

Entwurfsmuster von Kopf bis Fuß

Wäre es nicht wundervoll,
wenn es ein Buch über Entwurfsmuster gäbe, das mehr Spaß macht
als ein Besuch beim Zahnarzt
und aufschlussreicher ist als ein
Steuerformular? Ist wohl nur
ein Traum ...



Eric Freeman
Elisabeth Robson

mit
Kathy Sierra
Bert Bates

Deutsche Übersetzung von
Jørgen W. Lang

O'REILLY®

Der Inhalt (im Überblick)

	Einführung	xxiii
1	Willkommen bei den Entwurfsmustern: <i>Einführung in Entwurfsmuster</i>	1
2	Ihre Objekte auf dem Laufenden halten: <i>Das Observer-Muster</i>	37
3	Objekte dekorieren: <i>Das Decorator-Muster</i>	79
4	In der OO-Bäckerei ...: <i>Das Factory-Muster</i>	109
5	Einmalige Objekte: <i>Das Singleton-Muster</i>	169
6	Aufrufe verkapseln: <i>Das Command-Muster</i>	191
7	Anpassungsfähigkeit beweisen: <i>Die Adapter- und Facade-Muster</i>	237
8	Algorithmen verkapseln: <i>Das Template Method-Muster</i>	277
9	Erfolgreiche Collections: <i>Die Iterator- und Composite-Muster</i>	317
10	Der (Zu-)Stand der Dinge: <i>Das State-Muster</i>	381
11	Objektzugriff kontrollieren: <i>Das Proxy-Muster</i>	425
12	Muster von Mustern: <i>Zusammengesetzte Muster</i>	493
13	Muster in der wahren Welt: <i>Schöner leben mit Mustern</i>	563
14	Anhang: <i>Übrig gebliebene Muster</i>	597

Inhalt (jetzt ausführlich)

Einführung

Ihr mustergültiges Gehirn. Sie versuchen, etwas zu lernen, und Ihr Hirn tut sein Bestes, damit das Gelernte nicht hängen bleibt. Es denkt nämlich: »Wir sollten lieber ordentlich Platz für wichtigere Dinge lassen, z. B. für das Wissen, welche Tiere einem gefährlich werden könnten, oder dass es eine ganz schlechte Idee ist, nackt Snowboard zu fahren.« Tja, wie schaffen wir es nun, Ihr Gehirn davon zu überzeugen, dass Ihr Leben davon abhängt, etwas über Entwurfsmuster zu wissen?

Für wen ist dieses Buch?	xxiv
Wir wissen, was Sie gerade denken	xxv
Und wir wissen, was Ihr Gehirn gerade denkt	xxv
Metakognition: Nachdenken übers Denken	xxvii
Machen Sie sich Ihr Hirn untertan	xxix
Fachgutachter	xxxii
Danksagungen	xxxiv

1

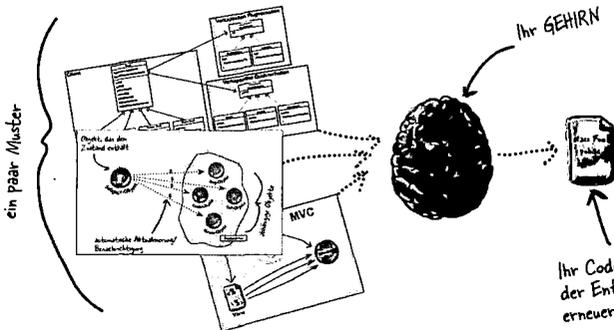
Willkommen bei den Entwurfsmustern

Irgendjemand hat Ihr Problem schon gelöst. In diesem Kapitel lernen Sie, warum (und wie) Sie die Weisheit und die Lehren anderer Entwickler nutzen können, die die gleichen Designprobleme bereits hatten und die Reise überlebt haben. Bevor dieses Kapitel zu Ende ist, kümmern wir uns um die Verwendung und die Vorteile der Entwurfsmuster, sehen uns ein paar grundsätzliche objektorientierte (OO-)Designprinzipien an und gehen mit Ihnen zusammen ein Beispiel für die Funktionsweise von Entwurfsmustern durch. Die beste Möglichkeit, die Muster zu verwenden, ist, sie *in Ihr Gehirn zu laden* und dann die Stellen in Ihren Designs und bestehenden Programmen zu *erkennen*, an denen der Einsatz sinnvoll ist. Im Gegensatz zur Codewiederverwendung können Sie mit Entwurfsmustern die *Erfahrung* anderer Menschen wiederverwenden.

Vergiss nicht: Das Wissen um Konzepte wie Abstraktion, Vererbung und Polymorphismus macht dich noch nicht zu einer guten OO-Designerin. Eine Design-Meisterin überlegt, wie sie flexible Entwürfe erschaffen kann, die wartbar sind und mit Veränderungen umgehen können.



Es begann mit einer einfachen SimUDuck-App	2
Aber jetzt sollen die Enten FLIEGEN können	3
Aber irgendetwas ging furchtbar schief ...	4
Joe denkt über Vererbung nach ...	5
Wie wäre es mit einem Interface?	6
Was würden Sie an Joes Stelle tun?	7
Die einzige Konstante in der Softwareentwicklung	8
Das Problem eingrenzen	9
Veränderliches und Unveränderliches voneinander trennen	10
Das Entenverhalten entwerfen	11
Das Entenverhalten implementieren	13
Das Entenverhalten integrieren	15
Den Entencode testen	18
Verhalten dynamisch festlegen	20
Das große Ganze: Verkapseltes Verhalten	22
HAT-EIN ist besser als IST-EIN	23
Da wir gerade von Entwurfsmustern sprechen ...	24
Im Bistro um die Ecke aufgeschnappt ...	26
Im Büro nebenan aufgeschnappt ...	27
Die Stärke eines gemeinsamen Mustervokabulars	28
Wie setze ich Entwurfsmuster ein?	29
Werkzeuge für Ihren Entwurfs-Werkzeugkasten	32



Das Observer-Muster

Ihre Objekte auf dem Laufenden halten

2

Sie wollen interessante Ereignisse doch nicht verpassen, oder? Es gibt ein Muster, das unsere Objekte auf dem Laufenden hält, wenn etwas für sie Wichtiges passiert, und zwar eins der am häufigsten verwendeten und nützlichsten Entwurfsmuster überhaupt: das Observer-Muster. In diesem Kapitel sehen wir uns alle möglichen interessanten Eigenschaften dieses Musters an, wie *Eins-zu-viele-Beziehungen* und *lose Kopplungen*. Mit dem Observer-Muster im Gepäck sind Sie der Star jeder Muster-Party!

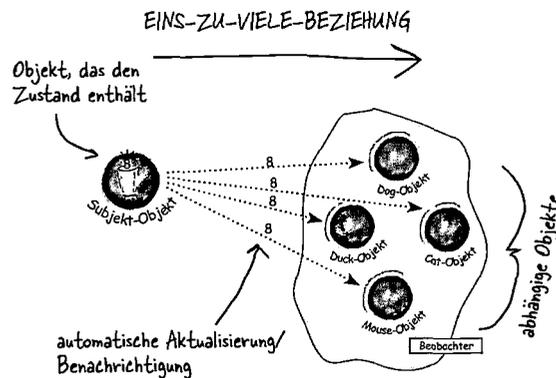
OO-Prinzipien

Verkapseln, was variabel ist.
Komposition vor Vererbung.
Auf Schnittstellen programmieren, nicht auf Implementierungen.
Streben Sie nach lose gekoppeltem Design zwischen Objekten, die interagieren.

OO-Grundlagen

Abstraktion
Klassifizierung
Vererbung

Die Wetterstation im Überblick	39
Willkommen zum Observer-Muster	44
Herausgeber + Abonnenten = Observer-Muster	45
Die Observer-Muster-Definition	51
Die Macht der losen Kopplung	54
Die Wetterstation entwerfen	57
Die Wetterstation implementieren	58
Die Wetterstation hochfahren	61
Das Observer-Muster in freier Wildbahn	65
Die lebensverändernde Applikation programmieren	66
Inzwischen bei Weather-O-Rama	69
Probefahrt für den neuen Code	71
Werkzeuge für Ihren Entwurfs-Werkzeugkasten	72
Die Entwurfsprinzipien-Challenge	73



Das Decorator-Muster

3 Objekte dekorieren

Nennen wir dieses Kapitel einfach »Vererbst du noch, oder designst du schon?«. Hier werfen wir einen weiteren Blick auf das typische Überstrapazieren von Vererbung. Sie werden lernen, wie Sie Ihre Klassen zur Laufzeit mit einer Form der Objektkomposition dekorieren können. Warum? Sobald Sie die Dekoration beherrschen, können Sie Ihren Objekten (oder denen anderer Leute) neue Verantwortung geben, *ohne hierfür den Code der zugrunde liegenden Klassen ändern zu müssen.*

Ich dachte immer, echte Männer benutzen grundsätzlich Subklassen. Bis ich die Macht der Erweiterung zur Laufzeit anstatt während der Kompilierung kennenlernte. Und jetzt sehen Sie mich an!

o
o



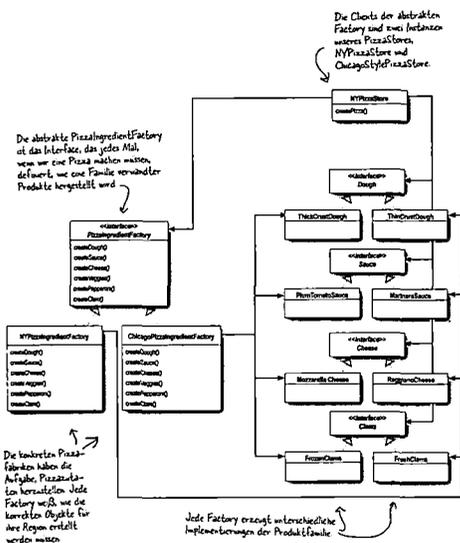
Willkommen bei Starbuzz Coffee	80
Das Offen/Geschlossen-Prinzip	86
Wir stellen vor: das Decorator-Muster	88
Eine Getränkebestellung mit Decoratoren aufbauen	89
Die Definition des Decorator-Musters	91
Unsere Getränke dekorieren	92
Den Starbuzz-Code schreiben	95
Getränke programmieren	96
Zutaten programmieren	97
Den Kaffee servieren	98
Decoratoren in freier Wildbahn: Java I/O	100
Die java.io-Klassen dekorieren	101
Einen eigenen Java-I/O-Decorator schreiben	102
Unseren neuen Java-I/O-Decorator testen	103
Werkzeuge für Ihren Entwurfs-Werkzeugkasten	105

Das Factory-Muster

4 In der OO-Bäckerei ...

Machen Sie sich bereit, ein paar lose gekoppelte OO-Entwürfe zu backen. Zur Erstellung von Objekten gehört mehr, als einfach den `new`-Operator einzusetzen. Sie werden lernen, dass Instanziierung nicht in der Öffentlichkeit durchgeführt werden sollte und oft zu *Kopplungsproblemen* führen kann. Und das wollen wir nun wirklich nicht, oder? Finden Sie heraus, wie das Factory-Muster Sie vor peinlichen Abhängigkeiten bewahren können.

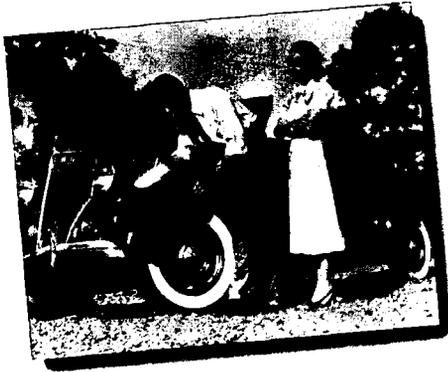
Das Veränderliche finden	112
Die Objekterstellung verkapseln	114
Eine einfache Pizzafabrik erstellen	115
Die einfache Fabrik definieren	117
Ein Framework für die Pizzeria	120
Den Subklassen die Entscheidung überlassen	121
Eine Fabrikmethode deklarieren	125
Jetzt ist es endlich Zeit, das Factory Method-Muster kennenzulernen	131
Ein paralleler Blick auf Hersteller und Produkte	132
Die Definition des Factory Method-Musters	134
Ein Blick auf Objektabhängigkeiten	138
Das Prinzip der Umkehrung der Abhängigkeiten	139
Das Prinzip anwenden	140
Zutatenfamilien	145
Die Zutatenfabriken bauen	146
Die Pizzas überarbeiten ...	149
Unsere Pizzerien überarbeiten	152
Was haben wir getan?	153
Die Definition des Abstract Factory-Musters	156
Factory Method und Abstract Factory im Vergleich	160
Werkzeuge für Ihren Entwurfs-Werkzeugkasten	162



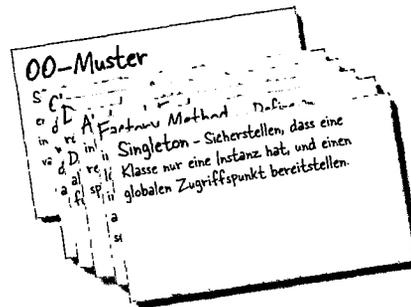
Das Singleton-Muster

5 Einmalige Objekte

Unser nächster Halt ist das Singleton-Muster, unsere Fahrkarte für die Erstellung einmaliger Objekte, von denen es immer nur eine Instanz gibt. Es wird Sie freuen, zu erfahren, dass das Singleton-Muster, bezogen auf sein Klassendiagramm, das einfachste Muster von allen ist. Tatsächlich enthält es nur eine einzige Klasse! Aber machen Sie es sich nicht zu bequem. Trotz des einfachen Klassendesigns müssen wir für seine Implementierung einige tiefgehende objektorientierte Überlegungen anstellen. Also, setzen Sie Ihre Denkmütze auf, und los geht's.



Die Implementierung des klassischen Singleton-Musters im Detail	173
Die Schokoladenfabrik	175
Definition des Singleton-Musters	177
Brüssel Houston, wir haben ein Problem ...	178
Mit Multithreading umgehen	180
Können wir das Multithreading verbessern?	181
Inzwischen in der Schokoladenfabrik ...	183
Werkzeuge für Ihren Entwurfs-Werkzeugkasten	186

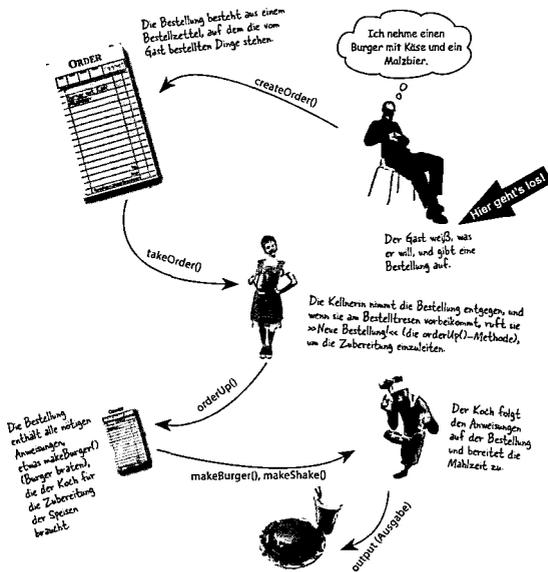


Das Command-Muster

6 Aufrufe verkapseln

In diesem Kapitel heben wir die Verkapselung auf ein ganz neues Niveau: Wir werden Methodenaufrufe verkapseln.

Ja, richtig gehört: Methodenaufrufe. Damit können wir Teile von Berechnungen »einfrieren«, wodurch sich das aufrufende Objekt nicht um die Details der Berechnung kümmern muss. Es nutzt einfach die eingefrorene Methode für die Erfüllung seiner Aufgabe. Mit diesen verkapselten Methodenaufrufen sind aber noch ganz andere schlaue Dinge möglich. Wir können sie beispielsweise zur Protokollierung nutzen oder sie wiederverwenden, um eine »Rückgängig«-Funktionalität zu implementieren.



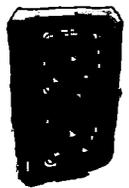
Ein Blick auf die Drittherstellerklassen	194
Inzwischen im Bistro ...	197
Vom Bistro zum Command-Muster	201
Unser erstes Command-Objekt	203
Das Command-Objekt verwenden	204
Befehle den Plätzen zuweisen	209
Die Fernsteuerung implementieren	210
Die Befehle implementieren	211
Die Fernsteuerung auf Herz und Nieren testen	212
Zeit für die Dokumentation ...	215
Was machen wir hier?	217
Zeit, den Rückgängig-Knopf auf seine Qualität zu testen!	220
Zustände für die Implementierung der »Rückgängig«-Funktion verwenden	221
Die Deckenventilator-Befehle mit einer »Rückgängig«-Funktion versehen	222
Jede Fernsteuerung braucht einen Partymodus!	225
Einen Makro-Befehl benutzen	226
Viele Verwendungsmöglichkeiten für das Command-Muster: Warteschlangen für Befehle	229
Weitere Anwendungen des Command-Musters: Aufträge protokollieren	230
Das Command-Muster in der wahren Welt	231
Werkzeuge für Ihren Entwurfs-Werkzeugkasten	233

Die Adapter- und Facade-Muster

7 Anpassungsfähigkeit beweisen

In diesem Kapitel versuchen wir so unmögliche Dinge wie die **Quadratur des Kreises**. Klingt ausgeschlossen? Aber nicht mit Entwurfsmustern. Erinnern Sie sich noch an das Decorator-Muster? Wir haben Objekte verpackt, um sie mit Verantwortlichkeiten zu versehen. Diesmal **verpacken wir Objekte**, damit ihre Schnittstellen wie etwas aussehen, das sie nicht sind. So können wir Designs, die bestimmte Schnittstellen erwarten, an Klassen anpassen, die eine andere Schnittstelle implementieren. Und das ist noch nicht alles. Wenn wir schon dabei sind, sehen wir uns gleich noch ein anderes Muster an, das Objekte verpackt, um ihre Schnittstelle zu vereinfachen.

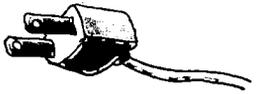
Steckdose in Großbritannien



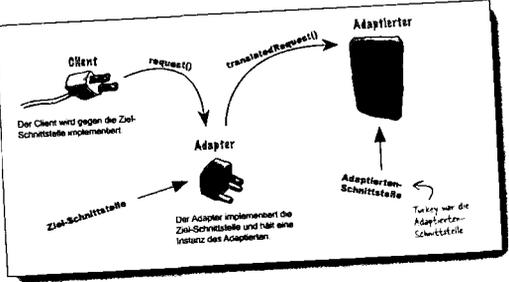
Stromadapter



US-Standardstecker



Überall Adapter	238
Objektorientierte Adapter	239
Probelauf für den Adapter	242
Das Adapter-Muster erklärt	243
Die Definition des Adapter-Musters	245
Objekt- und Klassen-Adapter	246
Adapter im echten Leben	250
Einen Enumerator an einen Iterator anpassen	251
Unser eigenes Heimkino	257
Einen Film ansehen (auf die harte Tour)	258
Licht, Kamera, Facade!	260
Die Heimkino-Facade konstruieren	263
Die vereinfachte Schnittstelle implementieren	264
Einen Film anschauen (auf die sanfte Tour)	265
Die Definition des Facade-Musters	266
Das Prinzip der Verschwiegenheit	267
Wie man KEINE Freunde gewinnt und KEINE Objekte beeinflusst	268
Das Facade-Muster und das Prinzip der Verschwiegenheit	271
Werkzeuge für Ihren Entwurfs-Werkzeugkasten	272

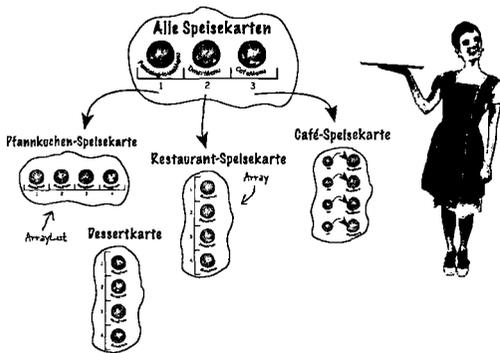


Die Iterator- und Composite-Muster

9

Erfolgreiche Collections

Es gibt viele Möglichkeiten, Objekte in einer Collection zu speichern. Zum Beispiel in einem Array, einem Stack, einer Liste oder einer HashMap – Sie haben die Wahl. Dabei hat jede Form ihre Vor- und Nachteile. Irgendwann werden Ihre Clients allerdings über diese Objekte iterieren wollen. Und wollen Sie ihnen dann Ihre Implementierung offenbaren? Hoffentlich nicht! Das wäre einfach nicht professionell. Aber keine Sorge. Ihre Karriere ist nicht gefährdet. In diesem Kapitel werden Sie sehen, wie Ihre Clients über Ihre Objekte iterieren können, ohne zu wissen, wie sie gespeichert sind. Außerdem lernen Sie, wie man Super Collections von Objekten erstellt, die mit einem einzigen Satz einige eindrucksvolle Datenstrukturen überspringen können. Und als wäre das noch nicht genug, werden Sie auch noch das eine oder andere über Objektverantwortlichkeit lernen.



Große Neuigkeiten: Das Restaurant und das Pfannkuchenhaus von Objectville fusionieren	318
Sehen wir uns die Gerichte an	319
Die Spezifikation implementieren: unser erster Versuch	323
Können wir die Iteration verkapseln?	325
Willkommen zum Iterator-Muster	327
DinerMenu mit einem Iterator versehen	328
Die Restaurant-Speisekarte mit einem Iterator überarbeiten	329
Den Kellnerin-Code aufmöbeln	330
Den Code testen	331
Unser aktueller Entwurf auf dem Prüfstand ...	333
Aufräumen mit java.util.Iterator	335
Die Definition des Iterator-Musters	338
Die Struktur des Iterator-Musters	339
Das Prinzip der einzelnen Verantwortlichkeit	340
Willkommen zu Javas Iterable-Interface	343
Javas erweiterte for-Schleife	344
Ein Blick auf die Speisekarte des Cafés	347
Iteratoren und Collections	353
Die Definition des Composite-Musters	360
Speisekarten mit dem Composite-Muster entwerfen	363
MenuComponent implementieren	364
Das Gericht (MenuItem) implementieren	365
Die Komposita-Speisekarte implementieren	366
Werkzeuge für Ihren Entwurfs-Werkzeugkasten	376

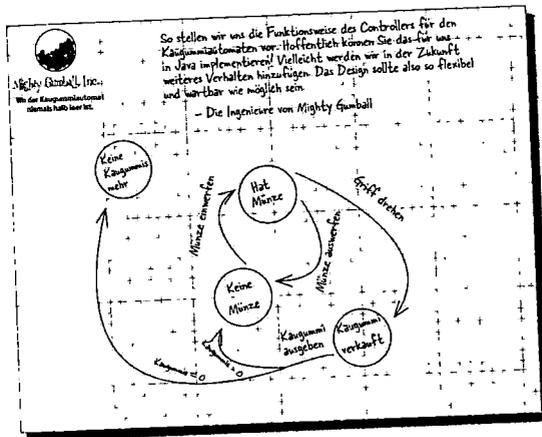
Das State-Muster

10

Der (Zu-)Stand der Dinge

Eine kaum bekannte Tatsache: Die Strategy- und das State-Muster sind Zwillinge, die bei der Geburt getrennt wurden.

Vielleicht denken Sie, dass beide ein ähnliches Leben führen. In Wirklichkeit hat Strategy jedoch ein unglaublich erfolgreiches Unternehmen rund um austauschbare Algorithmen aufgebaut, während State den vermutlich edleren Weg gewählt hat. Es hilft anderen Objekten, ihr Verhalten zu kontrollieren, indem es ihren inneren Zustand verändert. So unterschiedlich die Wege der beiden auch scheinen – hinter den Kulissen ist ihr Design fast identisch. Wie das sein kann und worum es beim State-Muster wirklich geht, werden wir herausfinden. Am Ende des Kapitels sehen wir uns dann an, welche Beziehung beide Muster tatsächlich zueinander haben.



Ein echter Java-Plombenzieher	382
Kurze Einführung in Zustandsautomaten	384
Den Code schreiben	386
Interner Testlauf	388
Sie haben es geahnt – ein Änderungswunsch!	390
ZUSTÄNDE wie bei Hempels unterm Sofa ...	392
Der neue Entwurf	394
Das Interface State und die Klassen definieren	395
Umbau des Kaugummiautomaten	398
Ein Blick auf die komplette Klasse GumballMachine ...	399
Weitere Zustände implementieren	400
Die Definition des State-Musters	406
Wir müssen uns wieder dem 1-von-10-Kaugummispiel widmen	409
Das Spiel fertigstellen	410
Demo für den CEO von Mighty Gumball, Inc.	411
Stimmt alles?	413
Das haben wir fast vergessen!	416
Werkzeuge für Ihren Entwurfs-Werkzeugkasten	419



11

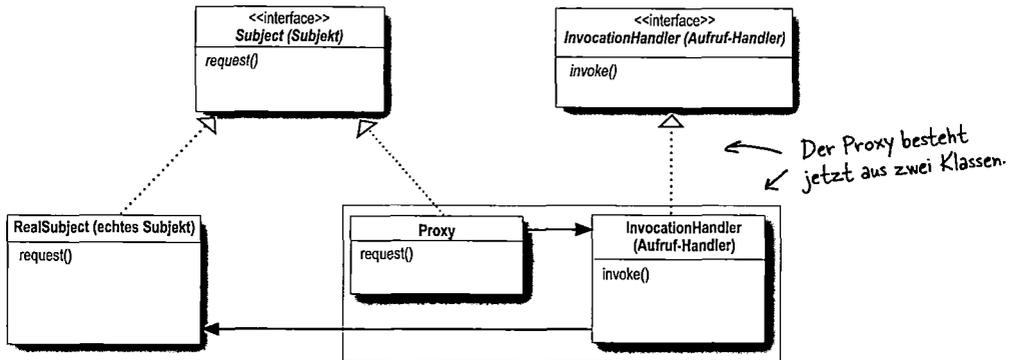
Objektzugriff kontrollieren

Haben Sie schon einmal »guter Bulle, böser Bulle« gespielt?

Sie sind der gute Polizist und stellen alle Ihre Dienste auf eine nette und freundliche Weise bereit. Wenn Sie aber nicht wollen, dass jeder ungefragt Ihre Dienste nutzt, übernimmt der böse Polizist die Zugangskontrolle für Sie. Denn genau das tun Proxies (»Stellvertreter«): Sie kontrollieren und verwalten den Zugriff. Wie Sie sehen werden, gibt es viele Möglichkeiten, Proxies als Vertreter für andere Objekte zu nutzen. Proxies sind dafür bekannt, dass sie für die von ihnen vertretenen Objekte komplette Methodenaufrufe über das Internet abwickeln. Außerdem nehmen sie bekanntermaßen den Platz einiger ziemlich fauler Objekte ein.



Den Überwachungscode schreiben	427
Den Überwachungscode testen	428
Einführung in entfernte Methodenaufrufe	433
Den Kaugummiautomaten (GumballMachine) als entfernten Dienst einrichten	446
Bei der RMI-Registry anmelden ...	448
Die Definition des Proxy-Musters	455
Bereitmachen für den virtuellen Proxy	457
Den virtuellen Proxy für die Albencover entwerfen	459
Den Bild-Proxy schreiben	460
Partnervermittlung für Geeks in Objectville	470
Die Person implementieren	471
Fünf-Minuten-Drama: Subjekte schützen	473
Das große Ganze: Einen dynamischen Proxy für Person erstellen	474
Der Proxy-Zoo	482
Werkzeuge für Ihren Entwurfs-Werkzeugkasten	485
Der Code für den Albumcover-Viewer	489

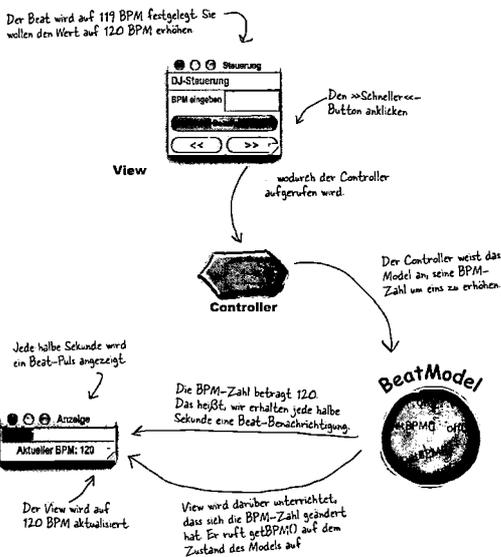


Zusammengesetzte Muster

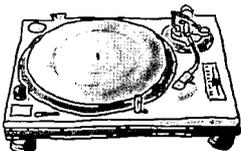
12

Muster von Mustern

Wer hätte gedacht, dass Muster auch zusammenarbeiten können? Sie haben bereits die erbitterten Kamingespräche miterlebt (und dabei haben wir Ihnen die Seiten mit dem »Pattern Death Match« noch gar nicht gezeigt, die wir auf Druck des Verlegers wieder entfernen mussten). Wer hätte gedacht, dass Muster eigentlich sogar recht gut miteinander auskommen können? Ob Sie's glauben oder nicht – einige der mächtigsten OO-Entwürfe verwenden Kombinationen mehrerer Muster. Machen Sie sich bereit für die nächste Stufe Ihrer Entwurfsmuster-Fähigkeiten: zusammengesetzte Muster (»Compound-Muster«)



Mustergültige Zusammenarbeit	494
Ein Wiedersehen mit den Enten	495
Was haben wir getan?	517
Ein Blick aus der Vogel-Entenperspektive: das Klassendiagramm	518
Der König der zusammengesetzten Muster	520
Willkommen zu Model-View-Controller	523
Genauer hingesehen ...	524
MVC als eine Reihe von Mustern verstehen	526
MVC, um den Beat zu steuern	528
Erstellung der Einzelteile	531
Ein Blick auf die konkrete Klasse BeatModel	532
Der View	533
Den View implementieren	534
Und damit zum Controller	536
Die Einzelteile zusammensetzen ...	538
Strategy erforschen	539
Das Model adaptieren	540
Noch ein Probelauf ...	542
Werkzeuge für Ihren Entwurfs-Werkzeugkasten	545

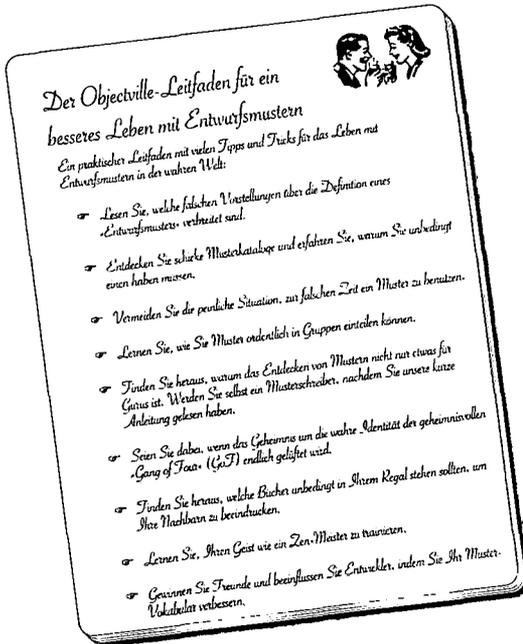


Schöner leben mit Mustern

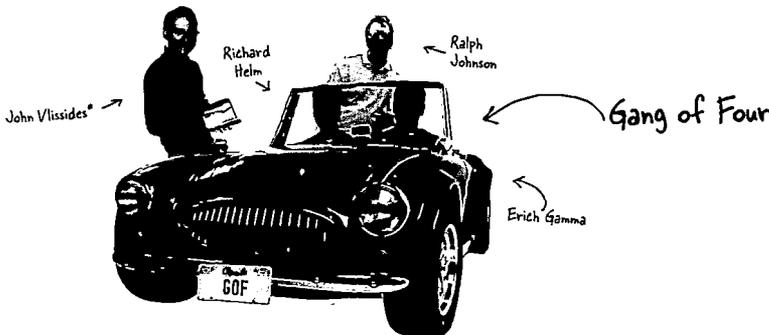
13

Muster in der wahren Welt

Und damit sind Sie bereit für eine strahlende neue Welt voller Entwurfsmuster. Aber bevor Sie all die neuen Chancen nutzen, müssen wir uns noch um ein paar Details kümmern, die Ihnen draußen in der wahren Welt begegnen können, wo die Dinge etwas komplexer sind als hier in Objectville. Auf den folgenden Seiten haben wir für Sie einen kleinen Reiseführer (oder Leitfaden) vorbereitet, der Sie beim Übergang begleiten wird ...



Definition von Entwurfsmustern	565
Ein genauerer Blick auf die Entwurfsmusterdefinition	567
Möge die Macht mit Ihnen sein	568
Sie wollen also eigene Entwurfsmuster schreiben	573
Entwurfsmuster ordnen	575
In Mustern denken	580
Denk-Muster	583
Vergessen Sie nicht die Macht des gemeinsamen Vokabulars	585
Spritztour durch Objectville mit der Gang of Four	587
Ihre Reise hat gerade erst begonnen!	588
Der Muster-Zoo	590
Mit Anti-Mustern das Böse auslöschen	592
Werkzeuge für Ihren Entwurfs-Werkzeugkasten	594
Abschied von Objectville ...	595



14 Anhang: Übrig gebliebene Muster

Nicht jeder kann der Beliebtste sein. Seit der Erstveröffentlichung von *Design Patterns: Elements of Reusable Object-Oriented Software* hat sich viel verändert. Entwickler haben diese Muster tausendfach verwendet. Die in diesem Anhang vorgestellten Muster sind Vollmitglieder der offiziellen GoF-Musterfamilie, werden aber nicht so oft genutzt wie die bisher gezeigten. Trotzdem sind sie auf ihre eigene Art großartig, und wenn die Situation es erfordert, können Sie sie einsetzen, ohne sich dafür zu schämen. In diesem Anhang wollen wir Ihnen einen Überblick darüber geben, worum es bei diesen Mustern geht.

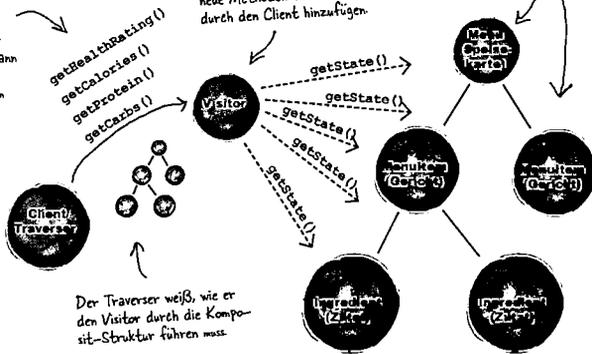
Der Client fordert beim Visitor Informationen aus der Komposit-Struktur an ... Der Visitor kann ohne Auswirkungen auf das Kompositum um neue Methoden erweitert werden.

getMealLehstang()
getCalories()
getProtein()
getCarbs()

Der Visitor muss in der Lage sein, getState() klassenübergreifend aufzurufen. Genau hier können Sie neue Methoden für die Verwendung durch den Client hinzufügen.

Alle diese Komposit-Klassen müssen nur eine getState()- (Zustand erfassen-)Methode hinzufügen (und keine Skrupel haben, sich selbst »blößzustellen«).

Der Traverser weiß, wie er den Visitor durch die Komposit-Struktur führen muss.



Bridge	598
Builder	600
Chain of Responsibility	602
Flyweight	604
Interpreter	606
Mediator	608
Memento	610
Prototype	612
Visitor	614