Softwarearchitektur, UML, Design Patterns und Unit Tests

Stefan Wehr
Prof. Dr. Peter Thiemann

7. Dezember 2005

Übersicht

- Softwarearchitektur
- UML
- Design Pattern
- Unit Tests

Softwarearchitektur

Es gibt verschiedene Softwarearchitekturen

- Batch Sequential
- Pipes & Filters
- Layered System
- ...
- Object Oriented Organization

Object Oriented Organization

Das System besteht aus Objekten, die mittels Methodenaufrufe (Nachrichten) miteinander kommunizieren.

Übersicht

- Softwarearchitektur
- UML
- Design Pattern
- Unit Tests

Wichtige Konzepte

Klassen
- Definieren Attribute und Methoden
- Werden zur Laufzeit instanziert
  $\Rightarrow$ Instanzen / Objekte

Interfaces
- Spezifizieren die Schnittstelle einer Klasse
- Enthalten keinen Code
- Können nicht instanziert werden

Klassendiagramme

- Repräsentieren Klassen und statische Beziehungen zwischen Klassen
- Keine zeitliche Informationen
- Ein Klassendiagramm ist ein Graph mit
  - Knoten: Klassen (Rechtecke)
  - Kanten: Beziehungen zwischen Klassen
- Kann auch Interfaces, Packages und Instanzen enthalten

UML

- Unified Modeling Language (Booch / Jacobson / Rumbaugh)
- Stellt verschiedene Diagrammarten zur Verfügung, um unterschiedliche Aspekte eines Systems zu modellieren
- Hier: nur Klassendiagramme

Literatur

Martin Fowler, UML distilled: a brief guide to the standard object modeling language

Klasse

Student
matriculation number
name
grades
count
issue certificate ()
enter grade ()
list degrees ()

Namensabteilung
Attribute

Operationen
(Methoden)

- Nur Name obligatorisch
- Weitere Abteilungen möglich (Verantwortlichkeiten, Ereignisse, Ausnahmen, . . .)

Inhalt der Namensabteilung

- Optionale Stereotypen
  «interface», «controller»
- Klassenname,
  abstrakte Klassen werden *kursiv* geschrieben
- . . .

Attributabteilung

Syntax von Attributen

Sichtbarkeit Name : *Typ* [*Kardinalität Ordnung*] = *Initialwert* { *Eigenschaften* }

Sichtbarkeit	`+`, `#`, `-`, `~`
Kardinalität	Menge natürlicher Zahlen
Ordnung	`ordered` / `unordered`
Eigenschaften	z.B.: `{frozen}`

- Statische Attribute werden unterstrichen

Sichtbarkeit

- `+`, `public`
- `#`, `protected`
- `-`, `private`
- `~`, `package`

Operationsabteilung

Syntax

Sichtbarkeit Name (*Parameterliste*) : *Rückgabetype*
{ Eigenschaften }

Sichtbarkeit	`+`, `#`, `-`, `~`
Parameterliste	*Art Name* : *Typ*
	Art ∈ `in`, `out`, `inout`
Eigenschaften	z.B.: `{query}`
	`{concurrency=...}`
	`{abstract}`

- Statische Operationen werden unterstrichen

Kardinalität

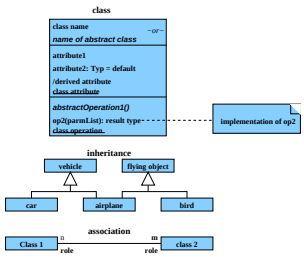
Definiert Menge natürlicher Zahlen

Kardinalität	::=	*Intervall* | *Kardinalität*,*Kardinalität*
Intervall	::=	*int*..*int** | *int**
*int**	::=	*int* | `*`

Beispiele:

- `1`, `0..1`, `0..*`, `1..*`, `*`
- `1,3..5,7..10,15, 19..*`

Beispiele (1)



Beziehungen

Binäre Beziehungen

- “Zusammenarbeit” zweier Klassen
- Durchgezogene Linie zwischen zwei Klassen
- Optional: Name für die Beziehung, Rolennamen, Navigation, Kardinalität

Generalisierung/Vererbung

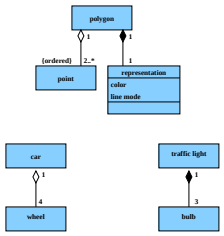
- Spezifiziert Subklassenbeziehung
- Durchgezogene Linie mit Pfeil auf Superklasse

Aggregation und Komposition (1)

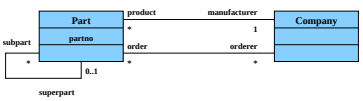
Aggregation

- Drückt eine “ist Teil von” Beziehung aus
- Bedeutung: Inhalt “gehört” einem Container
- Notation: Kante mit weißem Rombus als Pfeilkopf

Beispiel



Beispiele (2)



Aggregation und Komposition (2)

Komposition

- Spezieller Fall von Aggregation
- Container und Inhalt “leben und sterben” zusammen
- Notation: Kante mit schwarzem Rombus als Pfeilkopf

Übersicht

- Softwarearchitektur
- UML
- Design Pattern
- Unit Tests

Design Patterns

- Gamma, Helm, Johnson, Vlissides: Design Patterns, Elements of Reusable Object-Oriented Software, Addison Wesley, 1995. “gang of four”
- Häufig auftretende Muster in der Zusammenarbeit von Objekten
- Ziele: Flexibilität, Wartbarkeit, Kommunikation, Wiederverwendung
- Alternatives Vorgehen und Kombinationen möglich

Klassifi kation von Patterns (1)

Zweck

Creational Patterns	Abstrahieren über das Erzeugen von Objekten
Structural Patterns	Abstrahieren über häufig vorkommende Strukturen
Behavioral Patterns	Abstrahieren über Verhaltensmuster

Klassifi kation von Patterns (2)

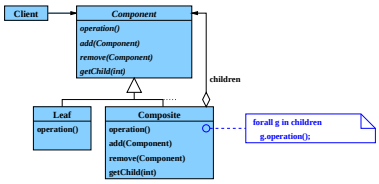
Wirkungsbereich

Klassen	Statische Beziehungen zwischen Klassen (Vererbung)
Objekte	Dynamische Beziehungen zwischen Instanzen von Klassen

Motivation

- Dateisystem
  - Container Verzeichnisse
    - Inhalt Verzeichnisse, Dateien
- Arithmetische Ausdrücke
  - Container Operatoren
    - Inhalt Operatoren, Konstanten, Variablen

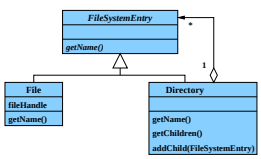
Allgemeine Struktur



Composite Pattern

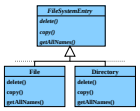
- Structural Pattern
- Rekursive Objektstrukturen
- Uniforme Behandlung von Container und Inhalt

Beispiel



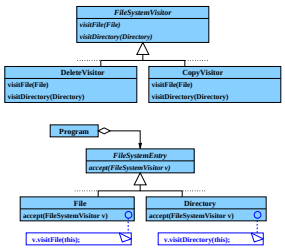
Motivation

- Operation eines Filesystems: Löschen, Kopieren, Liste aller Namen bestimmen, . . .
- Naiver Ansatz: Operationen werden in den Klassen `Directory` und `File` implementiert



Problem: Für neue Operationen müssen `Directory` und `File` geändert werden

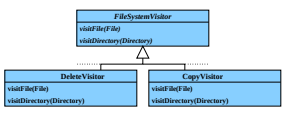
Beispiel mit Visitor



Visitor Pattern

- Behavioral Pattern
- Operationen auf einer Objektstruktur werden durch Objekte repräsentiert
- Hinzufügen neuer Operationen ohne Änderung der Klassen

Beispiel mit Visitor



Unit Tests

Idee

- Schreibe vor (bzw. während) der Implementierung einer Klasse eine Testklasse
- Archiviere alle Tests, so dass sie später wiederholt werden können

Nutzen

- Software enthält weniger Fehler
- Software kann leichter geändert werden
- Testklasse dient als eine Art Spezifikation für die eigentliche Klasse

Übersicht

- Softwarearchitektur
- UML
- Design Pattern
- Unit Tests

JUnit