8.1.3 Überlagern von Methoden

Neben den Membervariablen erbt eine abgeleitete Klasse auch die Methoden ihrer Vaterklasse (wenn dies nicht durch spezielle Attribute verhindert wird). Daneben dürfen auch neue Methoden definiert werden. Die Klasse besitzt dann alle Methoden, die aus der Vaterklasse geerbt wurden, und zusätzlich die, die sie selbst neu definiert hat.

Daneben dürfen auch bereits von der Vaterklasse geerbte Methoden neu definiert werden. In diesem Fall spricht man von Überlagerung der Methode. Wurde eine Methode überlagert, wird beim Aufruf der Methode auf Objekten dieses Typs immer die überlagernde Version verwendet.

Das folgende Beispiel erweitert die Klasse ZweisitzerCabrio um die Methode alter, das nun in Monaten ausgegeben werden soll:

```
001 class ZweisitzerCabrio
002 extends Cabrio
003 {
004   boolean notsitze;
005
006   public int alter()
007   {
008     return 12 * (2000 - erstzulassung);
009   }
010 }
```

Listing 8.5: Überlagern einer Methode in einer abgeleiteten Klasse

Da die Methode alter bereits aus der Klasse Cabrio geerbt wurde, die sie ihrerseits von Auto geerbt hat, handelt es sich um eine Überlagerung. Zukünftig würde dadurch in allen Objekten vom Typ ZweisitzerCabrio bei Aufruf von alter die überlagernde Version, bei allen Objekten des Typs Auto oder Cabrio aber die ursprüngliche Version verwendet werden. Es wird immer die Variante aufgerufen, die dem aktuellen Objekt beim Zurückverfolgen der Vererbungslinie am nächsten liegt.

Dynamische Methodensuche

Nicht immer kann bereits der Compiler entscheiden, welche Variante einer überlagerten Methode er aufrufen soll. In Abschnitt 4.6 und Abschnitt 7.1.6 wurde bereits erwähnt, daß das Objekt einer abgeleiteten Klasse zuweisungskompatibel zu der Variablen einer übergeordneten Klasse ist. Wir dürfen also beispielsweise ein Cabrio-Objekt ohne weiteres einer Variablen vom Typ Auto zuweisen.

Die Variable vom Typ Auto kann während ihrer Lebensdauer also Objekte verschiedenen Typs enthalten (insbesondere solche vom Typ Auto, Cabrio und ZweisitzerCabrio). Damit kann natürlich nicht schon zur Compile-Zeit entschieden werden, welche Version einer überlagerten Methode aufgerufen werden soll. Erst während das Programm läuft, ergibt sich, welcher Typ von Objekt zu einem bestimmten Zeitpunkt in der Variable gespeichert wird. Der Compiler muß also Code generieren, um dies zur Laufzeit zu entscheiden. Man bezeichnet dies auch als dynamisches Binden.

In C++ wird dieses Verhalten durch virtuelle Funktionen realisiert und muß mit Hilfe des Schlüsselworts virtual explizit angeordnet werden. In Java ist eine explizite Deklaration nicht nötig, denn Methodenaufrufe werden immer dynamisch interpretiert. Der dadurch verursachte Overhead ist allerdings nicht zu vernachlässigen und liegt deutlich über den Kosten eines statischen Methodenaufrufs. Um das Problem zu umgehen, gibt es mehrere Möglichkeiten, dafür zu sorgen, daß eine Methode nicht dynamisch interpretiert wird. Dabei wird mit Hilfe zusätzlicher Attribute dafür gesorgt, daß die betreffende Methode nicht überlagert werden kann:

- Methoden vom Typ `private` sind in abgeleiteten Klassen nicht sichtbar und können daher nicht überlagert werden.
- Bei Methoden vom Typ `final` deklariert der Anwender explizit, daß sie nicht überlagert werden sollen.
- Auch bei `static`-Methoden, die ja unabhängig von einer Instanz existieren, besteht das Problem nicht.

Hinweis

In Abschnitt 8.4 werden wir das Thema *Polymorphismus* noch einmal aufgreifen und ein ausführliches Beispiel für dynamische Methodensuche geben.

Aufrufen einer verdeckten Methode

Wird eine Methode x in einer abgeleiteten Klasse überlagert, wird die ursprüngliche Methode x verdeckt. Aufrufe von x beziehen sich immer auf die überlagernde Variante. Oftmals ist es allerdings nützlich, die verdeckte Superklassenmethode aufrufen zu können, beispielsweise, wenn deren Funktionalität nur leicht verändert werden soll. In diesem Fall kann mit Hilfe des Ausdrucks super.x() die Methode der Vaterklasse aufgerufen werden. Der kaskadierte Aufruf von Superklassenmethoden (wie in super.super.x()) ist nicht erlaubt.

8.1.4 Vererbung von Konstruktoren

Konstruktorenverkettung

Wenn eine Klasse instanziert wird, garantiert Java, daß ein zur Parametrisierung des new-Operators passender Konstruktor aufgerufen wird. Daneben garantiert der Compiler, daß auch der Konstruktor der Vaterklasse aufgerufen wird. Dieser Aufruf kann entweder explizit oder implizit geschehen.

Falls als erste Anweisung innerhalb eines Konstruktors ein Aufruf der Methode super steht, wird dies als Aufruf des Superklassenkonstruktors interpretiert. super wird wie eine normale Methode verwendet und kann mit oder ohne Parameter aufgerufen werden. Der Aufruf muß natürlich zu einem in der Superklasse definierten Konstruktor passen.

Falls als erste Anweisung im Konstruktor kein Aufruf von super steht, setzt der Compiler an dieser Stelle einen impliziten Aufruf super(); ein und ruft damit den parameterlosen Konstruktor der Vaterklasse auf. Falls ein solcher Konstruktor in der Vaterklasse nicht definiert wurde, gibt es einen Compiler-Fehler. Das ist genau dann der Fall, wenn in der Superklassendeklaration lediglich parametrisierte Konstruktoren angegeben wurden und daher ein parameterloser default-Konstruktor nicht automatisch erzeugt wurde.

Hinweis

Alternativ zu diesen beiden Varianten, einen Superklassenkonstruktor aufzurufen, ist es auch erlaubt, mit Hilfe der `this`-Methode einen anderen Konstruktor der eigenen Klasse aufzurufen. Um die oben erwähnten Zusagen einzuhalten, muß dieser allerdings selbst direkt oder indirekt schließlich einen Superklassenkonstruktor aufrufen.

Der Default-Konstruktor

Das Anlegen von Konstruktoren in einer Klasse ist optional. Falls in einer Klasse überhaupt kein Konstruktor definiert wurde, erzeugt der Compiler beim Übersetzen der Klasse automatisch einen parameterlosen default-Konstruktor. Dieser enthält lediglich einen Aufruf des parameterlosen Superklassenkonstruktors.

Überlagerte Konstruktoren

Hinweis

Konstruktoren werden nicht vererbt. Alle Konstruktoren, die in einer abgeleiteten Klasse benötigt werden, müssen neu definiert werden, selbst wenn sie nur aus einem Aufruf des Superklassenkonstruktors bestehen.

Durch diese Regel wird bei jedem Neuanlegen eines Objekts eine ganze Kette von Konstruktoren aufgerufen. Da nach den obigen Regeln jeder Konstruktor zuerst den Superklassenkonstruktor aufruft, wird die Initialisierung von oben nach unten in der Vererbungshierarchie durchgeführt: zuerst wird der Konstruktor der Klasse Object ausgeführt, dann der der ersten Unterklasse usw., bis zuletzt der Konstruktor der zu instanzierenden Klasse ausgeführt wird.

Destruktorenverkettung

Hinweis

Im Gegensatz zu den Konstruktoren werden die Destruktoren eines Ableitungszweiges nicht automatisch verkettet. Falls eine Destruktorenverkettung erforderlich ist, kann sie durch explizite Aufrufe des Superklassendestruktors mit Hilfe der Anweisung `super.finalize()` durchgeführt werden.