S06 – Mustergültige Zusammenarbeit

Geyer Stefan, Ritter Mathias

4AHIT – 18.12.2014

Inhaltsverzeichnis

Aufgabenstellung

Hier die Eckdaten zusammengefasst …

- Implementieren Sie die Quakologie!
- Erkennen Sie die verwendeten Muster!
- Wann und wo wurden die Muster eingesetzt?
- Erkläre die verwendeten Muster (kleiner Tipp, es sind deren sechs!)

Zeitaufzeichnung

Schätzung

Beschreibung	Zeitaufwand (Min.)
Implementierung	80
Testen	80
Protokoll	80
Gesamt	240 Minuten

Aufteilung

Geyer: ½ Testen, Protokoll → 120 Minuten
Ritter: Implementierung, ½ Testen → 120 Minuten

Tatsächlich benötigte Zeit

Geyer Stefan

Datum	Beschreibung	Zeitaufwand (Min.)
11.12.2014	Testcases erstellt	30
18.12.2014	Protokoll bearbeitet	50
11.01.2015	Protokoll bearbeitet	70
	Gesamt	150 Minuten

Ritter Mathias

Datum	Beschreibung	Zeitaufwand (Min.)
11.12.2014	Implementierung übernommen & modifiziert	45
18.12.2014	Testcases erstellt	70
11.01.2015	Protokoll bearbeitet	25
	Gesamt	140 Minuten

Gesamt

Insgesamt: 290 Minuten

Implementierung

Den Code für dieses Beispiel stellt der O'Reilly-Verlag zum Download zur Verfügung.[1] Wir haben diesen Code für unser Beispiel übernommen und leicht modifiziert.

Design-Patterns

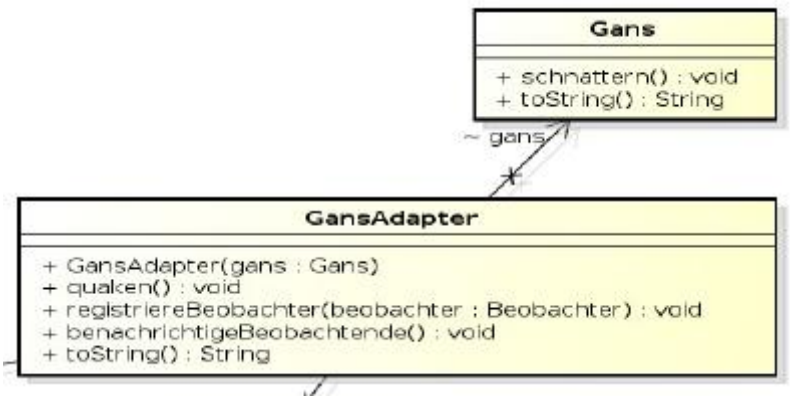
Adapter Pattern

Beim Adapter Pattern wird einer Klasse oder einem Interface, in dieser Veranschaulichung Grundinterface genannt, die Möglichkeit gegeben ein weiteres Interface, in dieser Veranschaulichung Gastinterface genannt, zu implementieren bzw. von einer weiteren Klasse zu erben ohne diese Änderungen dabei an der eigentlichen Klasse anzuwenden.
Die Adapter Klasse implementiert das gewünschte Interface bzw. erbt von der gewünschten Klasse und hat einen Konstruktor der eine Instanz des Grundinterfaces erwartet. Die implementierten Methoden des Gastinterfaces enthalten jeweils nur einen Aufruf auf eine Methode des Grundinterfaces.
Dadurch kann wird sichergestellt, dass die Klasse auch als Instanz des Gastinterface verwendet werden kann, allerdings muss das Grundinterface gleichzeitig dafür nicht verwendet werden.

Verwendung im Design

Gans – GansAdapter – Quackfaehig

Hierbei wird ein Adapter als Brücke zwischen Quackfaehig und Gans verwendet. Durch das oben beschriebene Verfahren wird die Methode schnattern() der Gans durch die Methode quacken() von Quackfaehig ersetzt.



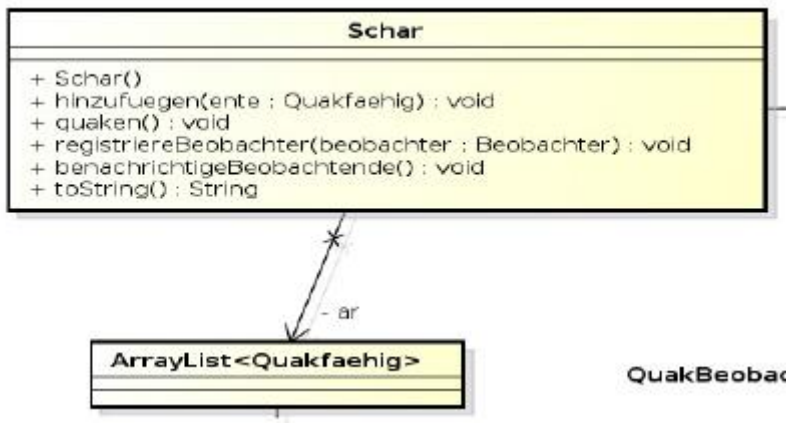
Composite Pattern

Das Composite Pattern ermöglicht es eine Verschachtelte Struktur an Elementen zu erstellen die trotzdem über ein Interface angesprochen werden können.
Eine Klasse implementiert ein Interface und hat gleichzeitig eine Collection oder ein Array vom Typ des implementierten Interfaces als Attribut. Die Klasse besitzt eine Möglichkeit Einträge hinzuzufügen und zu entfernen. Wird eine implementierte Methode der Klasse aufgerufen so wird durch alle Einträge der Liste durchgeloopt und jeweils die exakt gleiche Methode aufgerufen.
Falls einer der Einträge wieder eine Instanz eines Composite Patterns ist entsteht nun eine Verschachtelung die theoretisch unendlich lange fortgesetzt werden kann.
[HC10]

Verwendung im Design

Schar – Quackfaehig

Die Schar wird als Sammlung von Instanzen des Interfaces Quackfaehig definiert und kann, da es selbst auch Quackfaehig implementiert kann eine Instanz der Klasse theoretisch eine weitere hinzufügen.



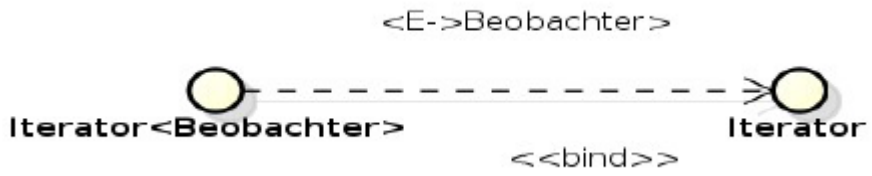
Iterator Pattern

Der Iterator erschafft eine Möglichkeit eine Reihe an Daten nach der Reihe abzufragen. Dabei stellt der Iterator pro Durchlauf ein Element aus der Reihe zur Verfügung. Eine Implementierung des Iterator Patterns beinhält eine .hasNext() und eine .next() Methode. Durch hasNext kann überprüft werden ob sich noch ein weiteres Element in der Reihe befindet. Mit next wird der Cursor au das nächste Objekt gesetzt und dieses wird zurückgegeben. Je nach Implementierung kann der Iterator entweder nur „nach vorne“ (vom Begin bis zum Ende) oder in beide Richtungen (man kann weiter und zurück springen).

Verwendung im Design

Iterator – Iterator<Beobachter>

Es wird ein Iterator für den Typ Beobachter erstellt welcher dann später benutzt wird um durch alle Vorhandenen Beobachter in der Methode benachrichtigeBeobachter der Klasse SenderRing zu iterieren.



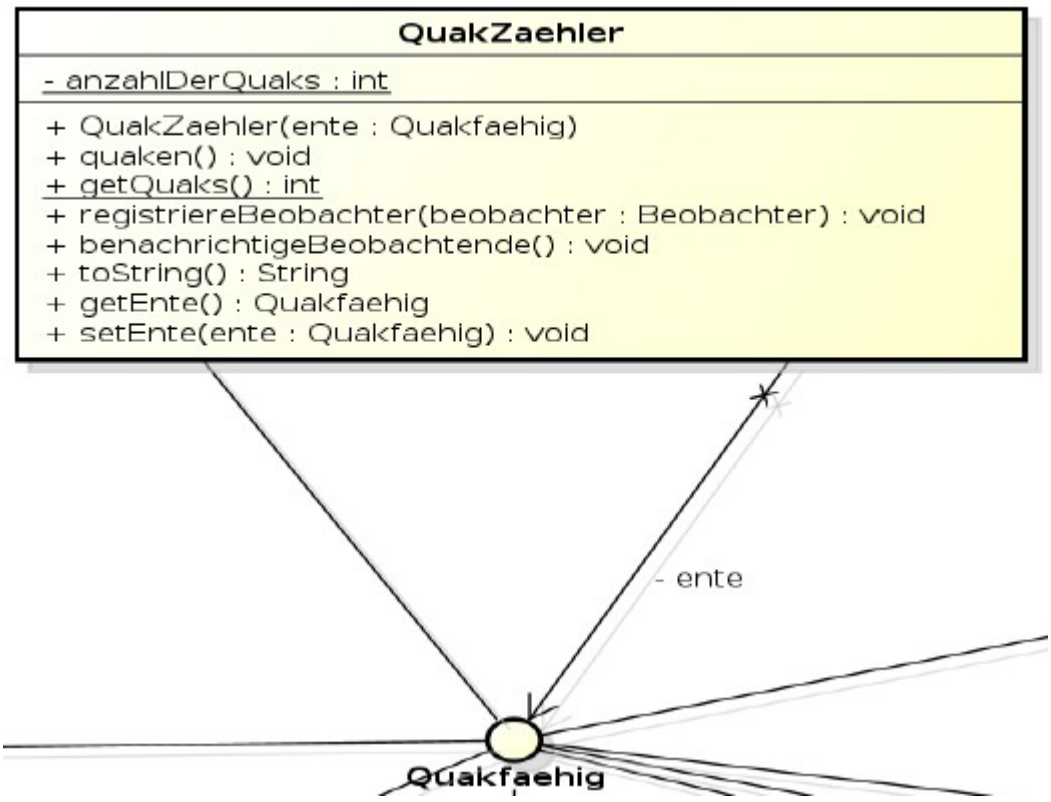
Decorator Pattern

Beim Decorator-Muster wird eine konkrete Komponente um beliebig viele konkrete Dekorierer erweitert. Die konkrete Komponente und der Dekorierer implementieren beide den Supertyp Komponente. Konkrete Dekorierer implementieren den Supertyp Dekorierer. Konkrete Dekorierer enthalten eine Instanzvariable für die Komponente, die sie dekorieren. Beim Dekorieren erweitern Sie den Zustand der gespeicherten Variablen, indem sie vor oder nach Abruf dieses Zustandes ihr eigenes Verhalten hinzufügen. Den konkreten Dekorierern ist es egal, ob sie eine konkrete Komponente oder eine bereits dekorierte Komponente erweitern.
[HD10, FFSB06]

Verwendung im Design

QuakZaehler – Quackfaehig

Der QuackZaehler dekoriert ein Quackfaehig Attribut. Da die Klasse auch Quackfaehig implementiert, kann diese Klasse wie das Übergebene Attribut behandelt und von anderen Klasen als solches verwendet werden.

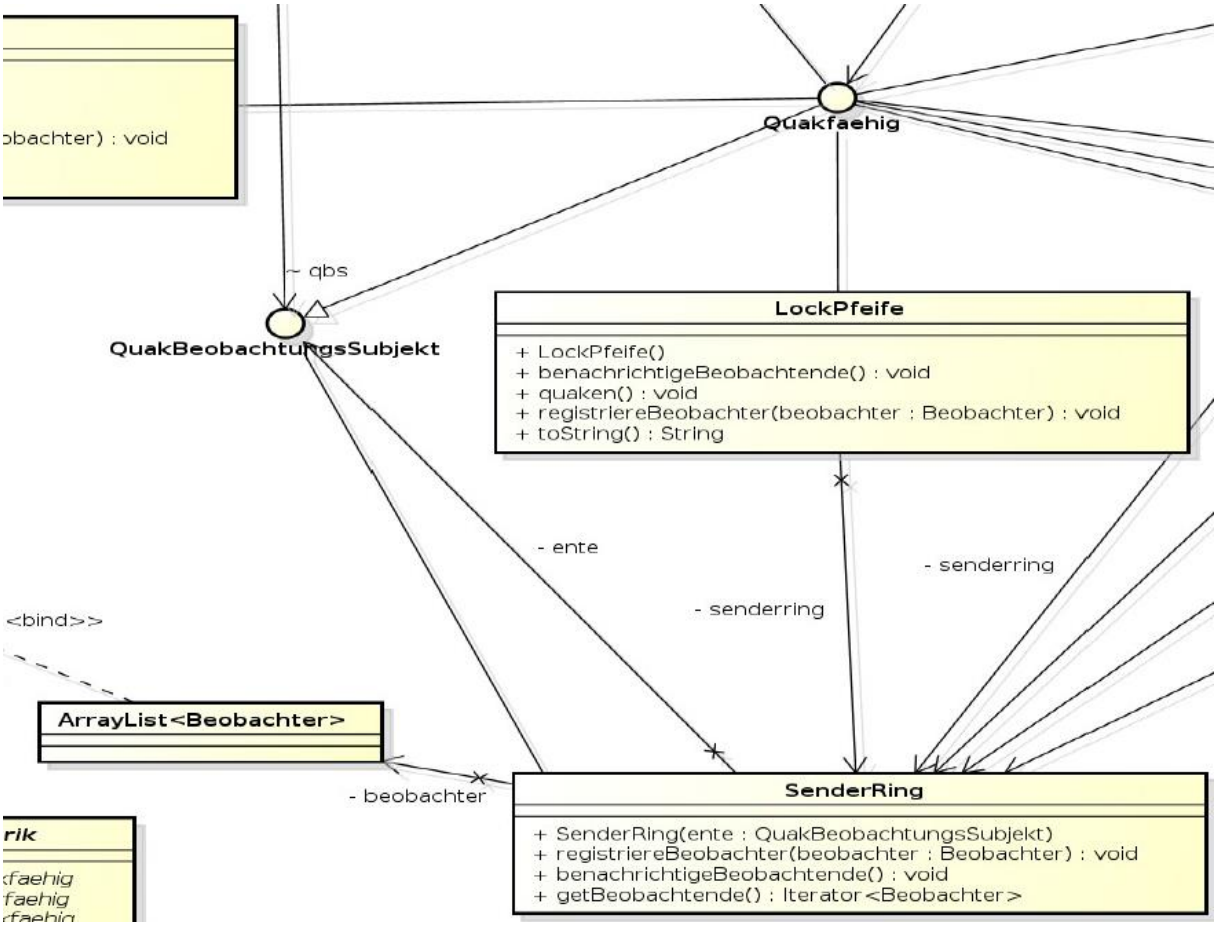


Observer Pattern

Das Observer-Muster wird eingesetzt, wenn mehrere Observer von einem Observable (Eins-zu-viele-Beziehung) benachrichtigt werden sollen. Das kann notwendig sein, wenn sich z.B. sein Zustand geändert hat. Die Observer können sich beim Observer an- und wieder abmelden durch entsprechende Methoden (von einer Schnittstelle vorgeschrieben). Das Observable benachrichtigt die Observer über eine update-Methode (von einer Schnittstelle vorgeschrieben). Die Informationen können vom Observable zum Observer entweder nach dem Pull-Ansatz (Observer fragt nach und gibt Daten an, die er haben möchte) oder nach dem Push-Ansatz (Alle Daten werden zu den Observern übertragen) übermittelt werden.
[HO10, FFSB06]

Verwendung im Design

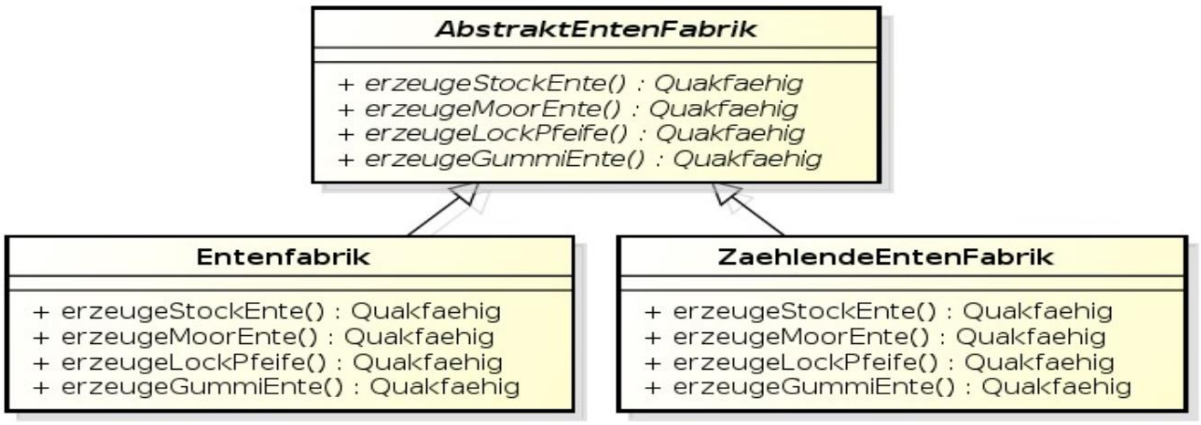
Quakfaehig extended nebenbei noch QuakBeobachtungsSubjekt. Die Klasse SenderRing (welche Gleichzeitig auch noch dem Design eines Decorator Patterns entspricht) speichert alle hinzugefügten Einträge in einer Liste und beinhält eine Möglichkeit alle Einträge zu benachrichtigen. LockPfeife enthält ein SenderRing Attribut welches den Observer aufrufen kann.



Abstract Factory

Beim Abstract-Factory-Muster werden eine oder mehrere Familien zusammenhängender Produkte definiert. Die abstrakte Fabrik bietet die Schnittstelle, welche von allen konkreten Fabriken implementiert werden muss. Jede konkrete Fabrik kann ein konkretes Produkt einer Produktfamilie produzieren. Der Client nutzt als Schnittstelle nur die abstrakte Fabrik, um ein Objekt zu erlangen. [FFSB06, HAF10]

Verwendung im Design



Literaturverzeichnis

[1]
O'Reilly: Deutsche Code-Beispiele zu "Head First Design Patterns" [Online].
Verfügbar unter:
http://examples.oreilly.de/german_examples/hfdesignpatger/
[abgerufen am 11.12.2014]

[HD10]
Philipp Hauer: Das Decorator Design Pattern [Online].
Verfügbar unter:
http://www.philipphauer.de/study/se/design-pattern/decorator.php
[abgerufen am 10.12.2014]

[HC10]
Philipp Hauer: Das Composite Design Pattern [Online].
Verfügbar unter:
http://www.philipphauer.de/study/se/design-pattern/composite.php
[abgerufen am 11.12.2014]

[SAF]
SourceMaking: Abstract Factory Design Pattern [Online].
Verfügbar unter:
http://sourcemaking.com/design_patterns/abstract_factory
[abgerufen am 10.12.2014]

[SO]
SourceMaking: Observer Design Pattern [Online].
Verfügbar unter:
http://sourcemaking.com/design_patterns/observer
[abgerufen am 10.12.2014]

[SA]
SourceMaking: Adapter Design Pattern [Online].
Verfügbar unter: http://sourcemaking.com/design_patterns/adapter
[abgerufen am 14.12.2014]

[SD]
SourceMaking: Decorator Design Pattern [Online].
Verfügbar unter: http://sourcemaking.com/design_patterns/decorator
[abgerufen am 10.12.2014]