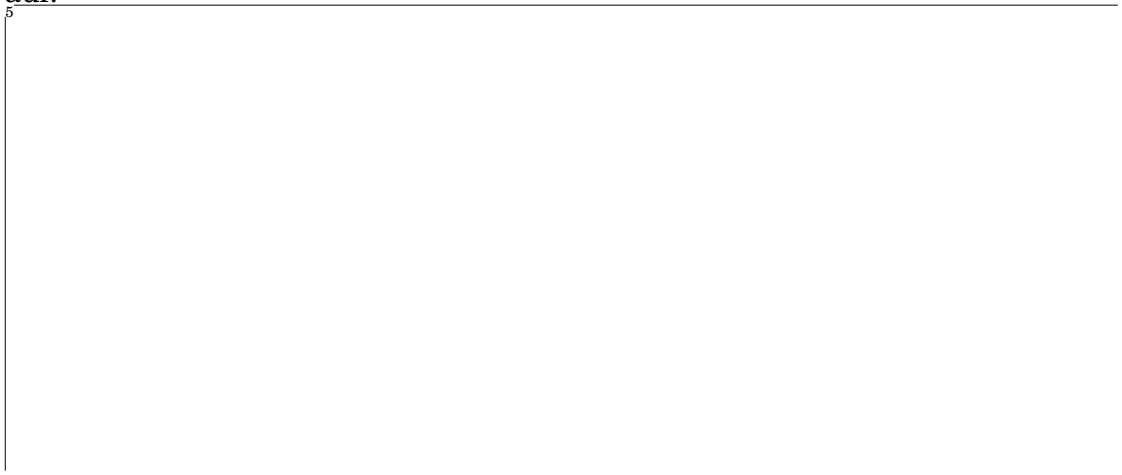
4



2 Konstruktoren für abgeleitete Klassen

Eine abgeleitete Klasse erbt eine wichtige Sache nicht: den Konstruktor. Der Konstruktor muss die neuen Datenelemente dieser Klasse richtig initialisieren – *vorher* muss er aber dafür sorgen, dass die geerbten Datenelemente richtig initialisiert werden. Dazu ruft er den Konstruktor der Mutterklasse [base class] auf:

5



In C++ schreibt man nicht `base`, sondern den Namen der Mutterklasse. In Java schreibt man `super(...)` in die erste Zeile nach der öffnenden Schweifklammer.

Schreibt man nicht ausdrücklich den Konstruktor der Basisklasse hin, sucht der Compiler nach einem parameterlosen Konstruktor der Basisklasse. Wenn es den nicht gibt, ist das ein Fehler.

Schreibt man gar keinen eigenen Konstruktor für die abgeleitete Klasse `BusinessMeeting`, ergänzt der Compiler einen parameterlosen Konstruktor („Standardkonstruktor“), so dass der Aufruf `new BusinessMeeting()` funktioniert – wenn die Mutterklasse `Meeting` einen parameterlosen Konstruktor besitzt. (Warum diese Einschränkung?)

3 Polymorphie, virtuelle Methoden, Überschreiben

Ein anderes Konzept ist mit dem Konzept der Vererbung verbunden: Polymorphie, übersetzbar als „Vielgestaltigkeit“. Eine Instanz einer Kindklasse oder Kindeskindklasse kann eine Instanz ihrer Mutterklasse vertreten – und sich dabei anderes verhalten. Man kann dazu in der Kindklasse Methoden der Mutterklasse überschreiben [to override, *nicht* to overwrite].

Um das in C# zu erlauben, muss die entsprechende Methode in der Mutterklasse als `virtual` und in der Kindklasse als `override` gekennzeichnet sein:

[6]

C# ist hier sicherheitshalber sehr wortreich: In C++ erübrigt sich das `override`, in Java obendrein das `virtual`.

Visual Studio hilft beim Überschreiben, wenn man direkt `override` und dann ein Leerzeichen eintippt. Standardmäßig entsteht dann eine Methode, die mit `base` einfach die Originalmethode der Mutterklasse aufruft.

Es gibt einige Randbedingungen beim Überschreiben. In C# sind das diese:

[7]

Auch eine Property von C# kann virtuell sein und überschrieben werden:

[8]

4 Anwendungen der Polymorphie. Upcast, Downcast

Überall, wo eine Instanz der Mutterklasse steht, kann eine Instanz einer abgeleiteten Klassen verwendet werden. Etwa lässt sich eine `BusinessMeeting`-Instanz an eine `Meeting`-Variable zuweisen (von der Kindklasse zur Mutterklasse wandeln: „Upcast“; das geht immer). Für überschriebene virtuelle Methoden und Properties wird der Code der jeweiligen Kindklasse aufgerufen:

[9]

Insbesondere macht Polymorphie so in den meisten Fällen auf elegante Art ein `switch ... case` überflüssig.

Jeder Code, den man für `Meeting` geschrieben hat, wird sofort auch mit den Kindklassen funktionieren! Man beachte: Den Code für `Meeting` könnte jemand Anderes vor fünf Jahren geschrieben haben. In Aktion sehen wir das schon bei unserer Hauptfensterklasse: Die ist von der Klasse `Window` der Bibliothek abgeleitet und erbt deshalb einen riesigen Satz an fertig gebauten Methoden und Properties. Dies ist ein massives Recycling von Code („Erben der Implementierung“).

In C# und Java erbt jede Klasse automatisch und unabänderlich von der Klasse `object`. So hat jedes Objekt zum Beispiel eine virtuelle Methode `ToString`. Wann immer die Klassenbibliothek eine Zeichenkette für ein Objekt benötigt, ruft

sie diese Methode auf. Beispiele: `"abc"+d`; das Hinzufügen eines Eintrags zu einer `Listbox`; die Anzeige im Debugger. Durch Überschreiben von `ToString` kann man steuern, wie sich die Objekte einer Klasse in Textform präsentieren:

10

Hinter eine Variablen vom Typ `Meeting` könnte sich auch eine Instanz einer der Kindklassen `BusinessMeeting` oder `Dinner` verbergen. Die `Meeting`-Variable kennt aber nicht die neuen Attribute, Methoden und Properties der Kindklassen. Um auf die zuzugreifen, muss man den Typ wandeln:

11

Von der Mutterklasse zur Kindklasse wandeln: „Downcast“; das geht nur dann, wenn das Objekt auch tatsächlich eine Instanz der gwünschten Kindklasse ist – oder eine Instanz einer davon abgeleiteten Klasse. In C# kann man mit `is` abfragen, ob der Downcast funktionieren würde:

12

In Java heißt das `instanceof`. In C++ ist die Abfrage, zu welcher Klasse ein Objekt gehört, ein heikles Thema (RTTI, Run-Time Type Identification).

5 Zusammenfassung: Klassen