

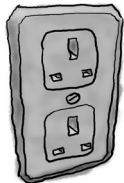
Die Adapter- und Facade-Muster

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.

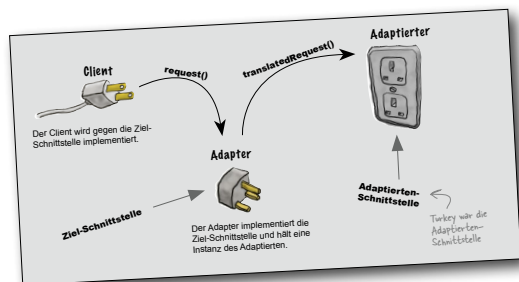
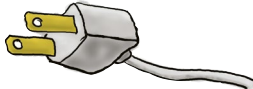
Steckdose in Großbritannien



Stromadapter



US-Standardstecker



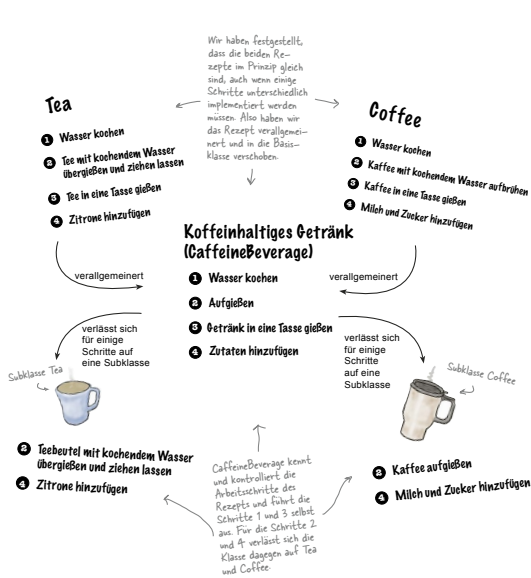
Überall Adapter	238
Objektorientierte Adapter	239
Probelauf für den Adapter	242
Das Adapter-Muster erklärt	243
Die Definition des Adapter-Musters	245
Objekt- und Klassen-Adapter	246
Adapter im echten Leben	250
Einen Enumerator an einen Iterator anpassen	251
Unser eigenes Heimkino	257
Einen Film ansehen (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

Das Template Method-Muster

8

Algorithmen verkapseln

Wir sind auf dem Verkapselungstrip. Was haben wir schon alles verkapselt? Objekterstellung, Methodenaufrufe, komplexe Schnittstellen, Enten, Pizzas – was kommt wohl als Nächstes? In diesem Kapitel gehen wir der Verkapselung von Algorithmenteilen auf den Grund, damit Subklassen sich bei Bedarf jederzeit direkt in eine Berechnung einklinken können. Außerdem lernen wir ein Entwurfsprinzip kennen, das von Hollywood inspiriert ist. Na dann mal los ...



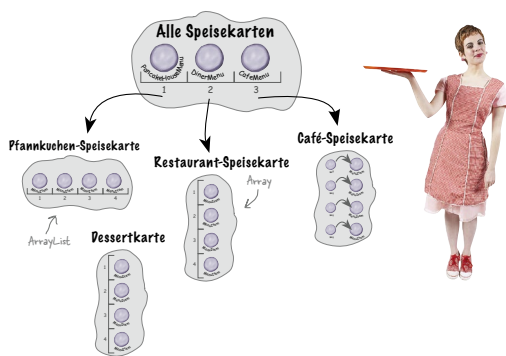
Zeit für mehr Koffein	278
Ein paar Kaffee- und Tee-Klassen zusammenrühren (in Java)	279
Kaffee und Tee abstrahieren	282
Das Design verbessern	283
prepareRecipe() abstrahieren	284
Was haben wir getan?	287
Willkommen beim Template Method-Muster	288
Was hat uns das Template Method-Muster gebracht?	290
Die Definition des Template Method-Musters	291
Eingehängt in die Template Method ...	294
Den Hook verwenden	295
Das Hollywood-Prinzip und das Template Method-Muster	299
Template-Methoden in freier Wildbahn	301
Mit dem Template Method-Muster sortieren	302
Was ist compareTo()?	303
Enten mit Enten vergleichen	304
Ein paar Enten sortieren	305
Das Making-of der Entensortiermaschine	306
Swinging mit Frames	308
Eigene Listen mit AbstractList	309
Werkzeuge für Ihren Entwurfs-Werkzeugkasten	313

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

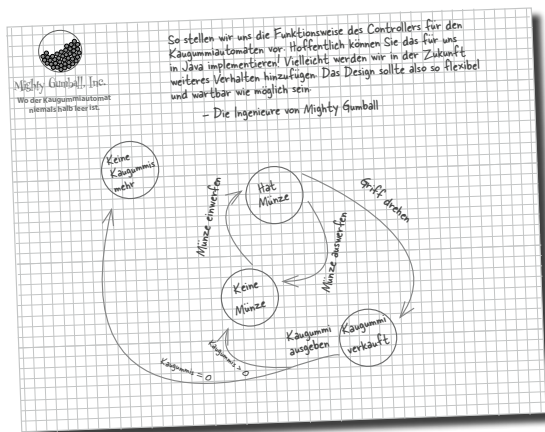
Das State-Muster

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

