Studienarbeit

Konzeption und Umsetzung eines Fachwertkonzeptes

Klaus Müller
Löh 10
27446 Deinstedt
email: 4kmuelle@informatik.uni-hamburg.de

6. Juli 1999

Betreuer: **Prof. Dr.-Ing. Heinz Züllighoven**
Dipl.-Inform. Wolf-Gideon Bleek

Fachbereich Informatik
Arbeitsbereich Softwaretechnik
Unversität Hamburg
Vogt-Kölln-Str. 30
22527 Hamburg

Inhaltsverzeichnis

1. Einführung

1.1. Motivation

Wenn man als Softwareentwickler einen Anwendungskontext für die Entwicklung eines Softwaresystems auf seine zu modellierenden Bestandteile untersucht, wird man sehr schnell feststellen, daß es hiervon zwei unterschiedliche Arten gibt. Zum einen sind dies anwendungsnahe Gegenstände und Konzepte wie Formulare oder Ablagen. Diese lassen sich zusammen mit ihren Umgangsformen bei der Modellierung gut als Objekte abbilden. Daneben gibt es aber noch abstrakte Größen, die zum Abzählen, Ordnen oder zur Identifikation benötigt werden. Solche abstrakten Größen lassen sich gut durch Werte darstellen. Im Bankenkontext wäre hier zum Beispiel eine Kontonummer oder ein Geldbetrag zu nennen. Der Umgang mit diesen Größen wird aber nicht durch die oben genannten mathematischen Operationen geprägt, sondern durch ihre Verwendung in dem jeweiligen Anwendungskontext. Aus diesem Grund werden sie im Folgenden als fachliche Werte bezeichnet. Das Problem bei der Umsetzung von fachlichen Werten ist jedoch, wie diese in dem zu erstellenden Softwaresystem modelliert werden sollen. Zwar ist es möglich, diese durch die Standarddatentypen der jeweiligen Programmiersprache darzustellen, doch in diesem Fall trennt man den Wert von seinen fachlichen Umgangsformen. Will man die Einheit von Werten und fachlichen Operationen erhalten, ist dies in objektorientierten Programmiersprachen nur dadurch möglich, daß alle fachlichen Werte als abstrakte Datentypen [Meyer 88] über Objekte modelliert werden. Es hat sich aber gezeigt, daß bei dieser Art der Modellierung ein wertähnliches Verhalten nur durch einen erheblichen Verlust an Performanz und einen unverhältnismäßig hohen Verbrauch an Speicherplatz umzusetzen ist. Als Lösung dieser Problematik wurde in verschiedenen Artikeln und Büchern (z.B. in „*Values in Object Systems*“ [Bäumer et al. 98], „*Values and Objects Revisited*“ [van der Werf 98] oder „*Das objektorientierte Konstruktionshandbuch*“ [Züllighoven et al. 98]) vorgeschlagen, das Spektrum der objektorientierten Programmierung um ein Konzept für Fachwerte zu erweitern.

Werte und somit auch Fachwerte bieten von ihrer Verwendung her in vielen Fällen einige Vorteile gegenüber den Objekten. Der offensichtliche Vorteil ist natürlich, daß Fachwerte eine praktikable Antwort auf das Problem sind, benutzerdefinierte Werte in der Objektorientierung darzustellen ohne auf elementare Datentypen zurückgreifen zu müssen. So wird zum Beispiel das Konzept einer Kontonummer mit all seiner Funktionalität auch im Programm realisiert, und es muß kein Integer als Hilfskonstrukt benutzt werden, bei dem auch nur durch Programmierkonventionen eine fachlich korrekte Handhabung gewährleistet werden kann.

Ein weiterer entscheidender Vorteil gegenüber den Objekten ist, daß Werte eine vollständige referenzielle Transparenz bieten, d.h. sie haben keine Seiteneffekte. Wenn man fachliche Werte einfach durch Objekte modellieren würde, hätte man das Problem, daß sich diese Objekte ändern könnten, ohne daß andere Objekte, die diese referenzieren, von dieser Änderung Kenntnis nähmen. Es müßte in diesem Fall ein Benachrichtigungsmechanismus gebaut werden, über welchen Änderungen mitgeteilt werden können. Bei einer Verwendung von Werten entfällt dieses Problem, da Werte nicht verändert werden können. Es müssen also keine Ressourcen darauf verwendet werden, Veränderungen

mitzuteilen, da dieser Fall bei der Verwendung von Werten nicht auftreten kann. Ein weiterer Vorteil von Werten ist, daß sie anders als Objekte keine Zustände haben und völlig zustandsunabhängig sind. Darüber hinaus können Werte nicht erzeugt oder gelöscht werden. Es können nur die verschiedenen möglichen Repräsentationen eines Wertes interpretiert werden. Weiterhin können Werte nach Wertsemantik verwendet werden, und es sind auf ihnen nur fachlich vorher definierte Operationen möglich. Natürlich gibt es einige Funktionen, die immer vorhanden sein müssen, wie zum Beispiel der Test auf Gleichheit zweier Fachwerte oder Funktionen, die überprüfen, ob es sich bei einer gegebenen Darstellung eines Fachwertes überhaupt um einen gültigen Wert handelt. Bei dieser Prüfung darf allerdings kein Wissen über den Einsatzkontext verwendet werden, da dies das Fachwertkonzept verletzen würde. Andere Operationen müssen den jeweiligen Erfordernissen angepaßt werden. So werden Geldbeträge anders addiert als einfache Dezimalzahlen, da zuerst verglichen werden muß, ob es sich bei beiden Beträgen um die gleiche Währung handelt oder ob zuerst eine Umrechnung erfolgen muß.
Aus den oben genannten Punkten lassen sich einige wichtige Fragestellungen für die Studienarbeit und den damit verbundenen Entwurf eines Fachwertkonzeptes und dessen Umsetzung in das JWAM-Rahmenwerk des Arbeitsbereiches Softwaretechnik ableiten. Zunächst einmal stellt sich natürlich die Frage, wo es sinnvoll ist, Fachwerte anstelle von Objekten zu verwenden. Dabei sollte es klar sein, daß nicht alle Artefakte und Konzepte einer Anwendungsdomäne nur noch mit Werten modelliert werden sollten. Dadurch würde man auf die durch die Objektorientierung gewonnenen Möglichkeiten zur Modellierung und Umsetzung von Artefakten des Anwendungskontextes verzichten. Diese Fragestellung wird im Abschnitt 2.1. näher betrachtet.
Wenn man allerdings die Frage nach einer sinnvollen Verwendung von Fachwerten beantworten will, muß man vorher genau klären, wo die Unterschiede in der Konzeption und der Verwendung von Fachwerten und Objekten liegen. Hierbei stellt sich nun die Frage, wie sich das Fachwertkonzept mit den bestehenden Implementationsmustern aus der Softwaretechnik umsetzen und in die bestehenden Rahmenwerke einfügen läßt, oder ob dafür neue Entwurfsmuster notwendig sind. Weiter muß geklärt werden, welche Eigenschaften von Fachwerten überhaupt zwingend umgesetzt werden müssen, um ein nach Wertsemantik korrektes Verhalten zu garantieren. Daneben ergibt sich aus den oben genannten Punkten noch eine andere für die Verwendung von Fachwerten wichtige Frage: Wie wird ein Fachwert geeignet repräsentiert? Diese Frage mag auf den ersten Blick trivial erscheinen, aber Werte existieren in ihrem eigenen „Werte-Universum“ und manifestieren sich nur über ihre Repräsentationen. Reicht hierbei eine einfache Repräsentation als Zeichenkette aus, oder müssen noch weitere Formen der Darstellung gefunden werden? Auch wenn eine Repräsentation als Zeichenkette ausreicht, gibt es doch viele unterschiedliche Möglichkeiten, sich mit verschiedenen Darstellungen auf denselben Wert zu beziehen. Zum Beispiel wird mit den Ausdrücken „42“ und „40 + 2“ der gleiche Wert <42> repräsentiert. Ein anderes Beispiel ist ein Geldbetrag auf einem Scheck. In diesem Fall wird der Wert einmal als Dezimalzahl und einmal als Wort dargestellt. Ähnliche Beispiele lassen sich in jeder Anwendungsdomäne finden. Das Problem hierbei ist, aus allen möglichen Darstellungsformen die für eine Modellierung notwendigen herauszusuchen, da sich mit Sicherheit nicht alle möglichen Darstellungsformen in einem Anwendungssystem implementieren lassen. Dies ist allerdings eine Frage, die sich nicht allgemeingültig beantworten lassen wird, sondern nur in einem konkreten Problemfeld.

An diese mehr auf das Konzept der Fachwerte bezogenen Fragen, schließen sich noch andere Fragen an, die mehr auf die programmiertechnische Seite einer Implementation abzielen. Zuerst muß hier natürlich die Frage gestellt werden, wie die Fachwertobjekte im System modelliert werden sollen. Schon allein die Bezeichnung Fachwertobjekte beinhaltet ein grundlegendes Problem, nämlich daß man einen Wert als Objekt modellieren muß, da in den objektorientierten Sprachen wie Java oder C++ gar keine andere Möglichkeit besteht, eine solche Datenstruktur mit den zugehörigen fachlichen Umgangsformen abzubilden.
Weiterhin ist es sicherlich keine sehr effiziente Lösung, für jeden Wert ein Fachwertobjekt zu erzeugen. Es ist klar, daß man bei einer solchen Art der Modellierung eine erhebliche Redundanz in Kauf nimmt, was sicherlich merkbare Auswirkungen auf die Performanz und den Speicherbedarf eines solchen Systems hat. Wenn man diese Redundanz vermeiden möchte, muß man gleiche Werte zusammenfassen und zulassen, daß der entstandene Wert von mehreren Objekten verwendet wird. Hierbei muß allerdings garantiert werden, daß die Werte nicht verändert werden können. Dadurch können bei der Verwendung von Fachwerten keine Seiteneffekte auftreten. Für die Verwaltung muß es eine Instanz geben, die kontrolliert, ob ein Wert schon vorhanden ist oder ob ein neues Fachwertobjekt erzeugt werden muß. Grundsätzlich gibt es dafür zwei verschiedene Ansätze: Die Modellierung als unveränderliche oder veränderliche Objekte. In der Softwaretechnik gibt es für diese Lösungsansätze auch entsprechende Entwurfsmuster wie etwa das „flyweight/factory-pattern“ [Gamma et al. 96] oder das „body/handle-pattern“ [Coplien 92], welche hier auf ihre Eigenschaften und ihre Eignung für das JWAM-Rahmenwerk untersucht werden sollen.
Auch in Hinblick auf Persistenz, Serialisierung und Verteilung müssen einige Fragen beantwortet werden. Bei Persistenz und Serialisierung steht natürlich die Frage nach einer platzsparenden Abbildung auf Speichermedien im Vordergrund. Wenn man Fachwerte in einem verteilten System einsetzen will, so muß man auch dort ein korrektes Verhalten garantieren können.
Zusammenfassend lassen sich drei übergeordnete Fragestellungen erkennen:

- Welche Vorteile lassen sich aus der Verwendung von Fachwerten gegenüber Objekten ziehen ?
- Wie lassen sich Fachwerte geeignet implementieren?
- Welche Grundfunktionalitäten und Eigenschaften sind für Fachwerte unbedingt notwendig?

1.2. Zielsetzung

Ziel der Studienarbeit war es, ein Rahmenwerk zur Umsetzung des Fachwertkonzeptes zu entwerfen und in das JWAM-Rahmenwerk zu integrieren sowie die dabei gewonnenen Erkenntnisse auszuwerten. Hierzu war es nötig, aus der Menge der zur Verfügung stehenden Entwurfsmuster diejenigen auszuwählen, die für eine Implementation geeignet erschienen. Die Auswahl erfolgte hierbei nach ihrer konzeptionellen und ihrer praktischen Eignung. Anschließend wurden die mit Hilfe der Entwurfsmuster entwickelten Komponenten zu einem Fachwert-Rahmenwerk zusammengefügt. Die ausgewählten Entwurfsmuster sowie das entstandene Rahmenwerk wurden anschließend noch an verschiedenen anwendungsfachlichen Beispielen auf ihre Tauglichkeit geprüft (siehe Abschnitt 1.5.). Desweiteren wurden die Entwurfentscheidungen des

Rahmenwerkes erläutert und die Ergebnisse der Tests dokumentiert. Im Anschluß sollte das Fachwert-Rahmenwerk in das JWAM-Rahmenwerk integriert werden.

1.3. Vorgehensweise

Bei der Erstellung eines geeigneten fachlichen Entwurfs und dessen Umsetzung wechselten sich die theoretisch-konzeptionelle Arbeit und der praxisnahe Test der gewonnenen Erkenntnisse turnusmäßig ab. Das heißt konkret, daß alle bei der theoretischen Arbeit erlangten Kenntnisse anhand von einfachen Fallbeispielen aus verschiedenen Anwendungsbereichen auf ihre Anwendbarkeit getestet wurden. Die dort gewonnenen Erkenntnisse wurden dann wiederum in einem theoretischen Teil aufgearbeitet und wurden anschließend wieder auf ihre Tauglichkeit in den verschiedenen Anwendungsbereichen untersucht. Bei dieser Mischung aus theoretischen und empirischen Vorgehensweisen fand darüber hinaus noch eine regelmäßige Rückkopplung mit den betreuenden Mitarbeitern bzw. den zuständigen Arbeitsgruppen statt, um über fachliche Diskussionen Schwächen oder Unklarheiten im konzeptionellen Entwurf oder im Vorgehen zu erarbeiten. Alle während der Studienarbeit erstellten Dokumente wurden hiernach in mehreren Autor-Kritiker-Zyklen überarbeitet und diskutiert.
Für die weitere Arbeit an der Studienarbeit bedeutete dies konkret, daß zuerst mit Hilfe von geeigneter Fachliteratur Informationen über Fachwerte und Wertsemantik zusammengetragen wurden. Die Grundlagen zu der behandelten Thematik stammen aus dem Artikel „*Objects and Values in Programming Languages*“ [MacLennan 82]. Die Quellen zum Thema der Fachwerte waren die Artikel von Bäumer et al. „*Values in Object Systems*“ [Bäumer et al. 98] und van der Werf „*Values and Objects Revisited*“ [van der Werf 98] sowie „*Das objektorientierte Konstruktionshandbuch*“ [Züllighoven et al. 98]. Hierauf aufbauend wurde untersucht, welche Entwurfsmuster es überhaupt gibt und welche sich davon am besten für eine Implementation eignen. Diese Entwurfsmuster wurden aus den „*Design Patterns - Elements of Reusable Object-Oriented Software“* [Gamma et al. 96] und „*Ruminations on C++“* [Koenig&Moo 97] entnommen. Die ausgewählten Entwurfsmuster wurden dann anhand von einfachen Beispielen aus dem Banken- und dem Verwaltungsbereich implementiert und untersucht. Anschließend wurden sie, wie oben beschrieben, überarbeitet und reimplementiert, um wiederum getestet zu werden. Dieser Zyklus wiederholte sich solange, bis ein für eine endgültige Umsetzung geeignetes Entwurfsmuster gefunden wurde.
Im ersten Abschnitt der Arbeit überwog natürlich der theoretische Anteil der Arbeit, da hier die benötigten Grundkenntnisse gesammelt werden mußten. In den darauffolgenden Phasen rückte dann ein immer mehr empirisch geprägtes Vorgehen mit Implementation und anschließenden Tests in den Vordergrund. Das Ziel hierbei war es, eine anwendbare Implementation für das JWAM-Rahmenwerk zu schaffen, die sich konzeptionell an den Entwurfsrichtlinien des Werkzeug-Automat-Material-Ansatzes [Züllighoven et al. 98] orientiert. In einer abschließenden Phase wurden die Ergebnisse zu einer Studienarbeit zusammengefaßt und kritisch unter theoretischen und praktischen Gesichtspunkten bewertet.

1.4. Anwendungsfachliche Beispiele

Um das gewählte Entwurfsmuster besser auf seine Anwendbarkeit prüfen zu können, wurde im Rahmen der Studienarbeit auf konkrete Anwendungsbeispiele aus verschiedenen Bereichen Bezug genommen.
Da der Arbeitsbereich viele Projekte aus dem Bankenbereich betreut hat bzw. noch betreut, bot es sich an, ein Beispiel aus diesem Bereich zu verwenden, da hier ein ausreichendes Fachwissen schon vorhanden war, wodurch die Notwendigkeit der Einarbeitung in ein fremdes Themengebiet entfiel.
Als weiteres anwendungsfachliches Beispiel sollte das Fachwertkonzept an einem Beispiel aus dem Verwaltungskontext erprobt werden. Bei diesem Beispiel wurde versucht, einen Laufzettel, welcher dazu dient, Abläufe innerhalb der Verwaltung des Fachbereichs Informatik der Universität Hamburg zu koordinieren bzw. zu steuern, samt seinen Umgangsformen zu modellieren. Anschließend wurde das gewonnene Modell mit Hilfe des Werkzeug-Automat-Material-Ansatzes [Züllighoven et al. 98] implementiert. Auch aus diesem Anwendungsbeispiel wurden einigen Fachwerte, welche bisher mit einfachen Objekten oder elementaren Datentypen umgesetzt worden waren, für eine Testimplementation ausgewählt.
Die oben genannten Beispiele stellen natürlich nur eine sehr begrenzte Auswahl der möglichen Fachwerte dar, allerdings war es anhand von ihnen möglich, Erfahrungen mit Fachwerten und den dazugehörigen Entwurfsmustern zu sammeln. Im Folgenden werden allerdings nur einige der oben genannten als Beispiele gezeigt.

1.5. Zum Inhalt der Studienarbeit

Im Kapitel 2 der Studienarbeit soll zunächst diskutiert werden, welche Eigenschaften Werte bzw. Fachwerte ausmachen und welche Unterschiede zu Objekten bestehen. Darüber hinaus soll hier gezeigt werden, wo bei der Modellierung die Trennlinie zwischen Objekten und Werten zu ziehen ist. Im Kapitel 3 wird dann auf die verschiedenen Entwurfsmuster und die konkrete Implementation für das Rahmenwerk eingegangen. Das Kapitel 4 wird sich in der Hauptsache mit der Bewertung des entstandenen Fachwertkonzeptes befassen. Daneben soll dort ein Ausblick auf mögliche Erweiterungsschritte für das Fachwertkonzept gegeben werden.

2. Fachwerte

In diesem Kapitel geht es zunächst darum, die Unterschiede zwischen Werten und Objekten herauszustellen. Dabei soll erläutert werden, welche Vorteile bzw. Nachteile Werte und damit auch Fachwerte gegenüber Objekten haben. Hierauf aufbauend soll dann definiert werden, was konzeptionell ein Fachwert ist. Auf diese Weise sollen vor allem die wesentlichen Eigenschaften von Fachwerten hervorgehoben werden. Weiterhin soll erläutert werden, wo in der Modellierung die Grenzen zwischen Objekten und Fachwerten zu finden sind.

2.1. Werte und Objekte

Werte und Objekte stellen zwei der grundlegenden Konzepte der Informatik dar, die für die modernen Programmiersprachen sowie für die damit entwickelten Anwendungssysteme unerläßlich sind. Im Folgenden soll ein kurzer Überblick über die Eigenschaften von Werten und Objekten gegeben werden.
Bei der Definition der Schlüsselattribute von Werten bzw. Objekten soll sich in dieser Arbeit an den Prinzipien orientiert werden, die MacLennan in seinem Artikel „*Objects and Values in Programming Languages*“ [MacLennan 82] beschrieben hat und die unter anderen auch in dem Artikel „*Values in Object Systems*“ [Bäumer et al. 98] bzw. in den entsprechenden Kapiteln aus dem Buch „*Das objektorientierte Konstruktionshandbuch nach dem Werkzeug & Material - Ansatz*“ [Züllighoven et al. 98] aufgegriffen wurden. Hiernach haben Werte bzw. Objekte folgende Schlüsseleigenschaften.

2.1.1. Werte

Werte sind zeit- und ortslos, d.h. Begriffe wie Zeit und Ort sind auf sie nicht anwendbar, womit sie auch keinen Lebenszyklus besitzen. Ferner können sie weder hergestellt noch verbraucht werden. Sie existieren also in ihrem eigenen „Werte-Universum“, auf das von außen kein Zugriff besteht.
Werte abstrahieren von ihrem Kontext, d. h. sie sind nicht an die Existenz eines konkreten Gegenstandes in einer Anwendungsdomäne gebunden. Gleichzeitig verfügen sie über keine Identität, sondern es gibt nur einen einzigen Wert, auf den man sich beziehen kann. Deshalb kann es auch nicht mehrere Exemplare eines Wertes geben, sondern es handelt sich dann immer nur um verschiedene Repräsentationen desselben Wertes.
Werte sind unveränderlich, was bedeutet, daß zwar sie berechnet bzw. aufeinander bezogen werden können, der Wert an sich aber immer der gleiche bleibt. Folglich haben sie keinerlei Seiteneffekte.
Werte sind referentiell transparent, was bedeutet, daß Ausdrücke auf Basis von Werten jederzeit aufgrund ihrer Teilausdrücke verstanden werden können. So repräsentieren sowohl „40 + 2“, „2 + 40“ als auch „42“ den eigentlichen Wert „42“. Der Wert eines solchen Ausdrucks hängt also nur von seinen Argumenten ab und ist unabhängig von der Reihenfolge immer gleich. Die verschiedenen Teilausdrücke könnten also einfach durch einander ersetzt werden, ohne daß sich das Ergebnis ändern würde. Allerdings kann ein Wert, auf den sich ein Bezeichner in einem Kontext bezieht, undefiniert sein. Daneben sollte immer bedacht werden, daß Werte verschiedene Repräsentationen besitzen können.

Werte können nicht gemeinsam genutzt werden, da sie weder eine Identität noch einen Zustand haben. Es kann folglich keine Kommunikation anhand von Werten vorgenommen werden, da Werte nicht ausgetauscht bzw. bearbeitet werden können. Um aber dennoch mit Werten kommunizieren zu können, muß ein Wert an den Einsatzkontext gebunden werden, was bedeutet, daß ein Wert über eine eindeutige Identifikationsmöglichkeit in einen Kontext eingeordnet wird. Hiernach ist es nur sinnvoll, über einen Wert „50 DM“ zu reden, wenn dieser zum Beispiel einer Banknote zugeordnet wird. Ohne diese Zuordnung kann mit diesem Wert nicht gearbeitet werden.

2.1.2. Objekte

Objekte haben im Gegensatz zu Werten einen Lebenszyklus. Dies bedeutet, daß sie sich zu einem bestimmten Zeitpunkt am einem bestimmten Ort befinden.
Desweiteren verfügen Objekte über eine Identität. Es kann also beliebig viele Exemplare eines allgemeinen Konzeptes geben, welche sich allerdings voneinander in ihrer Identität unterscheiden.
Objekte können in ihrem Lebenszyklus von ihren Klienten verändert werden, was bedeutet, daß sie verschiedene interne Zustände annehmen können.
Objekte können über Referenzen von mehreren Klienten gemeinsam benutzt werden, was allerdings die Konsequenz hat, daß Seiteneffekte auftreten können. Das bedeutet, daß ein Objekt in einem Kontext verändert wird, ohne daß dies in einem anderen bemerkt wird.

In der nachfolgenden Tabelle werden die wichtigsten Eigenschaften von Werten und Objekten nochmals aufgelistet.

Werte	**Objekte**
• haben keinen Lebenszyklus	• haben einen Lebenszyklus
• abstrakt, ohne Identität, nur Gleichheit	• Repräsentationen von Objekten in Einsatzkontext
• definiert oder undefiniert, aber unveränderlich	• veränderbar bei Wahrung der Identität
• keine gemeinsame Nutzung	• gemeinsame Nutzung
• referentiell transparent	

2.1.3. Unterscheidung von Werten und Objekten im Anwendungskontext

Nachdem die wesentlichen Eigenschaften von Werten und Objekten beschrieben wurden, läßt sich relativ einfach angeben, in welchen Fällen es angebracht ist, Werte anstelle von Objekten zu verwenden.
Objekte werden dann benötigt, wenn es darum geht, Artefakte und Konzepte eines Anwendungsbereiches mit ihren Umgangsformen und Zuständen zu modellieren. Im Gegensatz hierzu werden Werte immer dann benutzt, wenn in einem Anwendungskontext abstrakte Größen zu modellieren sind, bei denen man von den gegenständlichen oder anwendungsbezogenen Eigenschaften abstrahieren kann. Konkret heißt dies, Werte werden

immer dann benötigt, wenn eine Größe, wie ein Geldbetrag oder eine ganze Zahl, berechnet, repräsentiert oder quantifiziert werden soll. Das Ergebnis einer Berechnung ist dabei nur von den verwendeten Parametern und nicht vom Zustand eines Objektes oder des Gesamtsystems abhängig.
So sollte das Ergebnis einer Addition von zwei Geldbeträgen wie „20 DM“ und „30 DM“ unabhängig von seinem Verwendungskontext und dessen Zustand immer der Wert „50 DM“ sein. Ein Objekt „Konto“ kann beispielsweise durch einen Vorgang wie eine Überweisung von „30 DM“ den Wert seines Parameters „Kontostand“ von „20 DM“ auf „50 DM“ verändern, während der Inhaber des Kontos und die Kontonummer gleich bleiben. Das heißt das „Konto“ bleibt das Gleiche, nur sein Zustand hat sich geändert.
Bei [Züllighoven 98] wird darüber, wann Werte verwendet werden sollten, folgendes gesagt:

> *Wir benötigen Werte und Zahlen immer dann, wenn wir rechnen und ordnen müssen. Unbestritten brauchen wir dann ganze oder reelle Zahlen, aber auch DM-Beträge oder Zeiträume. Auch wenn wir von dem konkreten Umständen eines Gegenstandes absehen und meßbare Größen darstellen wollen, sind uns Werte wie eine Bankleitzahl oder der aktuelle Wert des DAX (Deutscher Aktien-Index) nützlich.*

Nachdem in diesem Abschnitt die Konzepte Wert und Objekt dargelegt wurden, sollen hierauf aufbauend im nächsten Kapitel die fachlichen Anforderungen an Fachwerte definiert werden.

2.2. Anforderungen an Fachwerte

Bei [Züllighoven et al. 98] wird als Grundlage für eine angemessene Verwendung von Werten die Einhaltung bestimmter Programmierkonventionen gefordert. Diese Art der Programmierung wird dort als wertorientierter Programmierstil bezeichnet. Hierbei sollten nur Ausdrücke verwendet werden, die vollständig aus Funktionen bestehen. Die Grundlage hierfür ist, daß diese Funktionen keinerlei Seiteneffekte haben und nah am Konzept algebraischer Ausdrücke sind. Das heißt, es muß Wertsemantik gelten. Wertsemantik bedeutet:

Wertsemantik:
Variablen haben einen Wert und können einen anderen Wert zugewiesen bekommen und auf Gleichheit geprüft werden.

Es wird also keine Identität von Objekten hergestellt. Wenn der Wert einer Variablen einer anderen zugewiesen wird und sich anschließend deren Wert ändert, bleibt bei der zweiten Variablen der Wert unverändert. Bei einer Prüfung auf Gleichheit zweier verschiedener Variablen werden nur die Werte verglichen. Die Wirkungsweise der Wertsemantik wird im folgenden Beispiel erläutert:

```
Integer   Var1,Var2;

Var1    = 42;
```
Zuordnung des Wertes <42> zur Variable `Var1`

```
Var2    = Var1;
```
Zuordnung des Wertes von `Var1` zur Variable `Var2`

```
Var1    = 0;
```
Zuordnung des Wertes <0> zur Variable `Var1` ohne Veränderung des Wertes von `Var2`

Alle objektorientierten Sprachen bieten die Wertsemantik für die Standarddatentypen an. Im Gegensatz hierzu steht die Referenzsemantik, bei der einer Referenzvariablen ein Verweis auf ein Objekt, nicht aber dessen Wert, zugewiesen wird.

Referenzsemantik:
Variablen verweisen auf ein Objekt und können ein Objekt zugewiesen bekommen. Sie können auf Gleichheit und Identität der Objekte geprüft werden.

Mehrere Variablen können also auf ein und dasselbe Objekt verweisen. Dadurch kann auf das Objekt über verschiedene Variablen zugegriffen werden. Somit ist die Möglichkeit gegeben, ein Objekt zu verändern, ohne daß dies an anderer Stelle bekannt wird.
Dadurch ist es möglich, daß es zu Seiteneffekten kommt, welche nicht mit dem mathematischen Variablenbegriff vereinbar sind. Hierbei sind die Variablen dann Bezeichner für bekannte bzw. unbekannte Werte. Die bezeichneten Werte sind hierbei unveränderbar. Dies bedeutet nichts anderes als referentielle Transparenz. Unter referentieller Transparenz soll hier folgendes verstanden werden:

Referentielle Transparenz:
Ein Ausdruck wird nur verwendet, um einen Wert zu benennen. In einem gegebenen Zusammenhang bezeichnet ein Ausdruck immer den gleichen Wert. Aus diesem Grund können Teilausdrücke durch andere mit dem gleichen Wert ersetzt werden.

[Hinze 92]

Aus diesem Grund sollten Werttypen nur im Sinne von Wertsemantik benutzt werden. Bei den objektorientierten Sprachen ist dies für die elementaren Datentypen, wie zum Beispiel Integer oder Boolean, weitestgehend realisiert. Sie lassen sich im allgemeinen so verwenden, wie man es aus der funktionalen Programmierung gewohnt ist. Probleme treten erst auf, wenn man benutzerdefinierte Werttypen verwenden möchte, die Werte in einem Anwendungsbereich modellieren. Solche Werttypen werden bei [Züllighoven et al. 98] als „Fachwerte“ und bei [Bäumer et al 98] als „domain values“ bezeichnet. Der Grund für die Verwendung solcher Werttypen läßt sich leicht aus den Kapiteln 1 und 2.1 erschließen, weshalb hier nur kurz nochmals darauf eingegangen wird. Seit den ersten Programmiersprachen war man bestrebt, Werttypen aus den jeweiligen Anwendungsbe-

reichen zu verwenden. Besonders wenn man wie beim WAM-Ansatz [Züllighoven et al. 98] versucht, Gegenstände und Konzepte aus dem jeweiligen Anwendungsbereich als Objekte zu modellieren, ist es naheliegend, die Werte mit ihren Wertebereichen und fachlichen Operationen als Werttypen abzubilden. Für die folgenden Kapitel wird der Begriff Fachwert als Bezeichnung für die oben beschriebenen Werttypen verwendet werden. Es läßt sich unschwer aus den vorangegangenen Abschnitten folgern, daß es keine befriedigende Lösung sein kann, solche Fachwerte durch elementare Datentypen oder Objekte darzustellen. Einen Geldbetrag nur durch eine Fließkommazahl zu modellieren reicht nicht aus, da hierbei oft nur am Bezeichner erkennbar ist, welche Art von Wert man gerade benutzt. Daneben kann nicht garantiert werden, daß nur fachlich motivierte Operationen auf diesen Wert ausgeführt werden. Es kann nicht einmal sichergestellt werden, daß Operationen, auch wenn alle Benutzer das gleiche Verständnis von diesen Operationen haben, an jeder Stelle auf die gleiche Art mit dem gleichen Ergebnis ausgeführt werden. Daneben erscheint die Lösung der Modellierung der Fachwerte durch abstrakte Datentypen schon deutlich besser, da hier Operationen und Wertebereich zusammengefaßt werden können und eine einheitliche Ausführung von Methoden garantiert wird. Die Einhaltung von Wertsemantik kann nur über einen Verzicht auf Methoden, die den Zustand eines Objektes nach seiner Erzeugung verändern können, garantiert werden. Ist dies nicht der Fall, kann durch solche verändernden Operationen der Wert eines Wertobjektes verändert werden, was den im Kapitel 2.1 aufgeführten Eigenschaften von Werten grundlegend widerspricht. Allerdings wird man nicht ohne die Verwendung von Objekten auskommen, da es in objektorientierten Sprachen wie Java oder C++ keine andere Möglichkeit gibt, eigene benutzerdefinierte Typen darzustellen.
Man braucht also ein erweitertes Konzept, um Werte bzw. Fachwerte in objektorientierten Sprachen umzusetzen. Als Grundlage dient hierbei die Definition von Fachwerten, wie sie bei [Züllighoven et al. 98] verwendet wird:

Fachwert:
Ein Fachwert ist ein benutzerdefinierter Wert. Er repräsentiert Werte im Anwendungsbereich.

Ein Fachwert ist ein Werttyp mit einem definierten Wertebereich und festgelegten Operationen. Seine innere Repräsentation ist verborgen.

Fachwerte werden in objektorientierten Sprachen als Klassen definiert. Wichtig ist, daß die Exemplare eines Fachwertes immer Wertsemantik besitzen, d.h. daß ein einmal gesetzter Wert nicht mehr verändert werden kann.

Um die Anforderungen an Fachwerte genauer beschreiben zu können, werden im folgenden Abschnitt die in der Definition genannten Eigenschaften von Fachwerten in die folgenden Themenbereiche unterteilt:

- Wertebereich und Konstruktoren
- Repräsentationen von Fachwerten und Selektoren

- Fachliche Operationen
- Fachwerte und Wertsemantik

2.2.1. Wertebereiche und Konstruktoren

Laut der obigen Definition muß für jeden Fachwert ein Wertebereich klar definiert sein. Ohne eine präzise Eingrenzung kann nicht entschieden werden, welche externen Repräsentationen gültige Werte darstellen und welche nicht. Hiermit wird auch schon klar, daß eine Fachwertklasse über einen oder mehrere Konstruktoren verfügen muß, welche in der Lage sein müssen, gültige von ungültigen Repräsentationen zu unterscheiden. Das heißt, die Konstruktoren realisieren also eine Abbildung aus der Menge aller möglichen Repräsentationen in den Wertebereich des jeweiligen Fachwertes. Die gültigen Repräsentationen bzw. Ausdrücke bilden den Argumentbereich. Gültig sind hierbei alle wohlgeformten Ausdrücke, da in der funktionalen Programmierung (siehe [Meyer 90] bzw. [Bird & Wadler 92]) jeder wohlgeformte (syntaktisch korrekte) Ausdruck einen Wert bezeichnet. Syntaktisch korrekt geformt bedeutet hier, es werden nur die erlaubten Argumenttypen wie zum Beispiel Integer oder Boolean in der richtigen Reihenfolge verwendet, wobei der konkrete Wert der Argumente keine Rolle spielt. Für alle anderen Repräsentationen muß bei der Typprüfung auf der technischen Ebene ein Programmabbruch erfolgen, was aber schon durch die verwendete Programmiersprache sichergestellt wird. Für Ausdrücke, die syntaktisch korrekt sind, deren Werte jedoch nicht zulässig sind, müßte deshalb ein undefiniertes Fachwertobjekt erzeugt werden. Dieses wäre eine formal saubere, aber ineffiziente Lösung, da hierdurch viele nicht benötigte Objekte erzeugt werden würden. Stattdessen müssen andere Möglichkeiten gefunden werden, mit dieser Situation umzugehen. Eine Möglichkeit wäre, in einem solchen Fall das Programm mit einer Fehlermeldung abzubrechen, wie dies auch in der funktionalen Programmierung bei einigen Sprachen der Fall ist. Um dies umzusetzen, könnte das Vertragsmodell nach Meyer [Meyer 90] verwendet werden. Bei allen anderen gültigen Repräsentationen muß ein neues Fachwertobjekt erzeugt werden, welches einen gültigen Wert darstellt. Wenn ein neuer Fachwert erzeugt wird, sollte dieser für seine interne Darstellung auf eine Normalform gebracht werden. Wie die im konkreten Fall aussieht, hängt stark von dem Einsatzkontext des Fachwertes ab. Bei den genannten Repräsentationen kann es sich sowohl um elementare Standarddatentypen als auch um andere Fachwerte handeln. Ein Konstruktor, welcher aus einer Zeichenkette ein Objekt des jeweiligen Fachwerttyps erzeugen kann, sollte immer vorhanden sein, da jeder denkbare Wert bzw. Fachwert über eine Zeichenkette dargestellt werden kann. Hierdurch hätte man eine universell verwendbare Repräsentationsform für Fachwerte, was insbesondere bei der Oberflächendarstellung oder der Speicherung von Fachwerten von Vorteil ist (siehe Kapitel 3.2.2. und 3.3.).

Aus den oben genannten Gründen sollten die Wertebereiche von Fachwerten deshalb immer auch den undefinierten Wert oder auch „Bottom“, wie er in der funktionalen Programmierung bezeichnet wird, enthalten (siehe [Cunningham 95]). Er sollte zwar nicht, wie oben angedeutet, bei jedem ungültigen Ausdruck erzeugt werden, aber man sollte ihn explizit über einen besonderen Konstruktor erzeugen können. Dies hätte den Vorteil, daß man ihn zum Beispiel als Default-Wert für noch nicht initialisierte Variablen einsetzen könnte. Dies wäre gegenüber der üblichen Methode, wo ein gültiger Wert als Default angesehen wird, ein großer Fortschritt. Bei der herkömmlichen Methode werden diese „gültigen undefinierten Werte“ oft wie andere Werte verwendet, was zu

erheblichen Problemen führen kann. Zum Beispiel wird oft ein Wert wie „999999“ für noch nicht bekannte Werte verwendet, wobei die Gefahr darin besteht, daß dieser Wert sich nicht nur in der Zahl von einem regulärem Wert „235638“ unterscheidet, sondern auch semantisch eine andere Bedeutung hat. Für diesen undefinierten Wert muß es allerdings auch eine geeignete sondierende Operation geben, mit der sich feststellen läßt, ob ein vorliegendes Fachwertobjekt definiert ist.
Wenn man nun Fachwertobjekte mittels der Konstruktoren erzeugen möchte, muß auch die Möglichkeit bestehen, im Vorfeld zu prüfen, ob eine Repräsentation einen gültigen Fachwert darstellt oder nicht. Dies bedeutet, daß es neben den Konstruktoren noch zwei weitere Arten von Operationen geben muß. Zunächst muß es passend für jeden Konstruktor eine sondierende Operation geben, welche die Aufgabe hat, zu prüfen, ob es sich bei einer gegebenen Repräsentation um einen gültigen Wert handelt. Die Entscheidung, ob ein Fachwert gültig ist, kann aber nur kontextfrei und nach syntaktischen Gesichtspunkten erfolgen. So kann zum Beispiel für einen Fachwert `Datum` und die Zeichenkette „01.01.1999“ ermittelt werden, ob es sich um eine gültige Repräsentation handelt. Es kann aber nicht entschieden werden, ob er in dem jeweiligen Anwendungskontext Gültigkeit besitzt. So könnte es sein, daß in einem Anwendungskontext unter bestimmten Voraussetzungen nur Daten verwendet werden dürfen, deren Wert kleiner ist als der „01.01.1990“. Demnach wäre der „01.01.1999“ zwar ein gültiger Fachwert vom Typ `Datum`, aber er könnte trotzdem nicht verwendet werden. Des weiteren sollte gegebenenfalls für die jeweilige Fachwertklasse die Möglichkeit vorhanden sein, sich Informationen über den gesamten Wertebereich geben zu lassen. Hierbei kann es oftmals sinnvoll sein, zu prüfen, ob der Wertebereich endlich oder unendlich ist. Falls er endlich und aufzählbar ist, ist es bei einigen Fachwerten hilfreich, sich die möglichen Werte geben zu lassen, um sie in einer Liste darzustellen. Ein Beispiel hierfür wäre die Auflistung aller möglichen Werte eines Fachwertes `Monat`, um sie in einer Auswahlliste bereitzustellen.

2.2.2. Repräsentationen von Fachwerten und Selektoren

Da die innere Repräsentation des Wertes eines Fachwertobjektes immer verborgen ist, muß man in der Lage sein, sich den Wert eines solchen anzeigen zu lassen, um ihn beispielsweise in Fenstersystemen, die keine Fachwerte benutzen können, durch elementare Datentypen wie Integer oder String anzeigen zu können. Als Rückgabewerte sind hierbei elementare Datentypen wie Integer, String und Fachwerte, falls das Fachwertobjekt solche enthält, erlaubt. Es bietet sich an, für dieselben Datentypen wie Integer oder String, die man auch als Argumenttypen bei den Konstruktoren findet, jeweils auch eine Methode anzubieten, um eine Repräsentation mit den gewünschten Typen zu erhalten. Die Mindestanforderung hierbei sollte eine Darstellung des Fachwertes als Zeichenkette sein. Der Grund hierfür liegt in der Tatsache, daß sich erstens jeder Fachwert mindestens über Zeichenketten darstellen läßt, und zweitens wird diese Form der Repräsentation für die Kommunikation mit dem jeweiligen Fenstersystem benötigt, da dieses im Normalfall keine benutzerdefinierten Datentypen verwenden kann. Vor jedem Zugriff muß geprüft werden, ob der betreffende Wert auch definiert ist. Sollte man aber dennoch versuchen, auf einen undefinierten Fachwert zuzugreifen, so müßte das Resultat (siehe [Meyer 90]) wieder ein undefinierter Wert sein. Allerdings gibt es bei den Standarddatentypen der gängigen objektorientierten Programmiersprachen keine undefinierten Werte. Eine einfache und pragmatische Lösung für dieses Problem wäre es, wenn es beim Zugriff auf

einen undefinierten Wert zu einer Fehlermeldung oder zu einem Programmabbruch kommen würde.
Neben den Repräsentationen des Fachwertes als Ganzes ist es bei zusammengesetzten Fachwerten oftmals sinnvoll auf einzelne Komponenten des Fachwertes zuzugreifen. So könnte bei einem Fachwert `Datum` nur der Tag von Interesse sein. Also muß es eine Möglichkeit geben, sich diese Teilkomponente mittels eines Selektors geben zu lassen. Für diese Selektoren muß im Falle eines Zugriffs auf einen undefinierten Wert das Gleiche gelten wie bei den Repräsentationen von Fachwerten.

2.2.3. Fachliche Operationen

Darüber hinaus muß es in den Fachwertklassen möglich sein, fachliche Operationen für den Umgang mit Fachwerten festzulegen, welche der vom Entwickler des Fachwertes festgelegten Semantik entsprechen. Das heißt für den jeweiligen Fachwert muß ein Kalkül festgelegt werden, welches alle bei der Ausführung von Operationen vorkommenden Fälle unter Einbeziehung des undefinierten Wertes behandelt. Auf diese Weise kann garantiert werden, daß nur Operationen immer zulässige Ergebnisse liefern. Unter zulässigen Operationen werden solche verstanden, die fachlich motiviert sind. Hierbei muß allerdings immer die Wertsemantik für Fachwerte gewahrt bleiben. Weiter sollten sich diese Operationen in ihrem Umgang wie Funktionen verhalten. Diese Methoden müssen natürlich nicht nur mit Fachwerten der eigenen Klasse arbeiten, sondern auch mit Standarddatentypen und anderen Fachwerttypen. Eine Methode `BerechneZinsen` würde folglich als Argumente einen Fachwert vom Typ `Geldbetrag`, einen vom Typ `Zinssatz` und einen vom Typ `Zeitspanne` erhalten und damit einen neuen Fachwert vom Typ `Geldbetrag` erzeugen. Dabei sollte es allerdings immer möglich sein, Ausdrücke aufgrund ihrer Teilausdrücke vollständig auszuwerten. Werte werden also gemäß dem Prinzip der referentiellen Transparenz verwendet. Vor der Ausführung einer solchen Operation sollte mit einer dafür zur Verfügung gestellten Methode geprüft werden, ob diese Operation mit den betreffenden Argumenten einen gültigen Wert liefert oder ob das Ergebnis ein undefinierter Wert ist. Ein Beispiel hierfür wäre die Division durch null bei den natürlichen Zahlen.
Werden bei Operationen undefinierte Werte als Argumente verwendet, muß das Ergebnis zwingend wiederum ein undefinierter Wert sein (siehe [Meyer 90]). Als Alternative wäre es allerdings auch möglich, es, wie oben bei den Konstruktoren vorgeschlagen, zu einem Programmabbruch bzw. einer Fehlermeldung kommen zu lassen. Wenn der Ergebnistyp allerdings ein elementarer Datentyp ist, hat man, wie schon bei den Konstruktoren beschrieben, keine Möglichkeit, einen undefinierten Wert zu erzeugen. Die einzige Möglichkeit hier ist, einen Programmabbruch herbeizuführen.
Zu den wichtigsten Operationen zählt der Test auf Gleichheit, welcher auch unbedingt in jeder Fachwertklasse realisiert sein muß. Dies kann durch das Überladen des entsprechenden Operators geschehen oder aber, wo dies nicht möglich ist, durch die Implementation einer geeigneten Methode, welche diesen Vergleich realisiert. Nach [Hoare 72] würde der Test auf Gleichheit und damit auch der auf Ungleichheit semantisch vollkommen ausreichen. Diese Tests auf Gleichheit bzw. Ungleichheit sind allerdings nur für Fachwerte gleichen Typs zugelassen. Es sollte nicht möglich sein, Fachwerte, welche vom Typ der Basisklasse sind, mit anderen Fachwerten, die den Typ einer abgeleiteten Subklasse haben, zu vergleichen. Sollten beide Werte dennoch verglichen werden, müßte ein solcher Test zu einem Programmabbruch bzw. Fehler führen. Eine Alternative hierbei

wäre, nur die Komponenten zu vergleichen, welche beiden Klassen zu eigen sind. Diese müßte man allerdings vorher mit Hilfe von Selektoren aus den Fachwerten gewinnen. Ausgenommen von dieser Regel ist der Fall, daß die Wertemenge der Basisklasse in der Wertemenge der Subklasse enthalten ist. Beispiel hierfür sind die rationalen Zahlen, welche die Menge der natürlichen Zahlen als Teil ihrer Wertemenge enthalten. In diesen Fällen müßte eine Typkonversion stattfinden, welche dann als zusätzliche Operation realisiert werden müßte.
Um die Handhabung von Fachwerten zu erleichtern, sollten, wo es für die betreffenden Fachwerte eine Ordnungsrelation gibt, auch noch andere Vergleichsoperationen wie „>“ und „<“ implementiert werden. Für diese Operationen gelten natürlich die gleichen Einschränkungen.

2.2.4. Fachwerte und Wertsemantik

Der nächste wichtige Punkt für Fachwerte ist ihre Realisierung durch Klassen und die gleichzeitige Forderung nach Wertsemantik. Dies bedeutet, daß Fachwerte nie mit Referenzsemantik sondern immer nur nach Wertsemantik behandelt werden dürfen. Ein Fachwertobjekt darf also nicht über zwei verschiedene Bezeichner zugreifbar sein. Streng genommen bedeutet die Forderung nach Wertsemantik sogar, daß nur einmal eine wertsetzende Operation ausgeführt werden darf. Danach darf der Wert nicht mehr verändert werden. Auf der konzeptionellen Seite ist dies ohne weiteres möglich, problematisch hierbei ist der enorm hohe Speicherverbrauch, der bei einer solchen Vorgehensweise benötigt wird, da laufend neue Objekte erzeugt werden müssen. Allerdings läßt sich dieses Problem auf technischer Ebene durch die Verwendung geeigneter Entwurfsmuster, welche im Folgekapitel erläutert werden sollen, entschärfen.
Wie bei den oben behandelten Punkten schon angedeutet, müssen Fachwerte und somit auch ihre Klassen und Objekte unter allen Umständen unabhängig von ihrem Kontext sein, da sonst die referentielle Transparenz nicht mehr gegeben ist. Daß heißt, daß bei der Erzeugung und der Ausführung von Operationen nur Informationen verwendet werden dürfen, welche in der Klasse bzw. in den vorliegenden Fachwertobjekten vorhanden sind oder als Argumente an einen Konstruktor für Fachwertobjekte übergeben werden. Im anderen Fall könnte es passieren, daß sich das Ergebnis einer Funktion abhängig vom Umfeld verändert. Als Beispiel für einen solchen Fall sei hier nochmals das Datumsbeispiel aus dem Abschnitt „Wertebereiche und Konstruktoren“ genannt.
In der nachfolgenden Abbildung werden die Anforderungen an Fachwerte noch einmal zusammengefaßt.

Anforderungen an Fachwerte:

1. Fachwerttypen sollten einen klar definierten Wertebereich besitzen, welcher auch den undefinierten Wert beinhaltet. Darüber hinaus müssen Operationen vorhanden sein, welche Informationen über den Wertebereich liefern
2. Neue Objekte werden nach vorheriger Prüfung durch entsprechende sondierende Operationen durch geeignete Konstruktoren erzeugt.
3. Fachwerte werden in objektorientierten Sprachen als Klassen definiert. Wichtig ist, daß die Exemplare eines Fachwertes immer Wertsemantik besitzen, d.h. daß ein einmal gesetzter Wert nicht mehr verändert werden kann.
4. Es sollten Operationen zur Verfügung stehen, über die auf die Repräsentationen der Fachwerte zugegriffen werden kann. Daneben müssen bei zusammengesetzten Fachwerten deren Teilkomponenten über Selektoren zugreifbar sein. Vor dem Aufruf muß aber über eine sondierende Operation geklärt werden, ob es sich um einen undefinierten Wert handelt.
5. Fachliche Operationen regeln den Umgang mit Fachwerten. Hierzu ist es erforderlich, daß sie eine eindeutige Semantik haben und natürlich auch über zugeordnete sondierende Operationen verfügen, mit denen sich die Vorbedingungen der fachlichen Operationen prüfen lassen.
6. Beim Umgang mit Fachwerten muß Wertsemantik garantiert werden.

3. Implementation

Nachdem im Kapitel 2 die grundsätzlichen Anforderungen an Fachwerte aufgestellt wurden, soll nun erläutert werden, welche konkreten Möglichkeiten es gibt, ein Fachwertkonzept umzusetzen. Darüber hinaus soll im Folgenden geschildert werden, welche Entwurfsmuster für eine Implementation zur Verfügung stehen und welche für die Beispielimplementation ausgewählt wurden.
Für eine konkrete Implementation des Fachwertkonzeptes wurden in *"Values in Object Systems*" [Bäumer et al. 98] und in *"Das objektorientierte Konstruktionshandbuch*" [Züllighoven et al. 98] zwei verschiedene Ansätze aufgezeigt. Bei Züllighoven werden die beiden Ansätze als veränderliche bzw. unveränderliche Fachwertobjekte bezeichnet. Bei Bäumer wird der Ansatz der veränderlichen Fachwertobjekte als "copy-on-write" bzw. der Ansatz der unveränderlichen Fachwertobjekte als "immutable objects" bezeichnet. Bei den Bezeichnungen wird natürlich schon eine Schwäche der ganzen Modellierung offensichtlich, nämlich daß man Werte als Objekte realisieren muß. In den vorangegangenen Abschnitten wurden die Unterschiede zwischen Werten und Objekten deutlich herausgestellt, deshalb erscheint es widersprüchlich, wenn nun Fachwerte durch Objekte in Programmen realisiert werden, aber leider gibt es in den gängigen objektorientierten Programmiersprachen wie zum Beispiel C++ oder Java keine andere Möglichkeit, derartige Konstrukte wie selbstdefinierte Werte umzusetzen. Deshalb muß bei der Implementation ein großer Aufwand darauf verwendet werden, daß sich diese Objekte gemäß den Anforderungen aus Kapitel 2 verhalten und gleichzeitig nur ein Minimum an Speicherplatz verbrauchen und auch sonst bei ihrer Verwendung keinen zusätzlichen Programmieraufwand verursachen.

3.1. Veränderliche und unveränderliche Fachwerte

Der Ansatz der unveränderlichen Fachwertobjekte ist in jeder objektorientierten Sprache einfach zu realisieren. Hierbei muß nur darauf geachtet werden, daß der Wert eines Fachwertobjekts nach der Erzeugung nicht verändert wird. Dadurch läßt sich die Forderung nach Wertsemantik auf relativ einfache Weise realisieren. Daraus ergibt sich, daß jedesmal, wenn ein anderer Wert benötigt wird, ein neues Objekt erzeugt werden muß. Hieraus folgt allerdings ein enormer Speicherverbrauch, da bei jeder Änderung ein neues Fachwertobjekt angelegt werden muß. Dies führt in allen Programmiersprachen schnell zu einem erheblichen Speicherplatzproblem.
Dieses Problem läßt sich jedoch durch die Verwendung des "flyweight/factory"-Entwurfsmusters (siehe [Gamma et al. 96]) lösen. Hierbei wird durch eine Fabrik, die als einzige Instanz in der Lage sein sollte, einen Fachwert zu erzeugen, sichergestellt, daß zu jedem möglichen Fachwert aus dem Wertebereich nur jeweils ein Fachwertobjekt existiert. Dies bedeutet, daß an der Schnittstelle der Fabrik die benötigten Methoden für die Erzeugung von Fachwerten zur Verfügung gestellt werden müssen. Desweiteren werden die bisher erzeugten Fachwertobjekte in der Fabrik gespeichert, so daß verglichen werden kann, ob ein zu erzeugender Fachwert bereits existiert. Sollte dies der Fall sein, wird kein neues Objekt erzeugt, sondern nur eine Referenz auf das schon be-

stehende Objekt als Resultat zurückgeliefert. Das Fachwertobjekt kann nun von mehreren Instanzen gemeinsam genutzt werden.

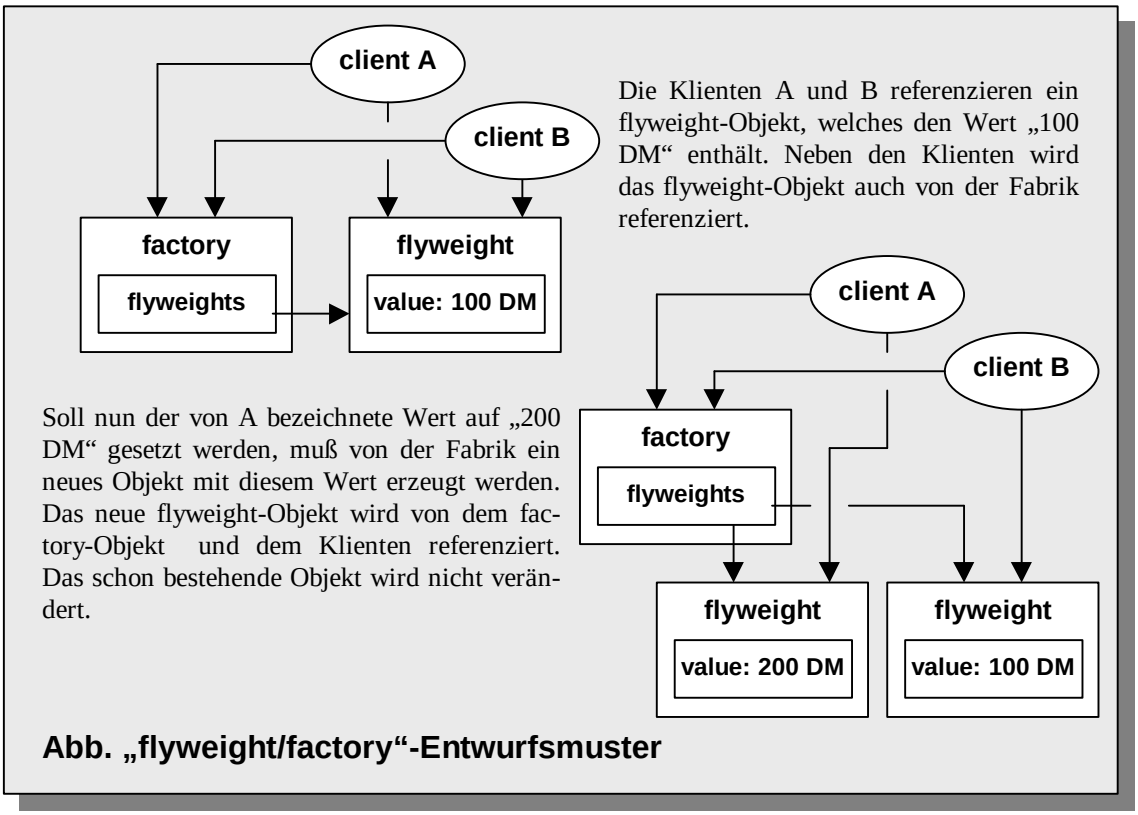


Abb. „flyweight/factory“-Entwurfsmuster

Da es nach der Erzeugung nicht mehr verändert werden kann, treten bei dieser Art der Nutzung keinerlei Seiteneffekte auf. Bei Züllighoven wird noch darauf hingewiesen, daß sich diese Art der Modellierung besonders für Fachwerte mit einem relativ kleinen und endlichen Wertebereich, dessen Werte bei Programmstart erzeugt werden und erst bei Programmende wieder gelöscht werden, eignet.
Bei der zweiten Möglichkeit der Implementation als veränderliche Fachwertobjekte wird ein anderer Weg beschritten. Die einfachste Form mit diesem ”copy-on-write”-An
satz Wertsemantik zu erreichen ist, die Werte bei einer Änderung einfach durch den Anwender kopieren zu lassen, so daß keine Seiteneffekte auftreten. Der Entwickler muß hierbei nur einen Kopiermechanismus entwerfen, alles andere liegt in der Verantwortung des Anwenders.
Eine verfeinerte Form des ”copy-on-write” ist die Verwendung des ”body/handle”-Entwurfsmusters (siehe [Coplien 92] und [Koenig&Moo 97]). Der Fachwert wird hierbei durch zwei Objekte implementiert. Bei dieser Konstruktion kann der Anwender nur über das handle-Objekt, welches eine Referenz auf das body-Objekt enthält, auf den Wert zugreifen. Dieser wird im body-Objekt gekapselt und ist über wertsetzende Operationen veränderbar. Alle Methodenaufrufe am handle-Objekt werden direkt an das zugehörige body-Objekt weitergeleitet. Da auf jedes body-Objekt mehrere handle-Objekte verweisen können, muß im body-Objekt mittels eines Referenzzählers die Anzahl der Verweise vermerkt werden. Bei jeder Änderung muß, wenn die Anzahl der Referenzen größer als eins ist, eine Kopie des body-Objektes angelegt und der Zähler dementsprechend um eins

verringert werden. Fällt die Anzahl der Referenzen auf null, so kann das Objekt gelöscht werden.

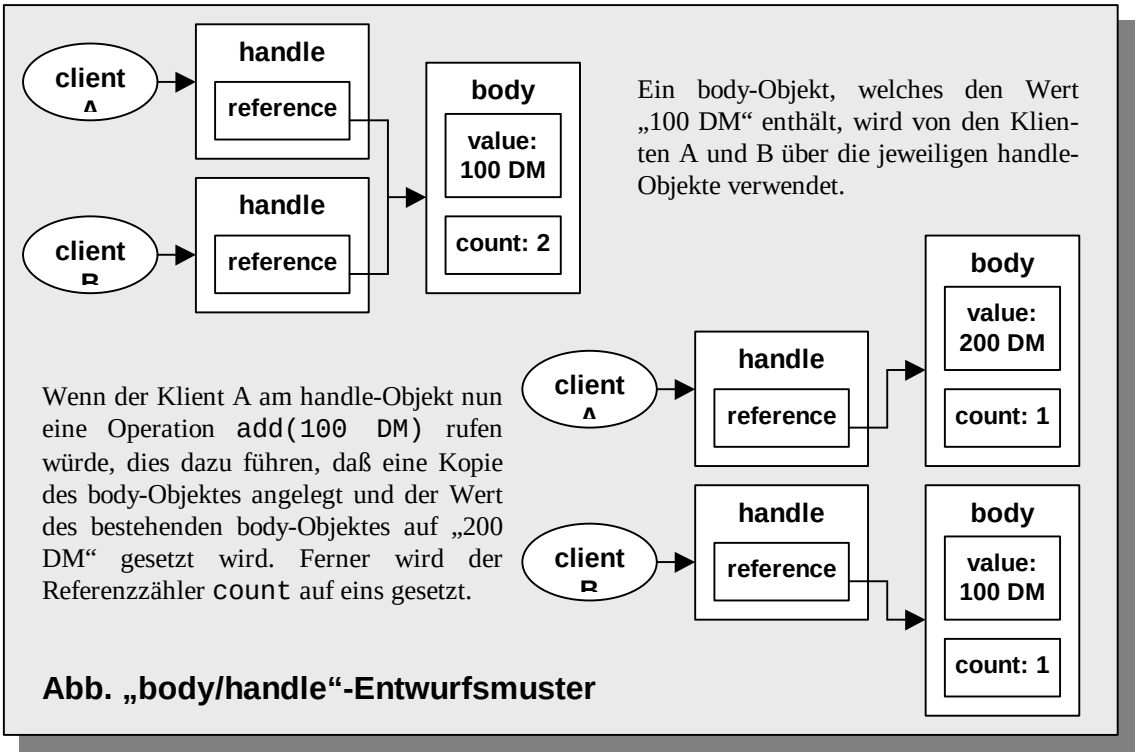


Abb. „body/handle“-Entwurfsmuster

Um allerdings Wertsemantik zusichern zu können, muß auch das handle-Objekt bei einer Zuweisung immer kopiert werden. Außerdem darf es nie über Referenzen benutzt werden. Ansonsten würde dieses Objekt nach Referenzsemantik benutzt werden. In diesem Fall könnte das handle-Objekt durch mehrere Instanzen referenziert werden, wodurch es zu Seiteneffekten kommen könnte. Der Verzicht auf Referenzsemantik kann nur durch Programmierkonventionen für die Verwendung von Fachwerten durch den Anwender garantiert werden. Zyklischen Referenzen , wobei die Objekte sich gegenseitig referenzieren und so der Referenzzähler nie auf null fallen kann, können hierbei nicht auftreten, da in diesem Fall schon bei der Modellierung der Fachwerte Fehler gemacht wurden. Es kann nämlich nie der Fall sein, daß ein Wert sich selbst als Element enthält.

3.1.1. Auswahl des Entwurfsmusters für die Implementation

Bei Umsetzung für das JWAM-Rahmenwerk wurden die Fachwerte als unveränderliche Objekte mit dem ”flyweight/factory”-Entwurfsmuster modelliert. Die Kriterien für diese Entscheidung waren hierbei die beste Annäherung an das Konzept der Werte, welche mit objektorientierten Sprachen möglich ist, die Erfahrungen aus dem Artikel ”*Values in Object Systems*” [Bäumer et al. 98] und die Eignung des zum Ansatz zugehörigen Entwurfsmusters für die Programmiersprache Java. Im Folgenden soll nun im Detail erläutert werden, warum dieser Ansatz ausgewählt wurde.

Bei der einfachen Form des ”copy-on-write”, wie sie im vorigen Abschnitt dargestellt wurde, fällt natürlich sofort auf, daß hier schnell ein Speicherplatzproblem entsteht, wenn viele Manipulationen an Fachwerten vorgenommen werden. Daneben ist die Einhaltung der Wertsemantik durch Benutzungskonventionen für die Anwender sehr fehleranfällig. Allerdings können hierbei auch solche Objekte als Fachwerte verwendet werden, die

ursprünglich nicht für eine Verwendung nach Wertsemantik vorgesehen waren. Wenn nun das ”body/handle”-Entwurfsmuster für die Implementierung angewendet wird, wird der Speicherverbrauch minimiert und gleichzeitig verhindern die handle-Objekte eine Verletzung der Wertsemantik. Dies gilt aber nur solange, wie sichergestellt ist, daß die body-Objekte nicht direkt zugreifbar sind und die handle-Objekte nur von jeweils einem Klienten referenziert werden. Nachteilig ist zudem, daß es jeweils zwei Klassen gibt, um einen Fachwert zu modellieren. Dies kommt besonders zum Tragen, wenn die body-Objekte relativ klein sind. Hinzu kommt noch, daß die Performanz von Programmen durch die Indirektion zwischen den Objekten vermindert wird. Als weiterer Vorteil wird häufig noch die leichte Manipulationsmöglichkeit in Listenstrukturen oder anderen Behältern genannt. Danach können diese Fachwertobjekte einfach durch wertverändernde Operationen manipuliert werden, ohne daß eine erneute Erzeugung notwendig wird. Es würden einfach die Werte der body-Objekte neu gesetzt werden. Hierfür müßten dann spezielle Fachwertcontainer gebaut werden, welche diese Aufgabe erledigen können. Beispielsweise könnte auf diese Weise in einer Liste, welche mit Fachwerten vom Typ `Betrag` gefüllt ist, leicht jeder Fachwert verdoppelt werden, ohne daß hierzu neue Fachwerte erzeugt werden müßten. Allerdings besteht hierbei die Gefahr, daß es zu Seiteneffekten kommt. Diese Möglichkeit der Benutzung von Fachwerten nach Referenzsemantik macht den schwerwiegenden Nachteil dieses Ansatzes deutlich, daß sich nämlich veränderliche Fachwertobjekte vom Konzept her nicht wie Werte verhalten. Werte und somit auch Fachwerte sind per Definition unveränderlich, und somit widerspricht schon die Idee der Veränderlichkeit von Werten dem Konzept der Werte und dem der Fachwerte.

Die Variante der unveränderlichen Fachwertobjekte ohne Verwendung des ”flyweight/-factory”-Entwurfsmusters hat gegenüber den veränderlichen Objekten den entscheidenden Vorteil, daß er sehr einfach zu implementieren und zu verwenden ist. Der Entwickler muß sich keine Gedanken über Seiteneffekte machen, da diese nicht vorkommen können. Die Einhaltung der Wertsemantik ist deshalb garantiert. Die Benutzung eines Fachwertobjektes durch mehrere Prozesse ist folglich ohne Einschränkung möglich.

Der Nachteil dieser Lösung ist der hohe Speicherverbrauch. Wird nun das ”flyweight/factory”-Entwurfsmuster benutzt, fällt der Nachteil des hohen Speicherbedarfs für mehrfach vorhandene, den gleichen Wert bezeichnende Fachwertobjekte weg. Der Aufwand, der für die Verwaltung der Fachwertobjekte entsteht, wird durch den Vorteil des geringeren Speicherbedarfs und Verwaltungsaufwandes für die Fachwertobjekte aufgewogen. Ein weiterer wichtiger Vorteil der unveränderlichen Fachwertobjekte ist die Tatsache, daß sie sich so nah am Konzept der Werte bewegen, wie es in Java nur möglich ist.

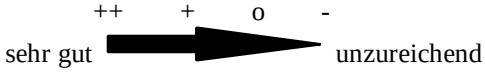
Die Forderung nach Wertsemantik bzw. referentieller Transparenz wird also bei beiden Ansätzen erfüllt, allerdings müssen beim Ansatz der veränderlichen Objekte viele Programmierkonventionen eingehalten werden. Beide Ansätze unterscheiden sich in Bezug auf ihren Speicherverbrauch kaum voneinander. Bei beiden wird der Speicherverbrauch durch die Verwendung des ”flyweight/factory”-Entwurfsmusters bzw. des ”body/handle”-Entwurfsmusters eingeschränkt. Allerdings kann es beim ”body/handle”-Entwurfsmuster vorkommen, daß Objekte mehrfach vorkommen, da hier nicht überprüft wird, ob der Wert schon einmal im System vorliegt.

Ähnliches gilt für die Erzeugung neuer Fachwertobjekte, bei der der Aufwand für den Anwender, der einen der beiden Ansätze verwendet, fast gleich ist. Unterschiede gibt es hier allerdings für den Entwickler, der den Fachwert implementieren muß. Denn bei den

unveränderlichen Objekten muß neben dem Fachwertobjekt auch noch die Fabrik mit all ihren Verwaltungs- und Erzeugungsfunktionen oder eine Instanz, welche ähnliches leistet, realisiert werden. Bei den veränderlichen Fachwertobjekten muß der Entwickler zusammen mit Anwender für die Einhaltung der Wertsemantik Sorge tragen. Der entscheidende Unterschied liegt in der Konzeptnähe der unveränderlichen Objekte. Durch diese Modellierung erreicht man den höchstmöglichen Grad an wertähnlichem Verhalten.
Neben diesen Überlegungen spielte noch die zu verwendende Programmiersprache Java bei der Entscheidung für einen Ansatz der unveränderlichen Objekte eine Rolle. Hiernach verursacht der Ansatz der unveränderlichen Objekte unter Java erheblich weniger Probleme als es bei den veränderlichen Objekten der Fall gewesen wäre, da in dieser Sprache schon eine funktionierende Garbage-Collection vorhanden ist und es somit an dieser Stelle keine Probleme mit Belegung von Speicherplatz durch nicht mehr benötigte Objekte gibt.
In der nachstehenden Tabelle aus dem oben genannten Artikel [Bäumer et al. 98] werden einige der Vor- bzw. Nachteile der verschiedenen Ansätze zusammengefaßt. Allerdings wurden hier noch einige weitere Punkte hinzugefügt.

	Unveränderliche Objekte [2]	**Unveränderliche Objekte (unter Verwendung des "flyweight/factory"-Entwurfsmusters)**	***Veränderliche Objekte*** [2]	**Veränderliche Objekte (unter Verwendung des "body/handle"-Entwurfsmusters)**
Klare Realisierung des Konzeptes	++	++	-	+
Einhaltung von Wertsemantik	++	++	+ *(nur durch Konventionen)*	+ (nur durch Konventionen)
Speicherverbrauch	-	++	*o*	++
Performanz beim Erzeugen neuer Fachwerte	+	+	*o*	+
Performanz bei der Benutzung	++	++	++	lesen: ++ schreiben: o
Verwendung in Bibliotheken	*o*	o	++	++
Geeignete Programmiersprachen	*Java*	Java (C++)	*Java (C++)*	C++

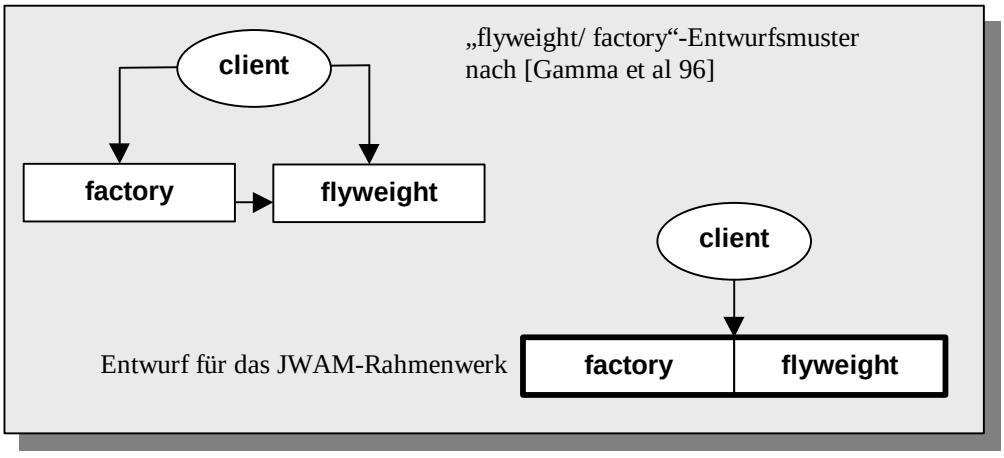
[1] Erfüllung der Anforderungen:



[2] Die hinzugefügten Spalten wurden kursiv dargestellt.

3.2. Architektur

Nachdem in den vorangegangenen Kapiteln das ausgewählte Entwurfsmuster erläutert und die Entscheidung für die Verwendung begründet wurde, soll nun die Architektur des Fachwertkonzeptes für das JWAM-Rahmenwerk beschrieben werden. Als Programmiersprache wurde hierzu die Sprache Java gewählt.
Wie bereits erläutert, werden bei dem hier verwendeten Ansatz unveränderliche Fachwertobjekte benutzt, um die Forderung nach Wertsemantik zu erfüllen. Um die dadurch entstehende Speicherplatzproblematik zu vermeiden, wird das ”flyweight/factory”-Entwurfsmuster verwendet. Um dieses zu realisieren, wird eine Fabrik oder eine äquivalente Instanz benötigt. Die Fabrik hat hierbei, wie oben beschrieben, die Aufgabe, Fachwertobjekte zu erzeugen bzw. zu verwalten. Das heißt, sie muß zumindest die eine Methode zur Erzeugung von Fachwerten und eine geeignete Behälterstruktur für die Fachwerte bereitstellen. Wenn dieses Entwurfsmuster ohne Anpassungen übernommen würde, hieße dies, daß alle Konstruktoren und fachlichen Operationen, durch welche neue Fachwerte erzeugt werden können, in der Fabrik realisiert werden müßten. Der Fachwert hätte dann nur noch die Aufgabe eine Repräsentationen seines Wertes zu liefern. Würde man auf der anderen Seite die vorhin genannten fachlichen Operationen an den Fachwertobjekten ansiedeln, müßte man einen Benachrichtigungsmechanismus für die Kommunikation zwischen Fachwertobjekt und Fabrik implementieren. Daneben hätte dieser Entwurf noch den Nachteil, daß hierbei für jeden Fachwert zwei Klassen zu implementieren und zu pflegen sind. Zum anderen ist es unter Java günstig, die Anzahl der Objekte, welche zur Laufzeit benötigt werden, möglichst gering zu halten, um die Performanz des Gesamtsystems nicht zu beeinträchtigen. Aus diesen Grund wurden im JWAM-Rahmenwerk die Klassen der Fabrik und der Fachwertobjekte, welche das „flyweight/factory“-Entwurfsmuster realisieren, zu einer Klasse zusammengefaßt, wobei die Methoden und Variablen der Fabrik als Klassenmethoden bzw. –variablen umgesetzt werden.



Dies hat die Vorteile, daß nur eine Klasse zu pflegen ist und zur Laufzeit kein Fabrikobjekt im System vorhanden ist. Daneben entfällt für den Benutzer der Fachwerte die Aufgabe, ein Fabrikobjekt vor der Verwendung explizit instanziieren zu müssen, da die benötigten Methoden und Variablen bei der ersten Benutzung schon als Teile der Klasse des Fachwertes vorliegen. Auch müßte dieses Fabrikobjekt als Singleton vorliegen, da ansonsten das „flyweight/ factory“-Entwurfsmuster seinen Zweck nicht erfüllen kann.

Unter Java würde dies bedeuten, daß über eine Klassenmethode erst geprüft werden müßte, ob es bereits ein Fabrikobjekt gibt. Ein Aufruf müßte dann unter Java wie folgt aussehen:

```
// Erzeugung der Fabrik
ValueFactory factoryObject  =  ValueFactory.getSingleton();

// Erzeugung des Fachwertes durch einen Aufruf an der Fabrik
DomainValue value = ValueFactory.newDomainValue([arguments]);
```

oder kürzer:

```
DomainValue value = ValueFactory.getSingleton().
                                    newDomainValue([arguments]);
```

Dagegen würde bei der Zusammenfassung von Fabrik und Fachwert der Schritt der expliziten Erzeugung wegfallen. Dort hätte der gleiche Aufruf folgende Form:

```
DomainValue value = DomainValue.newDomainValue([arguments]);
```

Daneben können fachliche Operationen, welche andere Fachwertobjekte erzeugen, an den Fachwertobjekten angesiedelt werden, ohne daß hierfür eine Kommunikation zwischen Fabrik und Fachwert stattfinden muß. Der Nachteil dieser Modellierung ist, daß Klassenmethoden nicht in Interfaces deklariert werden können, da dies unter Java nicht möglich ist. Zusammenfassend kann gesagt werden, daß mit dieser Form der Modellierung die Entwicklung und Verwendung vereinfacht wird.
Um also die Zahl der Klassen und Objekte zu verringern, sowie die Implementation und Pflege zu vereinfachen, werden die Verwaltungs- und Erzeugungsmethoden der Fabrik als Klassenmethoden in die Klasse des jeweiligen Fachwertes aufgenommen. Ebenso wird der Behälter, welcher die Aufgabe hat, die Fachwertobjekte zu verwalten, als Klassenvariable in diese aufgenommen. Hieraus folgt, daß die Verwaltungsfunktionen, der Behälter, die Konstruktoren mit den dazugehörigen sondierenden Operationen und alle Operationen zum Wertebereich, seien es sondierende Operationen oder zum Beispiel Methoden, welche alle Werte des Wertebereiches auflisten, Klassenmethoden bzw. -variablen sein müssen. Die Selektoren, die fachlichen und sondierenden Operationen werden an den Fachwertobjekten angesiedelt. Bei den Selektoren und sondierenden Operationen ist diese Entwurfsentscheidung auf den ersten Blick einleuchtend, denn was sollte selektiert oder sondiert werden, wenn kein Fachwertobjekt vorhanden ist. Auch bei fachlichen Operationen ist dieses Vorgehen sinnvoll, da alle Operationen auf Fachwerten zumindest unär sind. Das heißt, es wird immer mindestens ein Operand an einen Operator gebunden, also muß immer ein Fachwertobjekt vorhanden sein, damit Operationen überhaupt möglich sind. Daneben gibt es natürlich Ausdrücke mit mehr Operanden bzw. Operatoren. Hierfür läßt sich das Curry-Prinzip aus der funktionalen Programmierung (siehe [Bird & Wadler 92] und [Holyer 91]) als Vorbild ansehen. Beim Curry-Prinzip werden Operanden und Operatoren nach und nach zusammengefaßt. Eine Funktion, welche zwei Werte wie ”40” und ”2” addiert, könnte hiernach auch als eine Funktion aufgefaßt werden, bei welcher zu einem festen Argument ”40” beliebige andere Werte wie zum Beispiel ”2” addiert werden können. Hieraus folgt, daß ein Objekt mit einer dazugehörigen Operation als Ausdruck aufgefaßt werden kann, der noch einen zweiten Parameter benötigt, um weiter zusammengefaßt zu werden. Neben diesen

fachlichen Methoden müssen Fachwertobjekte noch ihre Werte in geeigneter Form enthalten, diese aber vor dem Anwender verbergen. Um aber auf die Repräsentationen und deren Elemente der Werte zuzugreifen, müssen an den Fachwerten Methoden vorhanden sein, durch die man Repräsentationen des gesamten Fachwertes bzw. einzelner Bestandteile dieses Wertes als Ergebnis erhält.
Die zur Persistenz benötigten Methoden lassen sich nicht so einfach zuordnen, deshalb wird die Persistenz von Fachwerten in einem eigenen Kapitel (siehe Kap. 3.3.) behandelt.

3.2.1. Aufbau von Fachwertklassen

Im Folgenden soll nun gezeigt werden, wie Fachwerte auf der Grundlage dieses Entwurfsmusters gemäß den Anforderungen aus Kapitel 2 realisiert werden können. Die getroffenen Designentscheidungen werden an den jeweiligen Stellen erläutert.
Hierfür wurden die Fachwertklassen `Zinssatz` und `Beschaeftigungsart` ausgewählt. Diese Beispiele wurden ausgewählt, da es sich hierbei um sehr einfach aufgebaute Fachwerte handelt. Hier wird allerdings jeweils nicht die ganze Schnittstelle sondern nur auszugsweise beschrieben. Zunächst soll die Klasse `Zinssatz` beschrieben werden. Die Methoden des Fachwertes `Beschaeftigungsart` werden nur beschrieben, wenn sie sich von der Verwendung oder von ihrer Implementation her deutlich von der des Fachwertes `Zinssatz` unterscheiden.

3.2.1.1. Beispiele

Fachwert `Zinssatz`

Bei der Klasse `Zinssatz` handelt es sich einen um numerischen Fachwert mit einem unendlichen Wertebereich. Die fachlichen Umgangsformen entsprechen der normalen Verwendung von Zinsen.

Fachwert `Beschaeftigungsart`

Bei der Klasse `Beschaeftigungsart` handelt es sich im Gegensatz zum `Zinssatz` um einen Fachwert mit einem endlichen Wertebereich, dessen Werte über Konstanten festgelegt wurden. Der fachliche Umgang mit diesem Fachwert ist für die folgenden Beispiele nicht von Interesse und wird deshalb nicht aufgeführt.

Für den Umgang mit dem undefinierten Wert wurde folgende Semantik festgelegt:

- Wenn externe Repräsentationen mittels Konstruktoren in Fachwerte transformiert werden sollen, wird nur für gültige Repräsentationen ein Fachwertobjekt erzeugt. Ansonsten wird eine Exception ausgelöst. Stellt die externe Repräsentation einen undefinierten Fachwert dar, wird ebenfalls eine Exception geworfen.
  Undefinierte Fachwerte lassen sich explizit über einen gesonderten Konstruktor erzeugen.
- Sollen Fachwerte in eine externe Repräsentation überführt werden, so wird beim Zugriff auf einen undefinierten Wert eine Exception ausgelöst.
- Ist bei fachlichen Operationen auf Fachwerten eines oder mehrere der Argumente undefiniert oder ist das Ergebnis dieser Operation undefiniert, wird eine Exception ausgelöst.

Der hier gezeigte Umgang mit dem undefinierten Wert ist nur einer von mehreren verschiedenen Umgangsformen. Es kann für bestimmte Anwendungsfälle der Fall sein, daß ein anderer Umgang notwendig ist. Deshalb muß er bei Bedarf an die Erfordernisse der jeweiligen Fachwerte angepaßt werden.

3.2.1.2. Unterteilung der Schnittstellen

Die Schnittstellen der Fachwerte lassen sich in zwei Abschnitte einteilen. Zunächst gibt es die Methoden, welche konzeptionell der Fabrik zugeordnet sind. Hierbei handelt es sich um die Konstruktoren, Verwaltungsmethoden, die Methoden zum Wertebereich und eine geeignete Behälterstruktur. Diese werden als Klassenmethoden bzw. -variablen implementiert. Der andere Abschnitt umfaßt die Selektoren, fachliche Operationen und die Variablen, welche intern den Wert des Fachwertes repräsentieren. Diese werden als Instanzmethoden bzw. -variablen umgesetzt.
Nachfolgend werden in den Abschnitten 3.2.1.3. bis 3.2.1.6 zuerst die Klassenmethoden aufgeführt und erläutert. Die Abschnitte 3.2.1.7 und 3.2.1.8 behandeln dann die Methoden der Objekte.

3.2.1.3. Methoden zum Wertebereich

Dies sind Methoden, welche Auskunft über den Wertebereich geben. Solche Methoden werden als statische Methoden an der Klasse zur Verfügung gestellt. Wie bereits im Kapitel 2 erläutert, umfaßt dies zum Beispiel für Fachwerte mit endlichem Wertebereich Auflistungen, welche alle möglichen Werte enthalten. Es sollte klar sein, daß solche Methoden als `public` deklariert werden sollten, um den allgemeinen Zugriff zu gewährleisten.
Für den Fachwert `Beschaeftigungsart` wäre es sinnvoll, wenn ein Entwickler sich alle möglichen Werte auflisten lassen könnte, um sie beispielsweise in einer Auswahlbox darzustellen. Hierzu wäre es hilfreich, eine Funktion zur Verfügung zu haben, welche beispielsweise alle Fachwerte erzeugt und als Array zurückgibt. (Die Erzeugung von Fachwerten mit der Methode `newValue` wird in einem nachfolgenden Abschnitt behandelt.)

Definition der möglichen Werte für die Klasse Beschaeftigungsart

```
private static final String _Beamter      = “Beamter“
private static final String _Angestellter = “Angestellter“
      . . .

public static Beschaeftigungsart[] listValues()
{
      Beschaeftigungsart[] valueList = {newValue(_Beamter),
                                        newValue(_Angestellter)};
      return valueList;
}
```

In anderen Fällen, wo man nicht alle möglichen Fachwerte aufzulisten kann, sollte der Anwender in der Lage sein sich, sich Grenzen des Wertebereichs angeben zulassen, wenn solche Grenzen fachlich motiviert sind. Solche Methoden sind allerdings nur bei solchen Fachwerten sinnvoll, für die eine Ordnungsrelation existiert. So könnte beispielsweise für den Fachwert `Zinssatz` der kleinste mögliche Wert „0 %“ und der größte mögliche

Wert „100 %“ betragen. Würde man dem Anwender diese Informationen vorenthalten, könnte dieser nur durch Ausprobieren herausfinden, wo diese Grenzen liegen.

Deklaration des maximal möglichen Wertes

```
private static final double _maxValue = 100.0;

public static Zinssatz getMaxValue()
{
      return newValue(_maxValue);
}
```

3.2.1.4. Verwaltung von Fachwerten

Wie schon beschrieben, benötigt man für die Verwaltung der Fachwerte einen Behälter, damit festgestellt werden kann, welche Fachwertobjekte bereits vorhanden sind. In dem Fall heißt dies, daß hier eine Behälterstruktur für die Fachwertobjekte vorhanden sein muß sowie eine Methode zum Einfügen von Fachwertobjekten in diese Struktur bzw. zum Prüfen, ob ein Wert bereits vorliegt. Diese Methode ist für beide Fachwerte gleich, da beide gemäß dem gleichen Entwurfsmuster gebaut wurden..
Als Behälterstruktur könnte man eine einfache Listenstruktur, wie einen `Hashtable`, verwenden. Dies führt aber zu einem Speicherplatzproblem. Unter Java werden Objekte solange nicht von der Garbage-Collection gelöscht, wie noch mindestens eine Referenz auf dieses Objekt verweist. Da wir aber über die Liste noch eine Referenz auf die Objekte halten müssen, um prüfen zu können, ob ein Objekt bereits vorhanden ist, kann keines dieser Objekte gelöscht werden. Das heißt, auch Fachwertobjekte, die von keinem Klienten mehr gebraucht werden, verbleiben im System. Ab dem JDK 1.2 gibt es aber nicht mehr nur einen Typ von Referenzen, sondern es gibt jetzt mehrere Arten von Referenzen, welche alle im Package `java.lang.ref` enthalten sind. Hiernach gibt es folgende Arten von Referenzen, bei denen die unterschiedlichen Grade von Erreichbarkeit der referenzierten Objekte (siehe [Pawlan 98]) unterschieden werden:

strongly reachable
Ein Objekt ist direkt von dem referenzierenden Objekt zugreifbar. Objekte, welche auf diese Weise referenziert werden, können von der Garbage Collection nicht gelöscht werden.

softly reachable
Ein über eine `soft reference` erreichbares Objekt kann von der Garbage Collection bei Speichermangel gelöscht werden. Wird nach dem Löschen versucht, über das Referenzobjekt auf das gelöschte Objekt zuzugreifen, wird das Objekt vom dem Referenzobjekt neu erzeugt.

weakly reachable
Der Unterschied zur soft reference besteht darin, daß die Garbage Collection ein so referenziertes Objekt nicht nur bei Speichermangel sondern immer löscht.

phantom reachable
Ein Objekt ist dann phantom reachable, wenn keine strong, soft oder weak reference vorliegt. Dieser Fall liegt vor, wenn die `finalize`-Methode eines Objektes ausgeführt wurde, aber das Objekt nicht gelöscht wurde.

unreachable
Ein Objekt ist dann unreachable, wenn es nicht durch eine der oben genannten Arten referenziert wird.

Wenn ein Objekt über mehrere verschiedene Arten referenziert wird, wird bei der Garbage Collection immer die stärkste Form der Referenz berücksichtigt.

Objekte können also auf verschiedene Arten referenziert werden. Für das oben genannte Problem würde sich die `weak reference` anbieten, da hierbei immer die Objekte gelöscht werden, die nur noch über diesen Typ von Referenzen zugreifbar sind, so daß alle nicht benötigten Fachwertobjekte gelöscht werden. Für die effiziente Verwaltung solcher Referenzen in Listen steht im JDK 1.2 die Klasse `WeakHashMap` (siehe Package `java.util`) zur Verfügung, welche alle notwendigen Operationen beinhaltet. In ihren Methoden ähnelt sie der aus dem JDK 1.0 bekannten `hashtable`. Nähere Informationen sollten der Dokumentation des JDK 1.2 entnommen werden.
Die Prüfung, ob ein Fachwertobjekt bereits vorhanden ist, und das Einfügen von Fachwertobjekten in die `WeakHashMap`, wird von der Verwaltungsfunktion `registeredValue` übernommen. Für jedes neue Objekt wird geprüft, ob ein Objekt mit dem gleichen Wert bereits in dem Behälter vorhanden ist. Dieser Methode wird ein Fachwertobjekt als Argument übergeben. Bei der Prüfung wird das Objekt selbst als Schlüssel verwendet, so daß kein gesonderter Schlüsselwert erzeugt werden muß. Wenn das Objekt bereits vorhanden ist, wird das existierende Objekt als Ergebnis zurückgeliefert. Ansonsten wird das neue Objekt eingefügt und als Ergebnis zurückgeliefert. Der Rückgabewert ist also ein Fachwert vom Typ `Zinssatz`.

```
private static WeakHashMap _domainValueHashMap;

private static Zinssatz registeredValue(Zinssatz value)[1]
{
```

Prüfung der Vorbedingungen:
Der übergeben Wert darf nicht ´null‘ sein.

```
  Contract.require(value != null);

  Zinssatz    resultValue;
```

Prüfung, ob das Objekt bereits vorhanden ist.

```
  if (_domainValueHashMap.containsKey(value)) {
```

Wenn bereits Objekt vorhanden ist, wird das bereits vorhandene Objekt der Ergebnisvariable zugewiesen.

```
        resultValue = (Zinssatz)
        _domainValueHashMap.get(value);
  } else {
```

Wenn bereits Objekt nicht vorhanden ist, wird das neue Objekt in die Liste aufgenommen und Ergebnisvariable zugewiesen.

```
        _domainValueHashMap.put(value,value);
        resultValue = value;
  }
```

Prüfung der Nachbedingungen:
Der Rückgabewert darf nicht gleich ‘null‘ sein und muß gleich dem übergebenen Wert sein.

[1] Die Verwendung der Methode `registeredValue` wird bei den Konstruktoren gezeigt.

```
        Contract.ensure(resultValue != null);
        Contract.ensure(resultValue.equals(value));

        return resultValue;
    }
```

In der `WeakHashMap` werden Objekte mittels der Methode `equals(Object value)` verglichen. Da bei der Standardimplementation dieser Methode die Objekte verglichen werden, muß diese Methode dahingehend geändert werden, daß die Werte der Objekte verglichen werden. Die Methode `equals(Object value)` wird in diesem Kapitel im Abschnitt „Fachliche Operationen und die dazugehörigen sondierenden Operationen“ erläutert. Daneben muß die Methode `hashCode()` überschrieben, so daß für Fachwerte, bei denen `equals` einen wahren Wert liefert, auch das Ergebnis dieser Methode das Gleiche ist. Bei der Methode `hashCode()` wird einfach der Fachwert in einen String konvertiert und an diesem dann die Methode `hashCode()`aufgerufen. Das bedeutet, daß Fachwerte, die durch den gleichen String darstellbar sind, auch den gleichen Rückgabewert bei dieser Methode haben.

```
public int hashCode()
{
        return (this.toString()).hashCode();
}
```

Die Implementation der Methode `toString()` wird im Abschnitt „Repräsentationen von Fachwerten und Selektoren“ beschrieben**.**

3.2.1.5. Sondierende Methoden zu den Konstruktoren

Dies sind Methoden, welche darüber Auskunft geben, ob eine gegebene Repräsentation einen gültigen Fachwert darstellt. Solche Methoden sollten als statische Methoden an der Klasse selbst zur Verfügung gestellt werden.
Für jeden Datentyp oder jede Kombination von Datentypen, aus denen ein Fachwert konstruiert werden kann, sollte eine sondierende Methode zur Verfügung stehen, um im Vorfeld zu prüfen, ob ein Wert gültig ist oder nicht. Eine Implementation für einen Fachwert `Zinssatz` sieht also folgendermaßen aus:

```
public static boolean isValid(double value)
{
        return (value <= _maxValue && value >= _minValue);
}
```

Er könnte dann wie im nachstehenden Beispiel verwendet werden:

```
        boolean _gueltig = Zinssatz.isValid(42.0);
```

Die Methode `public static boolean isValid(String value)` überprüft für eine übergebene Repräsentation, ob es sich dabei um einen gültigen Wert handelt. Diese Methode sollte bei jedem Fachwert vorhanden sein, da sich jeder Fachwert als Zeichenkette darstellen läßt. Bei vielen Fachwerten empfiehlt es sich, noch geeignete Parsermethoden zu schreiben, allerdings wurde bei diesem einfachen Beispiel darauf verzichtet. Eine Implementation für einen Fachwert `Zinssatz`, bei der die oben genannte Methode als Basis verwendet wird, könnte folgendermaßen aussehen:

```
public static boolean isValid(String value)
{
     String tmp;
     boolean result = false;
```

Prüfung, ob der String ungleich ‚null' ist.

```
     if (value != null) {
```

`parseString` versucht den String in einen `double`-Wert zu konvertieren, welcher anschließend `isValid(int value)` auf seine Gültigkeit geprüft wird. Kann der String nicht konvertiert werden, wird eine Exception ausgelöst.

```
          try {
               result = isValid(parseString(tmp);
          } catch (NumberFormatException NFE){
               result = false;
          }
     }
     return result;
}
```

Die Verwendung einer solchen Methode würde dann wie nachstehend beschrieben aussehen:

```
     boolean _gueltig  = Zinssatz.isValid(”42.0 %”);
```

Eine solche Methode hat für den Fachwert `Beschaeftigungsart` das folgende Aussehen:

```
public static boolean isValid(String value)
{
     boolean result = false;
```

Prüfung, ob dieser String einen gültigen und definierten Fachwert darstellt.

```
     if (value != null) {
          result = (value.equals(_Beamter)
                    || value.equals(_Angestellter)
               . . .
     }
     return result;
}
```

3.2.1.6. Konstruktoren

Desweiteren sind die Konstruktoren der Fachwertobjekte von Interesse. Diese Konstruktoren sollten die einzigen Methoden sein, mit denen der Wert eines Fachwertobjektes gesetzt werden kann. Dies bedeutet, daß nach der Erzeugung keine wertverändernden Operationen zugelassen sind. Es muß also immer ein neues Fachwertobjekt erzeugt werden, wenn man einen neuen Wert haben möchte. Durch diese Art der Implementation wird die geforderte Wertsemantik umgesetzt.
Um ein Objekt zu erzeugen, muß noch ein Objektkonstruktor gerufen werden, welcher mit dem Namen der Klasse bezeichnet ist. Da dieser unter Java immer am Objekt

aufgerufen wird, führt dies zu Problemen bei der Verwaltung von Fachwerten. Wird diese Methode gerufen, wird sofort ein Objekt erzeugt. Dies kann zwar in die `WeakHashMap` eingefügt werden, aber dieses Objekt kann nicht, wie in der Methode `registeredValue` vorgesehen, automatisch ersetzt werden. Würde der Konstruktor so verwendet werden, würde er, wenn ein Fachwertobjekt mit einem bestimmten Wert noch nicht vorliegt, dieses mit der Methode `registeredValue` in die `weakHashMap` einfügen. Liegt ein Objekt mit diesem Wert aber schon vor, wird dieses zwar eingefügt, es kann aber nicht durch das alte Objekt ersetzt werden, da der Rückgabewert eines Konstruktors der zur Zeit erzeugte Wert ist. Wenn man so etwas tun wollte, müßte es unter Java möglich sein, folgendes zu schreiben:

```
public Zinssatz (double value)
{
     _value = value;

     // Benutzung der Verwaltungsmethode
     this = registeredValue(this);
}
```

Hier würde bei der Übersetzung die syntaktische Prüfung scheitern.

Eine Alternative wäre es dem Benutzer die Verwaltung der Fachwertobjekte zu überlassen, doch das hätte zur Folge, daß der Anwender einen erheblichen Mehraufwand leisten müßte. Aus diesem Grund muß der Objektkonstruktor über eine zusätzliche Klassenmethode benutzt werden. Um auszuschließen, daß er von außen benutzt wird, sollte er als `protected` deklariert werden.
Den Konstruktoren, welche als Klassenmethoden implementiert werden, wird bei der Erzeugung eines Fachwertobjektes eine Repräsentation des betreffenden Fachwertes übergeben, worauf sie ein Fachwertobjekt als Ergebnis liefern. Allerdings gibt es für die meisten Fachwerte viele verschiedene Arten von Darstellungen (z.B. für einen Fachwert `Datum`: „31.12.1999“, „31. Dezember 1999“, 31121999, 1999123 usw.), welche alle jeweils den gleichen Wert bezeichnen. Es ist also nur unter großem Aufwand möglich, für jeden dieser Fälle einen Konstruktor zur Verfügung zu stellen oder die vorliegenden Konstruktoren so zu verändern, daß alle möglichen Fälle behandelt werden können. Deshalb sollte wenn möglich nur für die kanonische Form eines Fachwertes ein Konstruktor existieren. Es sollte aber trotzdem immer möglich sein, aus einer Zeichenkette (String) einen Fachwert zu erzeugen, da die Darstellung als Zeichenkette für alle Arten von Fachwerten möglich ist. Für diese Zeichenkette sollte aber nur eine gültige Form existieren. Werden vom Anwender der Fachwerte noch weitere Repräsentationen benötigt, so muß er die hierfür notwendige Transformation selber durch einen Wrapper realisieren.
Bei der Erzeugung ist darauf zu achten, daß die Konstruktoren selbstverständlich nur aus gültigen Repräsentationen einen Fachwert erzeugen. Also sollte vom Anwender zuvor mittels einer sondierenden Methode die Gültigkeit seiner Repräsentation geprüft werden. Bei der Erzeugung von Fachwerten wird mit Hilfe von Vorbedingungen (Assertions) die Gültigkeit nochmals geprüft. Bei einer ungültigen Repräsentation müßte eigentlich ein undefinierter Wert erzeugt werden, stattdessen wird eine Exception geworfen, damit der Anwender der Fachwerte sofort darauf hingewiesen wird. Sollte aber dennoch ein undefinierter Wert benötigt werden, so kann dieser explizit über einen gesonderten Konstruktor erzeugt werden. Dieser wird nachstehend erläutert. Die eigentliche

Erzeugung findet dann bei einem gültigen Wert über den normalen Konstruktor der Fachwertklasse statt. Damit Fachwertobjekte nicht mehrfach erzeugt werden, was dem ”flyweight/factory”-Entwurfsmuster widersprechen würde, wird durch das erzeugte Objekt an die in der Basisklasse vordefinierte Methode `registeredValue(DomainValue dv)` übergeben, welche noch nachprüft, ob dieser Wert schon vorhanden ist. Ist er schon vorhanden, wird der schon erzeugte Wert zurückgeliefert, ansonsten wird der übergebene Wert eingetragen und als Ergebnis zurückgegeben. Dies bedeutet also, daß jede erzeugende Funktion nach dem Aufruf des eigentlichen Objektkonstruktors die Methode `registeredValue` ruft.
Der Klassenkonstruktor sieht unter Java wie folgt aus:

```
public static Zinssatz newValue(double value)
{
     Zinssatz    dvValue;
```

Einfügen des neuen Fachwetrobjektes in die Liste bisher erzeugten Fachwerte unter Benutzung der Verwaltungsmethode.

```
Contract.require(isValid(value));
```

Aufruf des Objektkonstruktors

```
dvValue = new Zinssatz(value);
```

Einfügen des neuen Fachwertobjektes in die Liste bisher erzeugten Fachwerte unter Benutzung der Verwaltungsmethode.

```
return  registeredValue(dvValue);
}
```

Im Objektkonstruktor müssen nun nur noch die Werte der Exemplare gesetzt werden. Da aber die Fachwertobjekte auch in der Lage sein müssen, den undefinierten Wert darzustellen, muß neben den Variablen, welche den Wert enthalten, noch eine weitere Variable vorhanden sein, welche anzeigt, ob der Wert definiert ist. Am besten eignet sich hierfür ein `Boolean`.

```
private double  _Zinssatz;
private boolean _defined;

private Zinssatz (double value)
{
     this._Zinssatz    = value;
     this._defined     = true;
}
```

Diese Methoden werden nun von einem Anwender folgendermaßen verwendet:

```
Zinssatz    _dvZinssatz;
double      value = 42.0;
```

Prüfung der Vorbedingungen

```
if (Zinssatz.isValid(value) {
```

Aufruf des Klassenkonstruktors

```
        _dvZinssatz = Zinssatz.newValue(value);
    } else {
        ...
```

In der folgenden Graphik wird nochmals der Ablauf bei der Erzeugung von Fachwerten

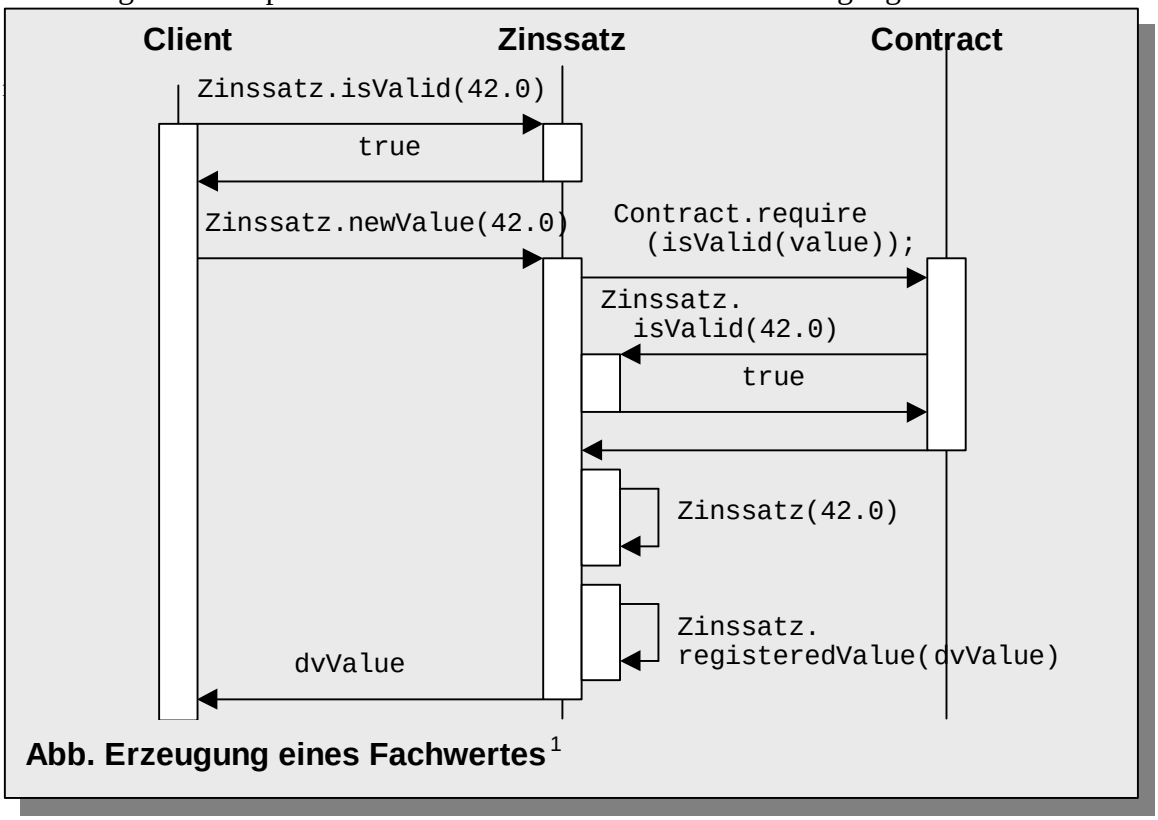


Abb. Erzeugung eines Fachwertes[1]

Neben der oben beschriebenen Art von Konstruktoren muß es aber auch noch die Möglichkeit geben, über einen anderen Konstruktor, welcher keine Argumente erhält, undefinierte Fachwertobjekte explizit zu erzeugen. Eine Gültigkeitsprüfung entfällt hierbei natürlich. Beim aufgerufenen Objektkonstruktor muß dann nur der Wert von `_defined` auf `false` gesetzt werden.

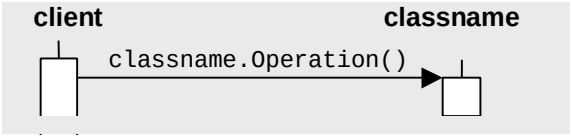
```
public static Zinssatz newValue()
{
    Zinssatz    dvValue;
```

Aufruf des Objektkonstruktors für undefinierte Fachwerte

```
    dvValue = new Zinssatz();
```

Einfügen des neuen Fachwertobjektes in die Liste bisher erzeugten Fachwerte unter Benutzung der Verwaltungsmethode.

[1] Anmerkung zur Notation:
Aufrufe von Klassenmethoden werden in dieser Arbeit folgendermaßen dargestellt:



```
        return registeredValue(dvValue);
    }
```

Der Default-Konstruktor:

```
    protected Zinssatz ()
    {
        this._defined     = false;
    }
```

Der Klassen- und der Objektkonstruktor für den Fachwert `Beschaeftigungsart` stellen sich folgendermaßen dar:

Der Klassenkonstruktor:

```
    public static Beschaeftigungsart  newValue(String value)
    {
        Beschaeftigungsart      dvValue;
```

Hier wird geprüft, ob die übergebene Repräsentation einen definierten Fachwert darstellt.

```
        Contract.require(isValid(value));
```

Aufruf des Objektkonstruktors

```
        dvValue = new Beschaeftigungsart (value);
```

Einfügen des neuen Fachwertobjektes in die Liste bisher erzeugten Fachwerte unter Benutzung der Verwaltungsmethode.

```
        return  registeredValue(dvValue);
    }
```

Der Objektkonstruktor:

```
    private double  _ Beschaeftigungsart;
    private boolean _defined;

    private Beschaeftigungsart (String value)
    {
        this._ Beschaeftigungsart     = value;
        this._defined                 = true;
    }
```

Die Default-Konstruktoren sind, bis auf den Typ des erzeugten Objektes, identisch mit denen der `Zinssatz`-Klasse, und werden deshalb nicht aufgeführt.

3.2.1.7. Repräsentationen von Fachwerten und Selektoren

Ebenso wie bei den Konstruktoren, wo Repräsentationen in Fachwerte transformiert werden, muß es Methoden geben, die eine Rücktransformation in eben diese Repräsentationen durchführen. Auch hier gibt es das Problem, daß nicht alle möglichen

Darstellungsformen berücksichtigt werden können. Deshalb sollte man sich bei der Darstellung des Fachwertes nach außen auf die Darstellung der kanonischen Form beschränken. Es sollte einsichtig sein, daß diese Methoden am eigentlichen Fachwertobjekt angesiedelt sind. Im Normalfall handelt es sich bei den Ergebnistypen um die Standarddatentypen wie zum Beispiel `int` und `String`. Wenn es sich aber bei der Normalform um eine Kombination aus anderen Fachwerten handelt, muß es möglich sein, auch diese wiederum als Ergebnis zu erhalten.
Sollte ein undefinierter Fachwert nach einer Repräsentation als Standarddatentyp gefragt werden, so wird eine Vorbedingung verletzt, wonach nicht auf undefinierte Fachwerte in dieser Form zugegriffen werden darf, und es wird eine Exception geworfen. Sollte der Ergebnistyp ein Fachwert sein, so ist das Ergebnis wiederum ein undefinierter Wert. Um dies zu vermeiden, kann mit der Methode `isDefined()` überprüft werden, ob es sich bei dem betreffenden Fachwert um einen definierten Wert handelt. Als Ergebnis wird der Wert von `_defined` zurückgeliefert, welcher bei der Erzeugung des Fachwertes gesetzt wird.
Eine mögliche Implementation für die vorliegende Klasse `Zinssatz` könnte dann wie folgt aussehen:

```
public double toDouble()
{
```

An dieser Stelle wird geprüft, ob die Vorbedingungen erfüllt sind und der Wert definiert ist.

```
    Contract.require(isDefined());
```

Sind die Vorbedingungen erfüllt, wird der Wert als double zurückgegeben.

```
    return _value;
}
```

Da die kanonische Form von ihren Datentypen für alle Fachwerte unterschiedlich ist, muß zusätzlich zu dieser Darstellungsform noch die Möglichkeit bestehen, Fachwerte durch einen einheitlichen Datentyp darzustellen, um beispielsweise Defaultimplementierungen für die Darstellung von Fachwerten in Fenstersystemen zu erstellen. In den vorangegangenen Kapiteln wurde festgestellt, daß sich alle Fachwerte, auch wenn sie undefiniert sind, über Zeichenketten darstellen lassen. Deshalb ist es auch nicht nötig, bei einer solchen Darstellung die Vorbedingungen explizit zu prüfen. Hierbei sollte es aber für jeden möglichen Fachwert nur jeweils eine eindeutige Zeichenkette geben. Eine Methode, welche den Wert eines `Zinssatz`-Objektes als Zeichenkette liefert, sieht also wie folgt aus:

```
public String toString()
 {
      String Result;
      int   decimalPoint = Double.toString(_value).
                                              indexOf('.');
```

Wenn der Wert undefiniert ist, wird die Konstante `_undefinedValue` als Ergebnis zurückgegeben.

```
      if (!isDefined()) {
            return _undefinedValue;
```

Ansonsten wird der wert als Zeichenkette auf zwei Nachkommastellen genau ausgegeben.

```
        } else {
               if
              (Double.toString(_value).substring(decimalPoint).-
              length()  > 2){
                    Result = Double.toString(_value).-
                    substring(0,decimalPoint + 2);
              } else  {
                    Result = Double.toString(_value);
              }
        }
        return Result + " %";
}
```

Um eine einheitliche Zeichenkettendarstellung des undefinierten Wertes zu ermöglichen, wurde hierfür eine String-Konstante `_undefinedValue` deklariert. Die Methode `toString()` für die Fachwertklasse `Beschaeftigungsart` sollte sich leicht aus der eben beschriebenen Methode für die Klasse `Zinssatz` ableiten lassen.
Für zusammengesetzte Fachwerte ist es darüber hinaus noch notwendig, Selektoren zu haben, die als Ergebnis die einzelnen Bestandteile des Fachwertes liefern. Für einen Fachwert `Datum`, der sich aus den Fachwerten `Tag`, `Monat`, `Jahr` zusammensetzt, wäre es zum Beispiel sinnvoll, wenn man sich das Element vom Typ `Tag` einzeln geben lassen könnte.

3.2.1.8. Fachliche Operationen

Neben den oben genannten Methoden gibt es noch solche, die Operationen auf Fachwerten realisieren. Alle diese Methoden, wie zum Beispiel eine Addition von Fachwerten, werden an den eigentlichen Fachwertobjekten angesiedelt. Bei der Realisierung der Methoden wurde sich, wie unter 3.2 erwähnt, am Curry-Prinzip aus der funktionalen Programmierung orientiert. Diese Verfahrensweise gilt für Operationen, bei denen ein Fachwert erzeugt wird und für solche, bei denen ein Standarddatentyp das Ergebnis ist. Bei jeder dieser Operationen muß anhand einer sondierenden Operation zuvor geprüft werden können, ob das Ergebnis ein definierter Wert ist oder nicht. Sollte trotz vorheriger Prüfung eine fachliche Operation mit undefinierten Fachwerten als Parametern aufgerufen werden, wird eine Exception ausgelöst.
Um den Anwender die Möglichkeit zu geben, die Vorbedingungen prüfen zu können, sollte zu jeder dieser Operationen eine sondierende Methode zur Verfügung stehen, mit der überprüft werden kann, ob das Ergebnis eine definierter Wert ist. Die sondierende Methode zu einer Funktion, welche zwei Zinssätze addiert, würde also folgendermaßen aussehen:

```
public boolean isResultOfAddDefined(Zinssatz dvZinssatz)
{
```

Prüfung, ob beide Werte definiert sind das Ergebnis ein gültiger Wert ist.

```
        return(isDefined() && dvZinssatz.isDefined &&
        isValid(toDouble() + dvZinssatz.toDouble());
}
```

Der Quelltext für eine Methode, welche zwei Zinssätze addiert, sieht dann so aus :

```
public Zinssatz add(Zinssatz value)
{
    An dieser Stelle wird geprüft, ob die Vorbedingungen erfüllt sind.

      Contract.require(isResultOfAddDefined(value));

    Aufruf des Konstruktors

      return newValue(toDouble() + value.toDouble());
}
```

Verwendung:

```
Zinssatz    dvZinssatz1,dvZinssatz2,dvZinssatz3;

if (dvZinssatz1.isAddValid(dvZinssatz2)) {
      dvZinssatz3 = dvZinssatz1.add(dvZinssatz2);
}
...
```

Neben diesen fachlichen Operationen muß für jeden Fachwert der Test auf Gleichheit möglich sein, da schon bei der Definition von Werten gefordert wird, daß Werte auf Gleichheit geprüft werden können. Diese Methode nimmt eine Sonderstellung ein, da wir hier die Fachwertobjekte unter ihrer Oberklasse `Object` verwenden müssen. Der Grund hierfür liegt darin, daß diese Methode für die Verwaltung der Fachwerte in der `WeakHashMap` (siehe Abschnitt „Verwaltung von Fachwerten“) benötigt wird. Da die Fachwerte unter der Oberklasse verwendet werden, muß in der Methode noch abgefragt werden, ob die Klassen beider Fachwertobjekte gleich sind.

```
public boolean equals(Object value)
{
    Prüfung, ob ein Downcast auf den Typ der vorliegenden Klasse möglich ist.

      Contract.require(Zinssatz.class.isAssignableFrom(
                                    value.getClass(), this ));
    Downcast-Anweisung

      Zinssatz dv = (Zinssatz) value;

    Hier erfolgt eine  Prüfung, ob beide Fachwertobjekte definiert sind.

      Contract.require(isDefined() && dv.isDefined(), this);
      // Vergleich der String-Repräsentationen und der Klassen
      return dv.getClass() == getClass() && dv.toString().
                                              equals(toString());
}
```

Verwendung:

```
      Zinssatz    dvZinssatz1, dvZinssatz2;

      if (dvZinssatz1.equals(dvZinssatz2)) { ...
```

Da auf dem Wertebereich der Klasse `Zinssatz` eine Ordnungsrelation definiert ist, gehören zu den fachlichen Operationen natürlich auch Vergleichsoperationen wie ”>” und ”<”. Auf eine genaue Erläuterung wird hier verzichtet, da sich beide Operationen in Hinsicht auf ihre Implementation unter Java bis auf die Tatsache, daß die Argumente vom Typ `Zinssatz` statt vom Typ `Object` wären, nur unwesentlich von der Methode `equals` unterscheiden.
Für die Fachwertklasse `Beschaeftigungsart` ist keine Ordnungsrelation definiert, deshalb kann für sie nur der Test auf Gleichheit implementiert werden. Allerdings läßt sich auch diese Methode einfach aus der gerade beschriebenen Methode `equals` ableiten.

3.2.2. Aufbau der Basisklasse

Vergleicht man die Schnittstelle des Fachwertes `Zinssatz` mit der des Fachwertes `Beschaeftigungsart` oder der anderer Fachwerte, stellt man fest, daß einige Methoden in allen Klassen vorkommen. Bei diesen Methoden handelt es sich um die Verwaltungsmethode und deren zugehörige Behälterstruktur, die Methode zur Darstellung eines Fachwertes als Zeichenkette sowie die sondierende Methode, welche angibt, ob ein Fachwert definiert ist. Aus diesem Grund bietet es sich an, diese Methoden in einer Basisklasse zur Verfügung zu stellen. Das Ziel hierbei ist es, eine möglichst einfache und einheitliche Konstruktion und Verwendung von Fachwerten im JWAM-Rahmenwerk zu ermöglichen. Die nachfolgende Abbildung stellt die sich daraus ergebende Klassenhierarchie dar.

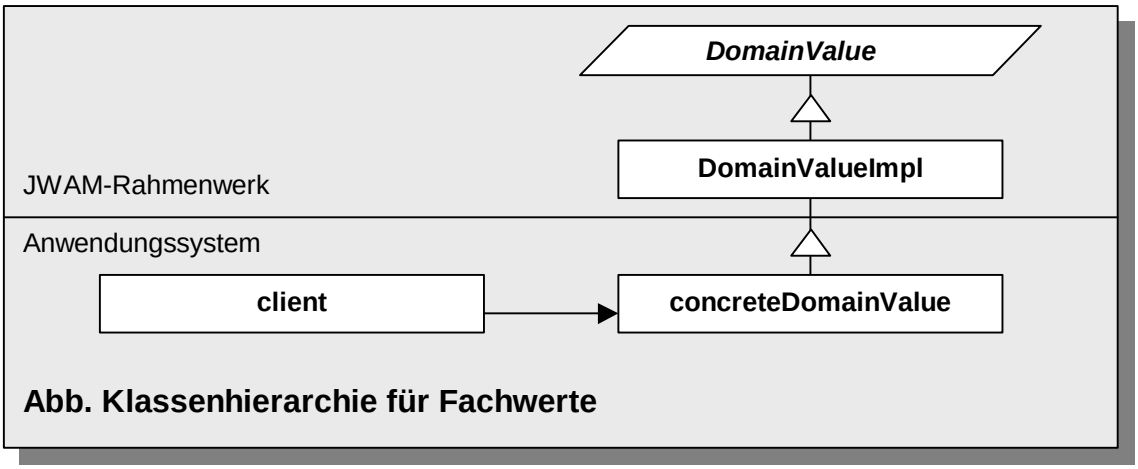


Abb. Klassenhierarchie für Fachwerte

Im JWAM-Rahmenwerk werden alle Elemente standardmäßig über ein Interface und eine abstrakte Implementation modelliert. Aus diesem Grund gibt es für die Fachwerte einmal das Interface `DomainValue`, welches die abstrakte Schnittstelle für Fachwerte beschreibt. Die abstrakte Klasse `DomainValueImpl` ist die Basisklasse für alle Fachwerte und enthält Default-Implementationen für die im Interface `DomainValue` deklarierten Methoden. Über das Interface und die Basisklasse werden für die Subklassen elementare Methoden zur Verfügung gestellt.
Alle Methoden, welche an die Subklassen vererbt werden sollen und nicht `public` sind, müssen als `protected` deklariert werden, da sie sonst unter Java nicht in den Subklassen verwendet werden können.
Im nachfolgenden Abschnitt werden die Schnittstelle und die Klassenvariablen sowie die damit verbundenen Entwurfsentscheidungen erläutert. Im Anhang finden sich vollstän-

dige Schnittstellenbeschreibungen für das Interface `DomainValue`, die Basisklasse `DomainValueImpl` und die angepaßte Beispielklasse `Zinssatz`.

3.2.2.1. Sondierende Methoden

Wie schon im Kapitel 2 festgestellt sollten die Wertebereiche aller Fachwerte den undefinierten Wert enthalten. Da also die Fachwertobjekte aller Klassen den undefinierten Wert annehmen können, muß auch immer geprüft werden können, ob ein Fachwert definiert bzw. undefiniert ist. Aus diesem Grund wurde in die Basisklasse eine Methode `isDefined()` aufgenommen, die diese Überprüfung leistet Der Standardrückgabewert dieser Methode ist `True`, da davon ausgegangen wird, daß in der Regel definierte Fachwertobjekte erzeugt werden. In einer Subklasse sollte diese Methode immer auf geeignete Weise überschrieben werden. Dies bedeutet, daß, wie in den Abschnitten 3.2.1.6. und 3.2.1.7 gezeigt, in den Fachwertobjekten selber vermerkt werden muß, ob ein Wert definiert ist.

3.2.2.2. Repräsentationen von Fachwerten

Wie schon festgestellt wurde, lassen sich alle definierten bzw. undefinierten Fachwerte als Zeichenketten darstellen. Deshalb kann die Methodenschnittstelle `toString`, welche den Zugriff auf die Zeichenkettenrepräsentation eines Fachwertes ermöglicht, ebenfalls in die Basisklasse und in das zugehörige Interface übernommen werden. Die Default-Implementation liefert als Ergebnis die Zeichenkette „DomainValue“. Auch diese Methode sollte in den Subklassen überschrieben werden, da in der Basisklasse nicht entschieden werden kann, wie die Zeichenkettendarstellung eines konkreten Fachwertes aussieht

3.2.2.3. Verwaltungsmethoden der Fachwertklassen

Neben diesen Methoden für den fachlichen Umgang eignet sich besonders die Verwaltungsmethode für eine Implementation in der Basisklasse, da in allen Fachwertklassen für die Verwaltung der einzelnen Objekte das „flyweight/factory“-Entwurfsmuster, durch welches eine minimale Anzahl an Objekten garantiert wird, angewandt wird. Daß heißt, daß sich alle Klassen in ihrer Fabrikkomponente gleichen. Damit die Verwaltungsmethode `registeredValue` in die Basisklasse übernommen werden kann, müssen einige Anpassungen vorgenommen werden. Für diese Methode muß, wie unter 2.1.2.4. beschrieben, ein Behälter vom Typ `WeakHashMap` existieren. Dieser muß ebenfalls in die Basisklasse mit der folgenden Implementation übernommen werden.

```
private static transient WeakHashMap _domainValueHashMap
                                          = new WeakHashMap();
```

Daneben müssen die Methoden `equals(Object value)` und `hashCode()` in die Basisklasse aufgenommen werden. Bei beiden Methoden müssen die Fachwertobjekte nun unter dem Interface `DomainValue` verwendet werden. Bei der Methode `hashCode()` muß, um ein eindeutiges Ergebnis zu erhalten, noch die Klasse einbezogen werden. Die Methode `equals(Object value)` kann ohne weitere Änderungen übernommen werden, da hier schon vorher die Klassen der zu vergleichenden Objekte überprüft wurden. Die Methoden müssen dann wie nachstehend aussehen:

```
public int hashCode()
```

```
{

      int result = this.getClass().hashCode();
      result += (this.toString()).hashCode();
      return result;
}

public boolean equals (Object value)
{
      Contract.require(DomainValue.class.isAssignableFrom(
                              value.getClass()), this );

      DomainValue dv = (DomainValue) value;

      Contract.require(isDefined() && dv.isDefined(), this);

      return dv.getClass() == getClass() &&
                             dv.toString().equals(toString());
}
```

Diese Verwaltungsmethode `registeredValue` kann dann so, wie sie beim Beispiel `Zinssatz` erläutert wurde in die Basisklasse übernommen werden. Die einzigen Anpassungen, die vorgenommen werden müssen, sind die Änderung des Argumenttyps in `DomainValue` und die Deklaration der Methode als `protected`. Die Schnittstelle würde dann wie folgt aussehen:

```
protected static DomainValue registeredValue(DomainValue value)
{
```

Prüfung der Vorbedingungen :
Der übergebene Wert darf nicht gleich ‚null' sein.

```
  Contract.require(value != null);

  DomainValue resultValue;
```

Prüfung, ob das Objekt bereits vorhanden ist.

```
  if (_domainValueHashMap.containsKey(value)) {
```

Wenn bereits Objekt vorhanden ist, wird das vorhandene Objekt der Ergebnisvariable zugewiesen.

```
        resultValue = (DomainValue)
        _domainValueHashMap.get(value);
  } else {
```

Wenn bereits Objekt nicht vorhanden ist, wird das neue Objekt in die Liste aufgenommen und der Ergebnisvariable zugewiesen.

```
        _domainValueHashMap.put(value,value);
        resultValue = value;
  }
```

Prüfung der Nachbedingungen:
Der Rückgabewert darf nicht gleich ‚null' sein und muß gleich dem übergebenen Wert sein. Da die Fachwertobjekte aller Fachwertklassen in einer

Basisklasse verwaltet werden, muß zugesichert werden, daß die Klassen von übergebenen Wert und Rückgabewert übereinstimmen.

```
        Contract.ensure(resultValue != null);
        Contract.ensure(resultValue.equals(value));
        Contract.ensure(resultValue.getClass() ==
                                    value.getClass());
        return resultValue;
    }
```

Die Handhabung dieser Methode ist weiterhin so, wie sie unter 2.2.1.6. beschrieben wurde.

3.2.3. Fachwerte und Interaktionskomponenten

Nachdem in den vorangegangenen Abschnitten die Methoden von Fachwertobjekten und ihre allgemeine Handhabung beschrieben wurden, soll im folgenden ein Beispiel beschrieben werden, wie Fachwerte im JWAM-Rahmenwerk verwendet werden können.
Im JWAM-Rahmenwerk wird eine Unterteilung in Werkzeuge und Materialien vorgenommen, wobei die Werkzeuge wiederum in eine Funktions- und eine Interaktionskomponente unterteilt sind(siehe [Züllighoven 98]). Nur die Funktionskomponenten sollten Zugriff auf das Material haben, da in diesem das fachlichen Wissen um den Umgang mit den Materialien gekapselt wird, während die Interaktionskomponenten nur zur Darstellung der Materialien dienen. Das heißt, man benötigt geeignete Kommunikationsmittel, die den Informationsaustausch zwischen den beiden Komponenten ermöglichen. Man hat nun die Möglichkeit, die Materialien in den Funktionskomponenten in ihre Elemente zu zerlegen und diese an die Interaktionskomponenten weiterzugeben. Da bei dieser Weitergabe keine Seiteneffekte auftreten dürfen, stehen hier nur die Standarddatentypen zu Verfügung. Diese sind aber nur für einen Teil der Aufgaben wirklich brauchbar, da sie zu allgemein sind und die fachlichen Werte des Anwendungsbereiches nur ansatzweise modellieren. Auf der anderen Seite kann man einer Interaktionskomponente nicht gestatten, direkt auf das Material zuzugreifen, da sie über keinerlei Anwendungswissen verfügen darf (siehe[Züllighoven 98]). An dieser Stelle bieten sich die Fachwerte als Lösung an, weil sie erstens fachliche Konzepte des Anwendungsbereichs realisieren und somit über eine erheblich mächtigere Semantik als Standarddatentypen verfügen. Und zweitens verursachen Fachwerte keinerlei Seiteneffekte. Sie können also problemlos in der Kommunikation zwischen Funktions- und Interaktionskomponenten eingesetzt werden, ohne daß die Fachwerte für jede Kommunikation in elementaren Datentypen überführt und nachher wieder in Fachwerte transformiert werden müssen. An dieser Stelle könnte allerdings mit der Begründung, daß dadurch fachliches Wissen in die Interaktionskomponenten gelangen würde, gegen die Verwendung von Fachwerten argumentieren. Wenn man aber die Fachwerte lediglich als Erweiterung der bestehenden elementaren Datentypen betrachtet, bei denen man ja auch prüft, ob sie syntaktisch korrekt sind, sind hierbei nur bei der Art Modellierung der Fachwerte als Objekte Unterschiede erkennbar. So wird, wenn von einer Interaktionskomponenente ein `double`-Wert zurückgeliefert wird, auch vorher überprüft, ob der Wert gültig ist. Die Prüfung wird aber hierbei durch die Laufzeitumgebung durchgeführt und muß nicht wie bei den Fachwerten vom Anwender durchgeführt werden.
Es ist also ohne weiteres möglich, Fachwerte von den Funktionskomponenten an die Interaktionskomponenten weiterzugeben. Diese können die Fachwerte dann durch ihre

Interaktionsformen darstellen. Andererseits können auch Fachwerte von den Interaktionskomponenten oder Interaktionsformen aus den Eingaben der Klienten ohne Seiteneffekte erzeugt werden. Hierbei findet dann auch gleichzeitig eine kontextfreie Prüfung der Gültigkeit des Wertes statt (siehe Kapitel 2). Mehr als diese kontextfreie Prüfung darf an dieser Stelle aber nicht erfolgen, da sonst das Wertekonzept, wonach Werte unabhängig von ihrem Umfeld sind, verletzt würde. Die Fachwertobjekte können dann an die Funktionskomponenten weitergeben werden. Hierbei unterscheidet sich der Umgang nicht von der vorher geschilderten Erzeugung von Fachwerten. In den Funktionskomponenten müßte dann nur noch geprüft werden, ob der Fachwert in dem Kontext der jeweiligen Funktionskomponente gültig ist, und nicht mehr, ob es überhaupt ein gültiger Wert ist. Man könnte sogar für alle möglichen Fachwerte eine auf Strings basierende allgemeine Interaktions- und Präsentationsform anbieten, was zum Beispiel bei einfachen Eingabemasken die Erstellung dieser erheblich beschleunigen kann, ohne daß der fachlich korrekte Umgang mit dem Fachwerten verlorengeht. Für spezielle Anforderungen sollte aber immer eine speziell auf den Fachwert zugeschnittene Interaktionsform geschaffen werden.
Der grundsätzliche Umgang mit Fachwerten in der Funktions- und Interaktionskomponenten sieht demnach folgendermaßen aus:

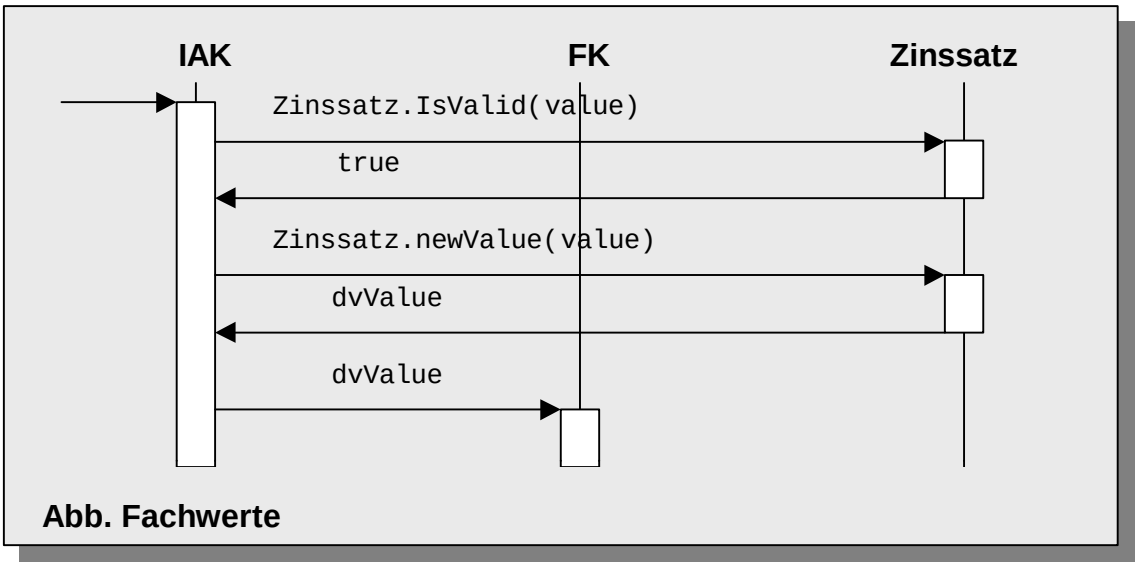


Abb. Fachwerte

3.3. Fachwerte und Persistenz

Um Fachwerte für den Einsatz in konkreten Anwendungssystemen nutzbar zu machen, muß es die Möglichkeit geben, Fachwerte in Datenbanken oder Dateien persistent zu halten. Bei der vorliegenden Implementation wurden zwei mögliche Arten zur Realisierung der Persistenz von Fachwerten umgesetzt. Als erstes wurde die Serialisierung von Fachwerten, wie sie unter Java als standardmäßiger Persistenzmechanismus für alle Objekte möglich ist, implementiert. Hierbei sind aufgrund des ausgewählten ”flyweight/factory”-Entwurfsmusters einige Besonderheiten zu beachten, damit die Fachwerte auch korrekt an der jeweiligen Fachwertklasse erzeugt und verwaltet werden können. Als zweites wurde als eine weitere Möglichkeit zur Persistenz das „Atomizer-Pattern“ des JWAM-Rahmenwerks an die Erfordernisse der Fachwerte angepaßt. Nähere Informatio-

nen über die notwendigen Schnittstellen finden sich in den Beschreibungen zum `Atomizer` in der Dokumentation des JWAM-Rahmenwerks des Arbeitsbereiches Softwaretechnik. Beide Mechanismen werden im Folgenden beschrieben.

3.3.1. Fachwerte und die Standardserialisierung unter Java

Die Objektserialisierung (siehe [Flanagan 97]) gehört unter Java zu den Standardmöglichkeiten, ein Objekt mit seinem Zustand einschließlich der enthaltenen Objekte in einen Ausgabe-Stream zu schreiben und dieses Objekt zu einem späteren Zeitpunkt wieder herzustellen. Das Wiederherstellen des Objektes geschieht, indem man den serialisierten Zustand des Objektes über einen Eingabe-Stream wieder einliest.
Zur Serialisierung eines Objektes wird dieses an die `writeObject`-Methode eines `ObjectOutputStream`-Objektes übergeben, welcher das Objekt mit seinem Variablen in einen `FileOutputStream` schreibt. Verweist eine Variable auf ein anderes Objekt, wird vom `ObjectOutputStream` automatisch die `writeObject`-Methode mit diesem Objekt als Argument aufgerufen, um auch dieses zu serialisieren.
Bei der Deserialisierung wird dieser Vorgang einfach umgekehrt. Ein Objekt wird aus einem Daten-Stream gelesen, indem die `readObject`-Methode eines `ObjectInputStream`-Objektes aufgerufen wird. Auf diese Weise wird der Zustand des Objektes wiederhergestellt, in dem es sich befand, als es serialisiert wurde. Verweist dies Objekt auf andere Objekte, werden diese ebenfalls rekursiv wiederhergestellt.
Sollen nicht alle Variablen eines Objektes serialisiert werden, ist es möglich über das `Serializable`-Interface eigene `writeObject`- und `readObject`-Methode an dem zu serialiserenden Objekt zu realisieren. Über diese Methoden wird dann festgelegt, welche Bestandteile des Objektes zu serialisieren sind. Wird nun das Objekt an ein `ObjectOutputStream`-Objekt übergeben, wird, anstatt den Zustand aller Variablen zu serialisieren, die `writeObject`-Methode des Objektes aufgerufen und nur die dort aufgeführten Bestandteile werden in den Ausgabe-Stream geschrieben. Genauso verhält es sich bei der `readObject`-Methode, bei der nur die bezeichneten Bestandteile eingelesen werden.
Wenn man nun Fachwertobjekte serialisieren will, braucht man diese nur wie oben beschrieben unter Aufruf der `writeObject`-Methode an ein `ObjectOutputStream`-Objekt zu übergeben. Ein Problem ergibt sich erst, wenn man das serialisierte Fachwertobjekt wieder aus dem Eingabe-Stream einlesen will. In diesem Fall wird das Fachwertobjekt zwar korrekt wiederhergestellt, aber es wurde nicht über die mit `registeredValue` in die Liste der erzeugten Fachwerte aufgenommen. Auch wenn, wie oben erläutert, die Fachwerte über das Interface `Serializable` die `writeObject`- und die `readObject`-Methoden ererben lassen und diese reimplementieren, ist dieses Problem dennoch nicht beseitigt, da die Objekte direkt nach der Deserialisierung schon erzeugt sind und diese dann nicht, wie dies in der `registeredValue` -Methode geschieht, durch einen schon bestehenden Wert ausgetauscht werden können. Das heiß, falls ein Fachwertobjekt schon besteht und ein anderes mit dem gleichen Wert aus einem Eingabe-Stream ausgelesen wird, liegt der gleiche Wert doppelt im System vor, womit das Konzept des „flyweight/factory“-Entwurfsmusters aufgehoben wäre. Um dieses Problem zu umgehen, wurde neben den Methoden noch eine weitere Methode implementiert, durch die dieses Problem umgangen wird. Doch zunächst soll die Serialisierung von Fachwerten an einem Beispiel beschrieben werden:

```
private void writeObject(ObjectOutputStream out)
throws IOException
{
      out.writeDouble(_value);
      out.writeBoolean(_defined);
}
```

Die Implementation der Methode `writeObject` ist in der Regel nicht nötig. Sie wurde hier aber dennoch genannt, um den Leser vor Augen zu führen was an dieser Stelle passiert. Im Normalfall die Default-Implementation aus.
Ein Methodenaufruf in Java hätte folgende Form:

```
FileOutputStream in = new FileOutputStream("oss.dat" );

ObjectOutput s  = new ObjectOutputStream(out);
. . .
Zinssatz    Zinssatz = newValue(42.0);
. . .
s.writeObject(Zinssatz);
```

In der Graphik wird nochmals die genaue Form der Interaktion zwischen den Objekten und Klassen deutlich gemacht:

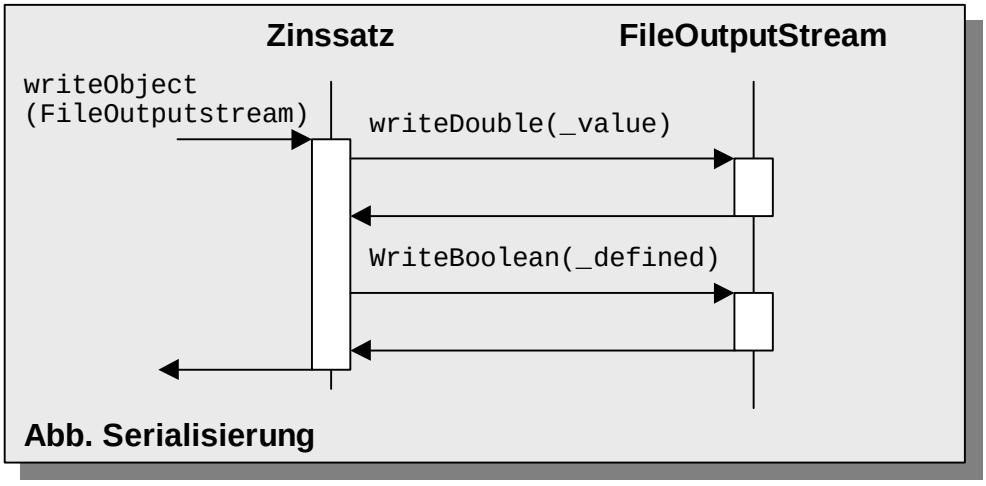


Abb. Serialisierung

Um nun das Problem bei der Deserialisierung zu lösen, muß neben der Methode `readObject` zusätzlich eine `static`-Methode implementiert werden, an welche der Eingabe-Stream übergeben wird und welche dann mit der Methode `readObject` die Werte des Fachwertes einliest. Nach der Erzeugung wird dieser Fachwert dann mit der üblichen `registeredValue`-Methode in die Liste der erzeugten Fachwerte aufgenommen und gegebenenfalls durch einen schon bestehenden Fachwert ersetzt. Nach diesen Schritten wird der Fachwert als Ergebnis an den Anwender zurückgegeben.

```
public static Zinssatz readDomainValue(ObjectInput ss)
throws IOException,ClassNotFoundException
{
      //Lesen des Objektes
      Zinssatz dvZinssatz = (Zinssatz) ss.readObject();
      // Eintragen in die Liste der erzeugten Fachwerte
      return (Zinssatz) registeredValue(dvZinssatz);
}
```

```
private void readObject(ObjectInputStream in) throws IOException
{
      _value    = (double) in.readDouble();
      _defined = (boolean) in.readBoolean();
}
```

Wie bei der `writeObject`–Methode wäre auch hier die Default-Implementation ausreichend gewesen.
Ein Aufruf der `readDomainValue`-Methode hat dann die folgende Form:

```
FileInputStream in = new FileInputStream( "oss.dat" );

ObjectInput ss = new ObjectInputStream( in );
      . . .
Zinssatz dvZinssatz = Zinssatz.readDomainValue(ss);
```

Die folgende Abbildung verdeutlicht die bei der Deserialisierung ablaufenden Schritte:

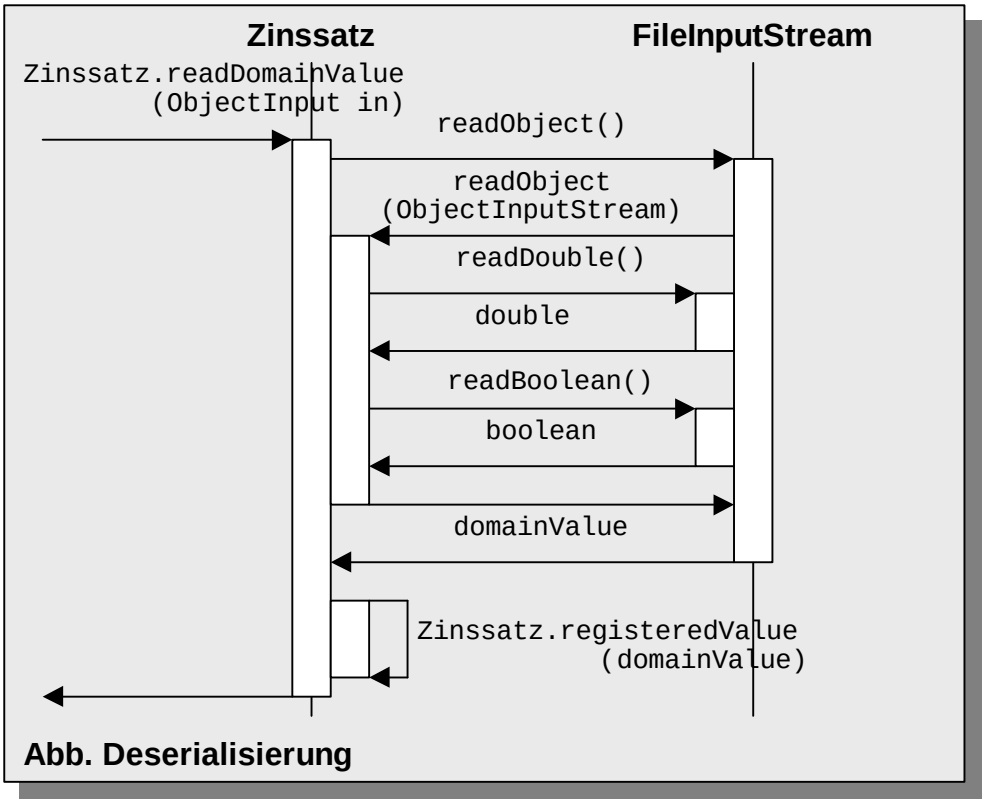


Abb. Deserialisierung

Aus den oben genannten Anpassungen ergibt sich folgende erweiterte Klassenhierarchie für Fachwerte im JWAM-Rahmenwerk:

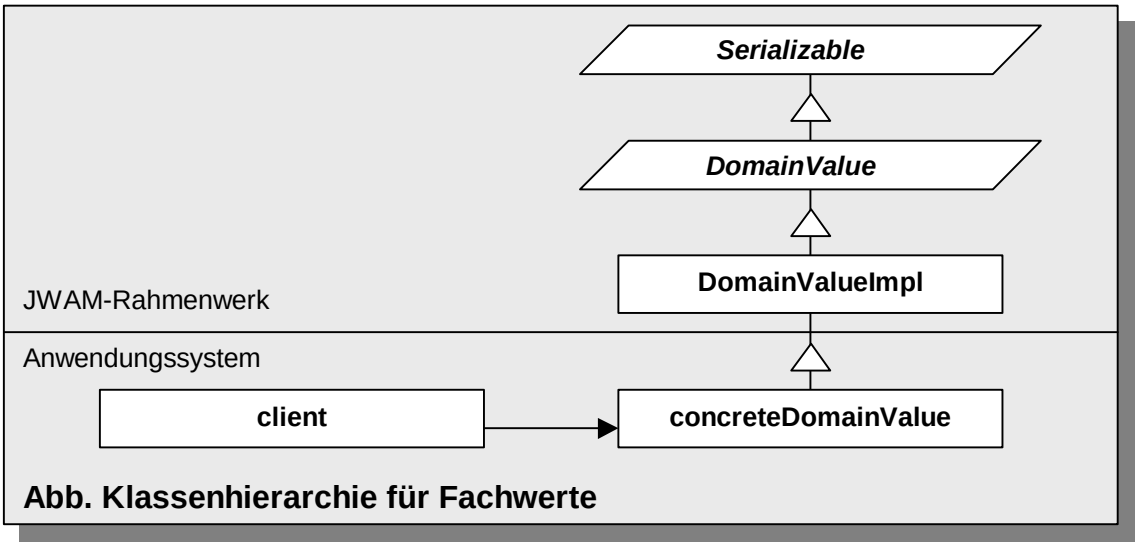


Abb. Klassenhierarchie für Fachwerte

3.3.2. Fachwerte und Serialisierung im JWAM-Rahmenwerk

Neben der Java-Standardserialisierung gibt es im JWAM-Rahmenwerk noch die Möglichkeit, Objekte mittels eines Atomizers persistent zu machen. Dieser Atomizer basiert auf dem Serialisierer aus [Züllighoven 98]. Für das JWAM-Rahmenwerk wurde der Serialisierer als Atomizer oder Atomisierer und die Schnittstelle serialisierbar als atomizable bezeichnet, um Verwechslungen mit den Bezeichnungen für die Standardserialisierung auszuschließen. Der Atomisierer ermöglicht es, beliebig komplexe Objektzusammenhänge in Dateien oder relationale Datenbanken zu Schreiben und wieder auszulesen. Hierbei wird durch den Atomisierer das Wissen um technische Strukturen wie zum Beispiel Datenbanken, vor den anwendungsfachlichen Klassen verborgen. Auf diese Weise haben Änderungen bei Persistenzmedien keine Auswirkungen auf die Objekte, so daß diese zum Beispiel beim Wechsel der Datenbank nicht angepaßt werden müssen. Um dies umzusetzen, wird das Lesen und das Schreiben an eigens hierfür vorgesehene Klassen delegiert. Für den Vorgang des Lesens und des Schreibens wird jeweils eine eigene Klasse für jede Datenbank realisiert. Jeder konkrete Leser oder Schreiber implementiert eine abstrakte Klasse. Objekte, welche serialisiert werden sollen, müssen, ähnlich wie bei der Standardserialisierung, eine bestimmte Schnittstelle haben. Bei Züllighoven wird diese als serialisierbar bezeichnet.

Im JWAM-Rahmenwerk werden die abstrakten Klassen als `Reader-` und `Writer-`Interfaces bezeichnet. Die Klassen, welche diese Interfaces implementieren, sind die Klassen `RDBMSReader` und `RDBMSWriter`. Über die geeigneten Methoden aus diesen Klassen ist es möglich, Objekte und Standarddatentypen in eine relationale Datenbank zu schreiben bzw. aus dieser auszulesen. Weitere Informationen zu diesem Thema sollten der Dokumentation des JWAM-Rahmenwerks entnommen werden.

Bei dem hier vorliegenden Lösungsvorschlag für das Lesen und Schreiben von Fachwerten mittels des Atomizers in relationale Datenbanken wurde darauf Wert gelegt, alle dafür benötigten Methoden schon im Rahmenwerk festzulegen, um den Anwendern möglichst viel Arbeit abzunehmen. An den `Reader-` bzw. `Writer`-Objekten müssen zusätzlich zu den bestehenden Methoden, welche Objekte und Standarddatentypen

persistent machen können, die Methoden `readValue und writeValue` implementiert werden. Diese Methoden sind für alle möglichen Fachwerte anwendbar. An dieser Stelle sollte noch gesagt werden, daß es neben diesem Ansatz gibt es natürlich noch weitere Möglichkeiten, Fachwerte über den Atomizer persistent zu machen. Beispielsweise könnte jeder Fachwert seine eigene Persistenzmethoden erhalten, dies ist aber für eine Implementation in einem Rahmen nicht denkbar, da hier möglichst vielseitig anwendbare Ansätze modelliert werden sollen.
Für einen Fachwert, der durch eine solche Schnittstelle persistent gemacht werden soll, muß der konkrete Fachwert das Interface `ValueAtomizable` implementieren. In diesem Interface sind Schnittstellen für die Methoden `writeTo` und `readFrom` deklariert. Bei der Methode `readFrom` handelt es sich wie bei der Methode `readDomainValue` um eine `static`-Methode, da es sonst auch hier zu den gleichen Problemen kommen würde, wie unter 3.3.1. bereits geschildert.
Will man nun einen Fachwert in eine Datenbank schreiben, muß an der Klasse `RDBMSWriter` die Methode `writeValue` gerufen werden. Diese erhält als Argumente den Fachwert unter seinem `ValueAtomizable-`Interface und den Bezeichner des Fachwertes. Der Bezeichner wird als Präfix für die Bezeichner der Komponenten des Fachwertes verwendet, um beim Auslesen feststellen zu können, welche Einträge zu dem Fachwert gehören. Nachdem am Fachwertobjekt die Methode `writoTo` gerufen und ausgeführt wurde, wird das Präfix wieder gelöscht. Die Methode `writeValue` sieht wie folgt aus:

```
public void writeValue (String name, ValueAtomizable value)
{
    Prüfung der Vorbedingungen

      Contract.require(name != null && name.length() != 0, this);
      Contract.require(value != null, this);

    Um beim Auslesen ermitteln zu können, welche Elemente zu dem Fachwert
    gehören, wird neben der Bezeichner des Fachwertes als Präfix für weitere
    Schreibvorgänge verwendet.

      _namePrefix = _nameprefix + '@' + name;

    Aufruf der Methode zum Schreiben des Fachwertes.

      value.writeTo(this);

    Löschen des Präfixes

      _namePrefix = _namePrefix.substring(0,
                                _namePrefix.lastIndexOf('@'));[1]
}
```

An der Fachwertklasse wird nun die Methode `writoTo` mit dem `RDBMSWriter` als Parameter gerufen. In dieser Methode wird eine Repräsentation des Fachwertes beispielsweise als String oder Integer erzeugt. Anschließend wird von dieser Methode

[1] Wird die Methode `substring` mit den Parametern `start` und `end` (beide sind vom Typ `int`) auf einen String angewandt, ist das Ergebnis ein Teilstring, welcher die Zeichen von `start` bis `end-1` enthält.

am RDBMSWriter die Methode writeString mit der Repräsentation und dem Bezeichner dieser Repräsentation als Parametern aufgerufen. Im Writer wird ein neues Feld in der Datenbank angelegt und mit dem Namen des Fachwertes versehen. Anschließend wird die Repräsentation des Fachwertes vom RDBMSWriter in ein geeignetes Feld geschrieben. Die Methode writeTo am Fachwert Zinssatz müßte nun wie folgt aussehen:

```
public void writeTo(Writer w)
{
      Contract.require(w != null, this);

      w.writeDouble("value", _value);

}
```

Die Abbildung zeigt die Abfolge der Methodenaufrufe beim Schreiben von Fachwerten.

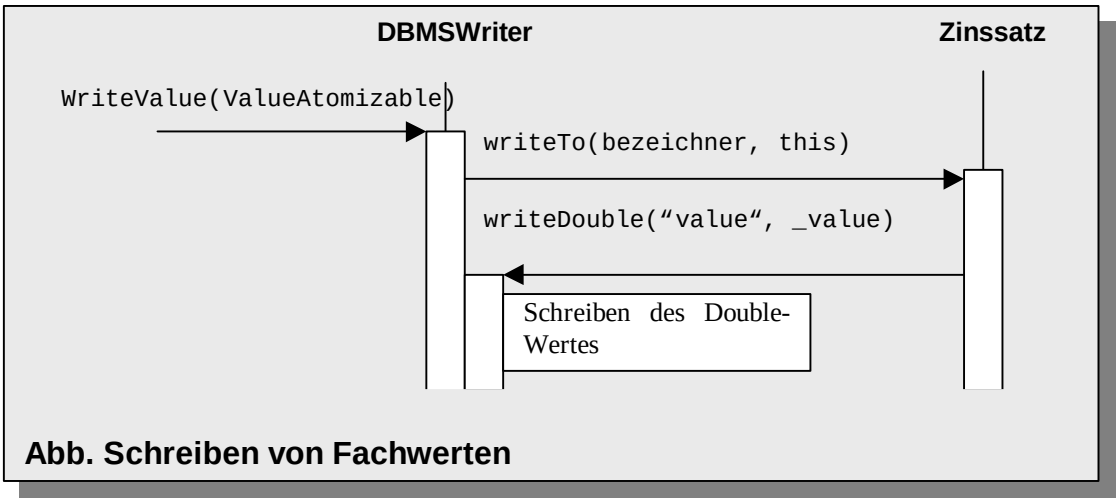


Abb. Schreiben von Fachwerten

Soll nun ein Fachwert mit der Methode readValue aus der Datenbank ausgelesen werden, muß beim Aufruf am DBMSReader-Objekt und der Bezeichner des betreffenden Fachwertes und dessen Klasse übergeben werden. Im Reader muß nun versucht werden die Methode readFrom am der Klasse des Fachwertes aufzurufen, um den Fachwert zu lesen. Da es sich bei der Methode readFrom um eine static-Methode handelt kann diese nicht in einem Interface deklariert werden. Da diese Methode deshalb nicht in dem Interface ValueAtomizable enthalten ist, muß sie erst über ein Method-Objekt instanziiert werden. Anschließend erfolgt über das Method-Objekt der Aufruf der readFrom-Methode an der Klasse Zinssatz. Ist dieser Aufruf erfolgreich, werden in dieser Methode die Attribute des Fachwertes gelesen und das Fachwertobjekt mit new-Value erzeugt. Anschließend wird der Fachwert von der Fachwertklasse unter seiner ValueAtomizable-Interface an den Reader übergeben.
Der Programmcode der Methode readValue ist nachfolgend aufgeführt:

```
public ValueAtomizable readValue (Class valueClass, String name)
{
    Prüfung der Vorbedingungen

      Contract.require(valueClass != null, this);
```

```
        Contract.require(name != null && name.length() != 0, this);
```

Initialisierung des Ergebnisses

```
        ValueAtomizable result = null;
```

Der Bezeichner des Wertes wird als Präfix für den Wert verwendet.

```
        _namePrefix = _namePrefix + '@' + name;
```

Beginn des Lesevorganges von der aktuellen Tabelle

```
        try {
```

Aufruf der `readFrom`-Methode an der Klasse Zinssatz.

```
              Class[] typeArgs = {Reader.class};
              Method m = valueClass.getMethod("readFrom",typeArgs);
              Object[] objArgs = {this};
              result = (ValueAtomizable) m.invoke(null, objArgs);
        } catch (Exception e) {
              handleClassCreationProblem(e, valueClass.getName());
        }
```

Löschen des Präfixes

```
        _namePrefix = _namePrefix.substring(0,
                                  _namePrefix.lastIndexOf('@'));
```

Prüfung der Nachbedingungen

```
        Contract.ensure(result != null, this);
        Contract.ensure(result.getClass() == valueClass, this);
```

Rückgabe des Ergebnisses

```
        return result;
    }
```

Der Programmcode der Methode `readFrom` ist nachfolgend aufgeführt:

```
    public static ValueAtomizable readFrom (Reader r)
    {
```

Lesen der Parameter

```
        double value = r.readDouble("value");
```

Erzeugung des Fachwertes

```
        return newValue(value);
    }
```

Die Abbildung zeigt die schematische Abfolge der Methodenaufrufe beim Lesen von Fachwerten.

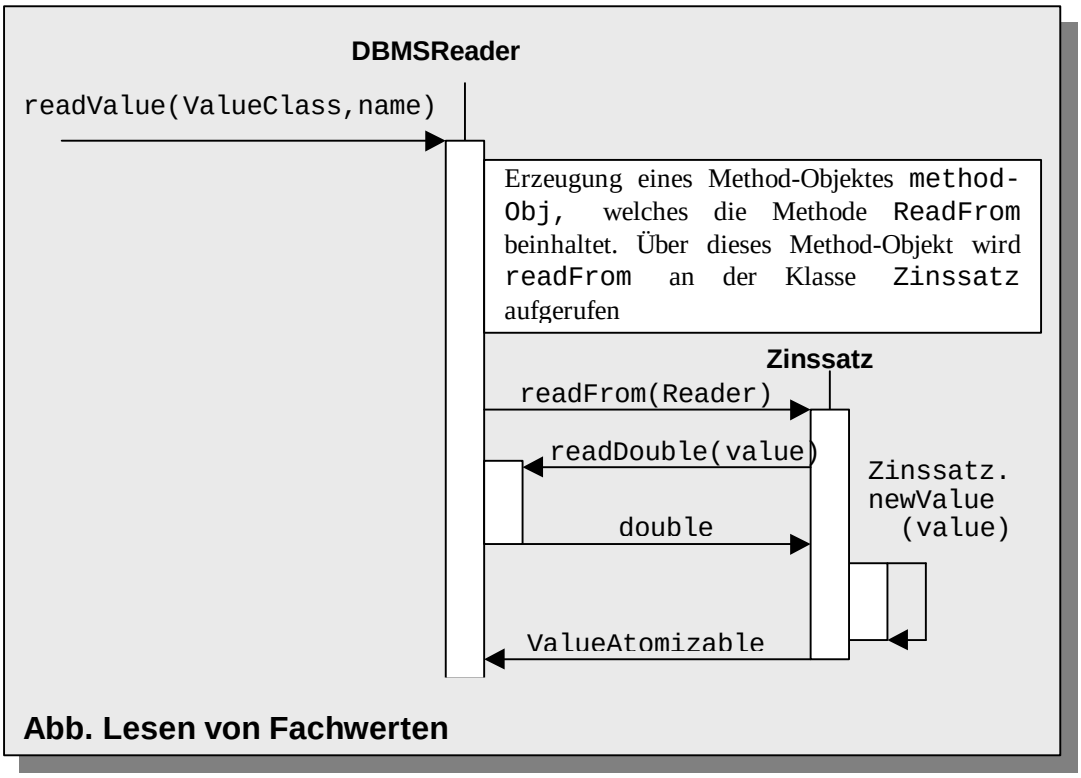


Abb. Lesen von Fachwerten

Aus den oben genannten Anpassungen ergibt sich folgende erweiterte Klassenhierarchie für Fachwerte im JWAM-Rahmenwerk:

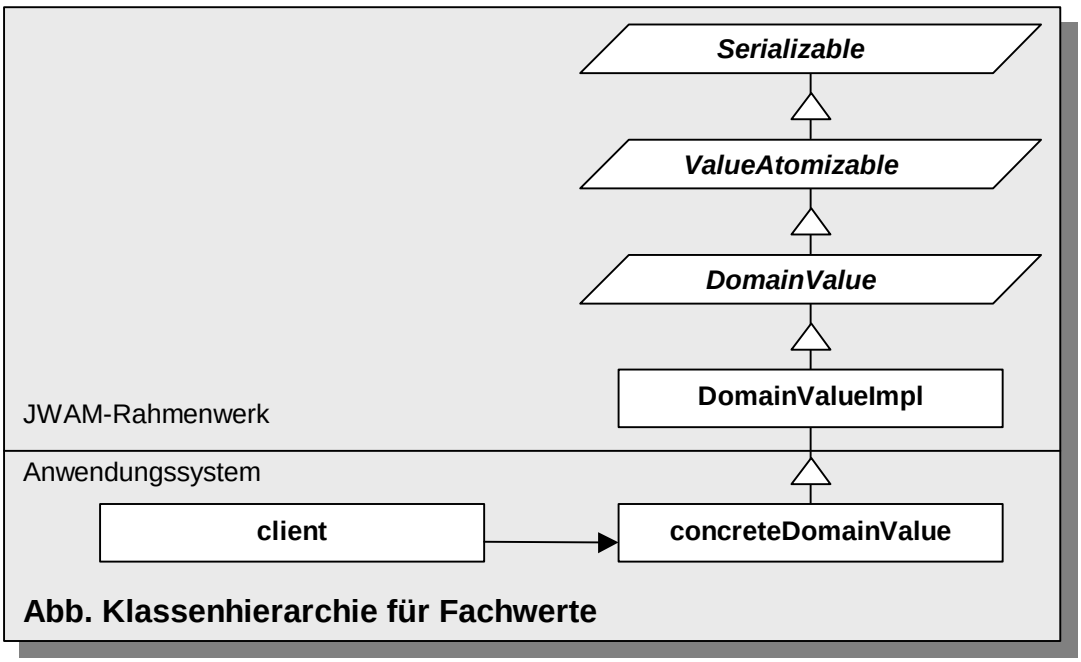


Abb. Klassenhierarchie für Fachwerte

4. Zusammenfassung

In der vorliegenden Studienarbeit wurde anhand der grundlegenden Quellen zum Thema der Fachwerte ein Fachwertkonzept für das JWAM-Rahmenwerk entwickelt und an ausgewählten aus verschiedenen Anwendungsbereichen stammenden Fachwerten auf seine Verwendbarkeit getestet. Mit den Fachwerten steht nun ein Ausdrucksmittel zur Verfügung, mit dem es möglich ist, fachliche Werte in einer objektorientierten Sprache unter Beibehaltung fachlicher Umgangsformen und Wertsemantik zu modellieren. Diese Anforderungen wurden in Kapitel 2 definiert und konnten, soweit dies unter Berücksichtigung der Einschränkungen durch die ausgewählte Programmiersprache Java möglich war, durch die gewählten Entwurfsmuster und deren Umsetzung realisiert werden. Die elementare Forderung nach Wertsemantik konnte durch den Ansatz der unveränderlichen Objekte erfüllt werden. Das normalerweise bei dieser Form der Modellierung auftretende Speicherplatzproblem zur Laufzeit, konnte durch die Verwendung des „flyweight/factory“-Entwurfsmusters, welches sich auch gut für eine Umsetzung unter Java eignet, auf ein Minimum begrenzt werden. Die Aufwand für die Implementation der Fachwertklassen, wurde dadurch reduziert, daß die Fabrik („factory“) und das Fliegengewicht („flyweight“) zusammen in einer Klasse realisiert wurden. Daneben wurden verschiedene Ansätze zur Persistenz von Fachwerten behandelt. Es wurde gezeigt, wie die Standardserialisierung unter Java anzupassen ist, damit das Konzept des „flyweight/factory“-Entwurfsmusters aufrecht erhalten werden kann. Wieterhin mit dem Atomizer aus dem JWAM-Rahmenwerk ein anderer Ansatz zur Persistenz von Fachwerten gezeigt. Bei dem vorgestellten Ansatz muß der Entwickler von konkreten Fachwertklassen zwar einen Teil der anfallenden Implementation selbst übernehmen, aber es werden bei den Schnittstellen der `Reader`- und `Writer`-Klassen keine zusätzlichen Methoden benötigt. Daneben gibt es noch weitere Mechanismen, die aber nicht alle in dieser Arbeit vorgestellt werden konnten.

Natürlich hat das Fachwertkonzept, wie es in dieser Studienarbeit umgesetzt wurde, einige Schwächen. Diese resultieren, wie in den Kapiteln 2 und 3 geschildert, zumeist aus den Beschränkungen der objektorientierten Programmiersprachen und im Speziellen aus den in Java zur Verfügung stehenden Sprachelementen. Ein Beispiel hierfür ist die aufwendige Erzeugung und Verwaltung von Fachwertobjekten. Daneben gibt es noch das generelle Problem, Werte und ein wertähnliches Verhalten durch Objekte zu modellieren. Daneben galt es die Anforderungen des JWAM-Rahmenwerks zu erfüllen. Dabei waren zum einen die Programmierkonvertionen des rahmenwerks einzuhalten und zum anderen sollten die Fachwerte einen möglichst vielen verschiedenen Anforderungen gerecht werden, um flexibel einsetzbar zu sein.

Allerdings sollte darüber nachgedacht werden, das Fachwertkonzept gegebenenfalls durch weitere Konzepte zum Beispiel aus der funktionalen Programmierung zu ergänzen. Ein Beispiel hierfür wäre „Lazy Evaluation“. Mit dieser hätte man die Möglichkeit mit der Auswertung von auf Werten beruhenden Ausdrücken bis zu dem Zeitpunkt zu warten, an dem sie benötigt wird. Auch mußte geprüft werden, ob das Fachwertkonzept nicht durch die Einführung von zusätzlichen Basisklassen, welche von der jetzigen Basisklasse `DomainValue` erben, einzuführen, um die im Rahmenwerk zur Verfügung stehenden Möglichkeiten noch zu erweitern. Beispielsweise könnte sinnvoll sein, eine

zusätzliche Basisklasse für Fachwerte anzubieten, auf deren Wertebereichen arithmetische Operationen definiert sind.
Des weiteren wäre es interessant zu untersuchen, wie sich Fachwerte, mit einem zur Laufzeit konfigurierbaren Wertebereich verhalten. Das heißt, wie läßt sich folgende Situation bewältigen: Im laufenden Betrieb soll der Wertebereich eines Fachwertes um ein Element erweitert, ohne eine Programmänderung notwendig wäre.
Diese Punkte müssen jedoch im Rahmen der Weiterentwicklung des JWAM-Rahmenwerkes gesondert untersucht werden.

Anhang

A. Schnittstelle des Basisinterfaces `DomainValue`:

```
public interface DomainValue
{
  // Methoden
      public boolean isDefined();
      public String  toString();
      public boolean equals(Object value);
      public int     hashCode();
}
```

B. Schnittstelle der Basisklasse `DomainValueImpl`:

```
public abstract class DomainValue
      implements DomainValue
{
  // Klassenmethoden
      protected static DomainValue registeredValue (DomainValue
            value);

  // Methoden
      public boolean isDefined();
      public String  toString();
      public boolean equals(Object value);
      public int     hashCode();

  // Klassenvariablen
      private static transient WeakHashMap _domainValueHashMap;
}
```

C. Schnittstelle der Subklasse `Zinssatz`:

```
public class Zinssatz
      extends DomainValueImpl
{
  // Klassenmethoden

      // Methoden zum Wertebereich
      public static Zinssatz getMaxValue();
      public static Zinssatz getMinValue();

      // Klassenkonstruktoren
      public static boolean  isValid(double value);
      public static boolean  isValid(String value);
      public static Zinssatz newValue(double value);
      public static Zinssatz newValue(String value);
      public static Zinssatz newValue();

  // Konstruktoren
      protected Zinssatz(double value);
      protected Zinssatz();
```

```
  // Methoden

      public boolean  isDefined();
      public double   toDouble();
      public String   toString();
      public boolean  isResultOfAddDefined(Zinssatz dvZinssatz);
      public Zinssatz add(Zinssatz dvZinssatz);
      public boolean  isSmaller(Zinssatz value);
      public boolean  isGreater(Zinssatz value);

 // Methoden zur Persistenz

      public static ValueAtomizable readFrom(Reader r);
      public static Zinssatz readDomainValue(ObjectInput ss)
                  throws IOException, ClassNotFoundException;
      protected void writeTo(Writer w);
      private   void writeObject(ObjectOutputStream out)
                                          throws IOException;
      private   void readObject(ObjectInputStream in)
                                          throws IOException;

// Variablen

      private double  _value;
      private boolean _defined;

// Konstanten
      private static final double _maxValue;
      private static final double _minValue;
}
```

D. Schnittstelle der Subklasse `Beschaeftigungsart`:

```
public class Beschaeftigungsart
      extends DomainValueImpl
{
// Klassenmethoden

      // Methoden zum Wertebereich
      public static Beschaeftigungsart[] listValues()

      // Klassenkonstruktoren
      public static boolean isValid(String value);
      public static Beschaeftigungsart newValue(String value);
      public static Beschaeftigungsart newValue();

// Konstruktoren

      protected Beschaeftigungsart (String value);
      protected Beschaeftigungsart ();

// Methoden

      public boolean isDefined();
      public String  toString();

// Methoden zur Persistenz

      public static ValueAtomizable readFrom(Reader r);
```

```
      public static Beschaeftigungsart readDomainValue
                  (ObjectInput ss) throws IOException,
                        ClassNotFoundException;
      protected void writeTo(Writer w);
      private   void writeObject(ObjectOutputStream out)
                                          throws IOException;
      private   void readObject(ObjectInputStream in)
            throws IOExceptionthrows ,ClassNotFoundException;

// Instanzvariablen
      private String  _value;
      private boolean _defined

// Konstanten
      private static final String _Beamter ;
      private static final String _Angestellter;
}
```

E. Schnittstelle des Interfaces `ValueAtomizable`:

```
public interface ValueAtomizable
      extends Serializable
{
  // Methoden
      public boolean writeTo(Writer w);
}
```

Literaturverzeichnis

[Bäumer et al. 98] D. Bäumer, D. Riehle, W. Siberski, C. Lilienthal, D. Megert, K.-H. Sylla, H. Züllighoven: *Values in Object Systems*, Ubilab Technical Report 98.10.1

[Bird & Wadler 92] R. Bird, P. Wadler: *Einführung in die funktionale Programmierung*, München: Carl Hauser und Prentice Hall, 1992.

[Coplien 92] J. Coplien: *Advanced C++: Programming Styles and Idioms*, Reading, Massachusetts: Addison-Wesley, 1992.

[Cunningham 95] W. Cunningham: *The Checks Pattern Language of Information Integrity*, In: J.O. Coplien and D.C. Schmidt: *Pattern Languages of Program Design*, Reading, Massachusetts: Addison-Wesley, 1995.

[Flanagan 97] D. Flanagan: *Java in a Nutshell, 2nd Edition* , O'Reilley Verlag, 1997.

[Gamma et al. 96] E. Gamma, R. Helm, R. Johnson, J. Vlissides: *Design Patterns - Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1996.

[Hoare 72] C. A. R. Hoare: *Notes on Data Structuring*, In: O.-J. Dahl, E. W. Dijkstra, C. A. R. Hoare: *Structured Programmming*, London: Academie Press, 1972.

[Holyer 91] I. Holyer: *Functional Programming with Miranda*, Pitman Publishing , 1991.

[Hinze 92] R. Hinze: E*inführung in funktionale Programmierung mit Miranda*, Teubner, 1992.

[Koenig&Moo 97] A. Koenig, B. Moo: *Ruminations on C++*, Reading, Massachusetts: Addison-Wesley, 1997.

[Naur 74] P. Naur: *Numbers and Arithmetic*, in: Concise Survey of Computer Methods, studentlitteratur Lund, Kopenhagen 1974.

[MacLennan 82] B. J. MacLennan: *Values and Objects in Programming Languages*, ACM SIGPLAN Notices, Vol.17, Nr. 12/1982.

[Meyer 88] B. Meyer: *Object-oriented Software Constructions*, New York: Prentice Hall, 1990.

[Meyer 90] B. Meyer: *Introduction to the Theory of Programming Languages*, Prentice Hall, 1990.

[Pawlan 98] M. Pawlan: *Reference Objects and Garbage Collection*, Technical Articles Index, Sun Microsystems Inc., 1998.

[Sebesta 93] R. W. Sebesta: *Concepts of Programming Languages*, Benjamin Cummings, 2. Auflage, 1993.

[van der Werf 98] P. van der Werf: *Values and Objects Revisited*, 1998

[Züllighoven 98] D. Bäumer, W.-G. Bleek, U. Bürkle, I. Dörre, M. Franz, G. Gryczan, J. Hess, R. Knoll, A. Krabbel, C. Lilienthal, D. Megert, D. Riehle, P. Rieth, S. Roock, W. Siebirsky, T. Slotos, W. Strunk, D. Weske, I. Wetzel, F. Wiegand, H. Wolf, M. Wulf, H. Zülighoven, : *Das objektorientierte Konstruktionshandbuch*, dpunkt.verlag, 1998.