

Inhalt

Mae govannen!	XIII
1 Singleton Pattern	1
1.1 Was Design Patterns sind	1
1.2 Die GoF und deren Verdienst	2
1.3 Die Musterkategorien der GoF	3
1.3.1 Erzeugungsmuster	3
1.3.2 Verhaltensmuster	3
1.3.3 Strukturmuster	3
1.3.4 Bewertung der Kategorisierung	4
1.4 Beschreibung der Muster	4
1.5 Was ist ein Pattern?	5
1.6 Objektorientierte Programmierung	6
1.7 Muster in der Praxis	9
1.8 Das Singleton Pattern	10
1.8.1 Aufgabe von Singleton und Beispiele	10
1.8.2 Realisierung des Patterns	11
1.8.3 Die Zugriffsmethode synchronisieren	12
1.8.4 Eine andere Lösung: Double-checked locking	13
1.8.5 Letzter Ansatz: early instantiation - frühes Laden	14
1.9 Antipattern	15
1.9.1 Kritik an Singleton	15
1.9.2 Ist Singleton ein Antipattern?	16
1.10 Zusammenfassung	16
2 Template Method Pattern	19
2.1 Die Arbeitsweise von Template Method	19
2.1.1 Der erste Ansatz	19
2.1.2 Der zweite Ansatz	20
2.1.3 Das Hollywood-Prinzip	21
2.1.4 Einführen von Hook-Methoden	21
2.2 Das Interface „ListModel“	23

2.3	Der patternCoder	24
2.3.1	Der patternCoder aus Anwendersicht	25
2.4	Zusammenfassung	34
3	Observer Pattern	35
3.1	Einleitung	35
3.2	Eine erste Realisierung	35
3.3	Den ersten Ansatz erweitern	37
3.4	Observer in der Klassenbibliothek	38
3.5	Nebenläufiger Zugriff	39
3.5.1	Zugriffe synchronisieren	40
3.5.2	Die Datenbasis kopieren	40
3.5.3	Einsatz einer thread-sicheren Liste	41
3.6	Observer als Listener	42
3.7	Listener in der GUI-Programmierung	43
3.8	Das Model-View-Controller Pattern	48
3.9	Zusammenfassung	49
4	Mediator Pattern	51
4.1	Abgrenzung zum Observer Pattern	51
4.2	Aufgabe des Mediator Patterns	51
4.3	Mediator in Aktion – ein Beispiel	52
4.3.1	Definition eines Consumers	52
4.3.2	Definition eines Producers	53
4.3.3	Interface des Mediators	54
4.3.4	Test des Mediator-Patterns	55
4.4	Mediator in Aktion – das zweite Beispiel	56
4.4.1	Mediator in der GUI-Programmierung	56
4.4.2	Aufbau der GUI	57
4.5	Kritik an Mediator	59
4.6	Zusammenfassung	59
5	Chain of Responsibility	61
5.1	Ein Beispiel aus der realen Welt	61
5.2	Beispiel 1: Lebensmitteleinkauf	61
5.2.1	Die benötigten Lebensmittel	62
5.2.2	Die Verkäufer der Lebensmittel	62
5.2.3	Der Client	64
5.2.3.1	Erweiterung des Projekts	64
5.2.3.2	Variationen des Patterns	65
5.3	Ein Beispiel aus der Klassenbibliothek	67
5.4	Zusammenfassung	68

6	State Pattern	69
6.1	Exkurs: das Enum Pattern	69
6.1.1	Einen Zustand durch Zahlenwerte darstellen	69
6.1.2	Einen Zustand durch Objekte darstellen	70
6.1.3	Umsetzung in der Java-Klassenbibliothek	71
6.2	Den Zustand eines Objekts ändern	72
6.2.1	Den Zustand ändern – erster Ansatz	72
6.2.2	Den Zustand ändern – zweiter Ansatz	73
6.3	Prinzip des State Patterns	75
6.3.1	Die Rolle aller Zustände definieren	75
6.3.2	Das Projekt aus der Sicht des Client	77
6.3.3	Veränderung des Projekts	78
6.3.3.1	State-Objekte zentral im Kontext verwalten	79
6.3.3.2	State-Objekte als Rückgabewerte von Methodenaufrufen	79
6.4	Das State Pattern in der Praxis	80
6.5	Zusammenfassung	81
7	Command Pattern	83
7.1	Befehle in Klassen kapseln	83
7.1.1	Version 1 – Grundversion	83
7.1.2	Weitere Anbieter treten auf	85
7.1.3	Einen Befehl kapseln	86
7.2	Command in der Klassenbibliothek	89
7.2.1	Beispiel 1: Nebenläufigkeit	89
7.2.2	Beispiel 2: Event-Handling	90
7.3	Befehlsobjekte wiederverwenden	91
7.3.1	Das Interface „Action“	91
7.3.2	Verwendung des Interface „Action“	91
7.4	Undo und redo von Befehlen	92
7.4.1	Ein einfaches Beispiel	93
7.4.2	Ein umfangreicheres Beispiel	95
7.4.3	Besprechung des Source Codes	96
7.4.3.1	Die beteiligten Klassen	96
7.4.3.2	Aufgabe der GUI	96
7.4.3.3	Arbeitsweise der Command-Klassen	97
7.4.3.4	Undo und redo	97
7.4.4	Undo und redo grundsätzlich betrachtet	97
7.5	Zusammenfassung	98
8	Strategy Pattern	99
8.1	Ein erster Ansatz	99
8.2	Strategy in Aktion – Sortieralgorithmen	101
8.2.1	Das gemeinsame Interface	101

8.2.2	Prinzip des Selection Sort	102
8.2.3	Prinzip des Merge Sort	103
8.2.4	Prinzip des Quick Sort	103
8.2.5	Der Kontext	104
8.2.6	Bewertung des Ansatzes und Variationen davon	105
8.3	Das Strategy Pattern in der Praxis	106
8.4	Abgrenzung zu anderen Mustern	107
8.5	Zusammenfassung	108
9	Iterator Pattern	109
9.1	Zwei Möglichkeiten, Daten zu speichern	109
9.1.1	Daten in einem Array speichern	109
9.1.2	Daten in einer Kette speichern	111
9.2	Aufgabe eines Iterators	113
9.3	Das Interface „Iterator“ in Java	114
9.3.1	Der Iterator der Klasse „MyArray“	114
9.3.1.1	Test des Iterators	115
9.3.1.2	Kritik am Iterator	115
9.3.2	Der Iterator der Klasse „MyList“	115
9.3.2.1	Test des Iterators	116
9.3.2.2	Nutzen des Iterators	117
9.4	Das Interface „Iterable“	117
9.5	Zusammenfassung	118
10	Objektorientierte Entwurfsprinzipien	121
10.1	Gegen Schnittstellen programmieren	121
10.2	Single Responsibility Principle (SRP)	123
10.3	Inheritance may be evil	124
10.4	Open/Closed Principle (OCP)	125
10.5	Prinzip des rechten Augenmaßes	126
11	Abstract Factory Pattern	127
11.1	Gärten anlegen	127
11.1.1	Der erste Ansatz	127
11.1.2	Der zweite Ansatz – Vererbung	129
11.1.3	Der dritte Ansatz – die abstrakte Fabrik	129
11.1.4	Vorteil der abstrakten Fabrik	131
11.1.5	Einen neuen Garten definieren	132
11.2	Diskussion des Patterns und Praxis	133
11.3	Gespenster jagen	134
11.3.1	Die erste Version	134
11.3.1.1	Der Spieler	134
11.3.1.2	Die vier Himmelsrichtungen	135

11.3.1.3 Die Bauteile des Hauses	136
11.3.1.4 Die Klasse „Haus“ steuert das Spiel	138
11.3.2 Die zweite Version des Projekts	140
11.3.3 Version 3 – Einführung einer weiteren Fabrik	142
11.3.3.1 Erweiterung in dieser Programmversion	142
11.3.3.2 Die neue Fabrik „TuerMitZauberspruchFabrik“	142
11.3.3.3 Das neue Bauteil „TuerMitZauberspruch“	143
11.3.4 Version 4 – das Geisterhaus	144
11.4 Zusammenfassung	145
12 Factory Method Pattern	147
12.1 Ein erstes Beispiel	147
12.2 Variationen des Beispiels	149
12.3 Praktische Anwendung des Patterns	150
12.3.1 Rückgriff auf das Iterator Pattern	150
12.3.2 Bei der Abstract Factory	151
12.4 Ein größeres Beispiel – ein Framework	152
12.4.1 Das Projekt, das erstellt werden wird	152
12.4.2 Die Schnittstellen für Einträge und deren Editoren	153
12.4.3 Die Klasse „Kontakt“ als Beispiel für einen Eintrag	154
12.4.4 Die Klasse „FactoryMethod“ als Client	155
12.5 Unterschied zur Abstract Factory	156
12.6 Zusammenfassung	157
13 Prototype Pattern	159
13.1 Objekte klonen	159
13.1.1 Kritik an der Realisierung	160
13.1.1.1 Das Interface „Cloneable“	160
13.1.1.2 Das Problem gleicher Referenzen	161
13.1.1.3 Was passiert beim Klonen?	162
13.1.2 In Vererbungshierarchien klonen	163
13.2 Ein größeres Projekt	166
13.2.1 Besprechung der ersten Version	167
13.2.2 Die zweite Version – deep Copy	169
13.2.3 Eigene Prototypen definieren	170
13.3 Zusammenfassung	171
14 Composite Pattern	173
14.1 Prinzip von Composite	173
14.2 Umsetzung 1: Sicherheit	174
14.3 Umsetzung 2: Transparenz	177
14.4 Betrachtung der beiden Ansätze	179
14.5 Einen Schritt weitergehen	181

14.5.1	Einen Cache anlegen	181
14.5.2	Die Elternkomponenten referenzieren	183
14.5.3	Knoten verschieben	186
14.6	Realisierung eines TreeModel	186
14.6.1	Die Methoden der Schnittstelle „TreeModel“	187
14.6.2	Knoten rendern und editieren	189
14.6.2.1	Knoten rendern	189
14.6.2.2	Knoten editieren	192
14.7	Zusammenfassung	195
15	Builder Pattern	197
15.1	Ein Objekt erzeugt andere Objekte	197
15.1.1	Umsetzung 1: Telescoping Constructor Pattern	197
15.1.2	Umsetzung 2: JavaBeans Pattern	199
15.1.3	Umsetzung 3: Builder Pattern	200
15.2	Ein komplexerer Konstruktionsprozess	202
15.2.1	XML-Dateien in ein TreeModel konvertieren	203
15.2.2	XML-Dateien als HTML darstellen	206
15.3	Zusammenfassung	209
16	Visitor Pattern	211
16.1	Ein einfaches Beispiel	211
16.1.1	Das Aggregat	211
16.1.2	Der Visitor	213
16.1.3	Der Client	214
16.1.4	Ein weiterer Visitor	215
16.1.5	Kritik am Projekt	215
16.2	Zusammenfassung	216
17	Memento Pattern	217
17.1	Aufgabe des Memento Patterns	217
17.2	Eine mögliche Realisierung	218
17.3	Ein größeres Projekt – der GrafikEditor	221
17.4	Zusammenfassung	222
18	Flyweight Pattern	225
18.1	Aufgabe des Patterns	225
18.2	Die Realisierung	226
18.3	Ein komplexeres Projekt: Pizza	228
18.3.1	Der erste Ansatz	228
18.3.2	Intrinsischer und extrinsischer Zustand	229
18.4	Flyweight in der Praxis	231
18.5	Zusammenfassung	233

19 Facade Pattern	235
19.1 Ein Beispiel außerhalb der IT	235
19.2 Die Fassade in einem Java-Beispiel	236
19.2.1 Einführung einer Fassade	238
19.2.2 Der Begriff „System“ genauer betrachtet	239
19.3 Die Fassade in der Klassenbibliothek	240
19.4 Das „Gesetz von Demeter“	241
19.5 Zusammenfassung	242
20 Adapter Pattern	243
20.1 Ein einleitendes Beispiel	243
20.1.1 Ein klassenbasierter Entwurf	243
20.1.2 Ein objektbasierter Entwurf	245
20.1.3 Kritik am Adapter Pattern	246
20.2 Ein Praxisbeispiel und falsche Namen	247
20.2.1 Ein Adapter im patternCoder	247
20.2.1.1 Objektbasierter Ansatz des Adapters	247
20.2.1.2 Klassenbasierter Ansatz des Adapters	248
20.2.2 Zum Schluss noch ein Etikettenschwindel	249
20.3 Zusammenfassung	250
21 Proxy Pattern	251
21.1 Virtual Proxy	251
21.2 Security Proxy	252
21.3 Smart Reference	253
21.3.1 Die Grundversion „Proxy_1“	253
21.3.2 Einführung eines Proxys im Projekt „Proxy_2“	255
21.3.3 Einen zweiten Proxy einführen	257
21.3.4 Dynamic Proxy	258
21.3.4.1 Das Reflection API	259
21.3.4.2 Der InvocationHandler	260
21.3.4.3 Die Proxy-Klasse	261
21.4 Remote Proxy	262
21.4.1 Aufbau von RMI grundsätzlich	262
21.4.2 Der RMI-Server	263
21.4.2.1 Die Schnittstelle „PiIF“	263
21.4.2.2 Die Serverklasse „PiImpl“	264
21.4.2.3 Die Klasse „ApplStart“ startet den Server	264
21.4.3 Der RMI-Client	265
21.4.4 Das Projekt zum Laufen bringen	267
21.5 Zusammenfassung	267

22	Decorator Pattern	269
22.1	Autos bauen – ein erstes Beispiel	269
22.1.1	Ein Attribut für jede Sonderausstattung	269
22.1.2	Mit Vererbung erweitern	270
22.1.3	Dekorieren nach dem Matroschka-Prinzip	270
22.1.3.1	Definieren der Grundmodelle	271
22.1.3.2	Definieren der Sonderausstattungen	271
22.1.3.3	Der Client steckt die Komponenten zusammen	272
22.1.4	Praxisbeispiele	273
22.1.4.1	Die Klasse „JScrollPane“	273
22.1.4.2	Streams in Java	274
22.2	Zusammenfassung	277
23	Bridge Pattern	279
23.1	Zwei Definitionen	279
23.1.1	Was ist eine Abstraktion?	279
23.1.2	Was ist eine Implementierung?	280
23.1.3	Ein Problem beginnt zu reifen	282
23.2	Das Bridge Pattern im Einsatz	283
23.2.1	Bridge Pattern – erster Schritt	284
23.2.2	Bridge Pattern – zweiter Schritt	285
23.2.2.1	Die Abstraktion erweitern	285
23.2.2.2	Die Implementierung erweitern	286
23.3	Diskussion des Bridge Patterns	288
23.3.1	Die Bridge in freier Wildbahn	288
23.3.2	Abgrenzung zu anderen Patterns	289
23.4	Zusammenfassung	289
24	Interpreter Pattern	291
24.1	Die Aufgabe in diesem Kapitel	291
24.2	Der Scanner	292
24.2.1	Die definierten Symbole	293
24.2.2	Der Scanner wandelt Strings in Symbole um	294
24.3	Der Parser	296
24.3.1	Abstrakte Syntaxbäume	296
24.3.2	Expressions für den Parser	297
24.3.3	Strichrechnung parsen	299
24.3.4	Punktrechnung parsen	301
24.3.5	Klammern berücksichtigen	303
24.4	Diskussion des Interpreter Patterns	304
24.5	Zusammenfassung	305
	Index	307