

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.

Wichtige Konzepte

Klassen

- Definieren Attribute und Methoden
- Werden zur Laufzeit instanziiert
⇒ Instanzen / Objekte

Interfaces

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

Übersicht

- Softwarearchitektur
- UML
- Design Pattern
- Unit Tests

UML

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

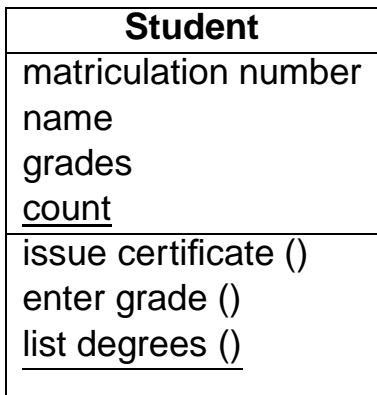
Literatur

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

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

Klasse



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 }*

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

- Statische Attribute werden unterstrichen

Sichtbarkeit

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

Kardinalität

Definiert Menge natürlicher Zahlen

$$\begin{array}{l}
 \textit{Kardinalität} ::= \textit{Intervall} \mid \textit{Kardinalität}, \textit{Kardinalität} \\
 \textit{Intervall} ::= \textit{int}.. \textit{int}^* \mid \textit{int}^* \\
 \textit{int}^* ::= \textit{int} \mid *
 \end{array}$$

Beispiele:

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

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

Beziehungen

Binäre Beziehungen

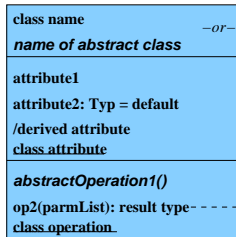
- “Zusammenarbeit” zweier Klassen
- Durchgezogene Linie zwischen zwei Klassen
- Optional: Name für die Beziehung, Rolennamen, Navigation, Kardinalität

Generalisierung/Vererbung

- Spezifiziert Subklassenbeziehung
- Durchgezogene Linie mit Pfeil auf Superklasse

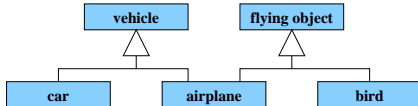
Beispiele (1)

class

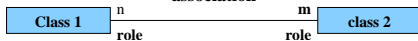


implementation of op2

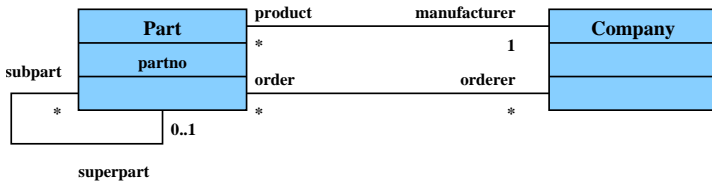
inheritance



association



Beispiele (2)



Aggregation und Komposition (1)

Aggregation

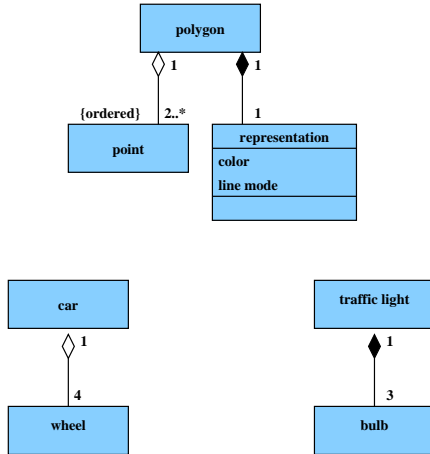
- Drückt eine “ist Teil von” Beziehung aus
- Bedeutung: Inhalt “gehört” einem Container
- Notation: Kante mit **weißem** Rombus als Pfeilkopf

Aggregation und Komposition (2)

Komposition

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

Beispiel



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

Klassifikation von Patterns (1)

Zweck

Creational Patterns Abstrahieren über das Erzeugen von Objekten

Structural Patterns Abstrahieren über häufig vorkommende Strukturen

Behavioral Patterns Abstrahieren über Verhaltensmuster

Klassifikation von Patterns (2)

Wirkungsbereich

Klassen Statische Beziehungen zwischen Klassen (Vererbung)

Objekte Dynamische Beziehungen zwischen Instanzen von Klassen

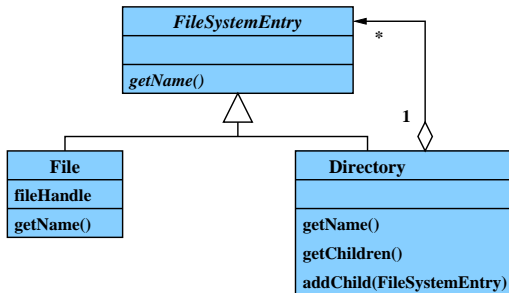
Composite Pattern

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

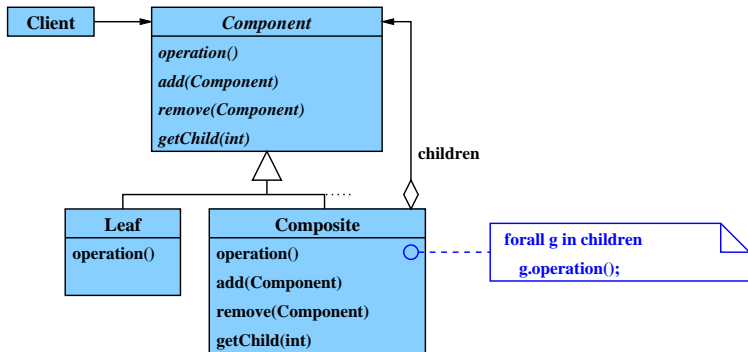
Motivation

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

Beispiel



Allgemeine Struktur

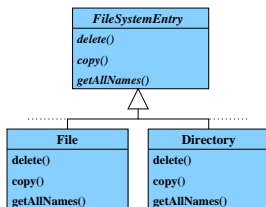


Visitor Pattern

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

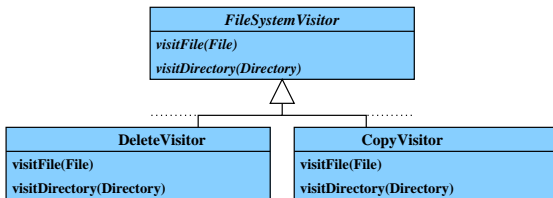
Motivation

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

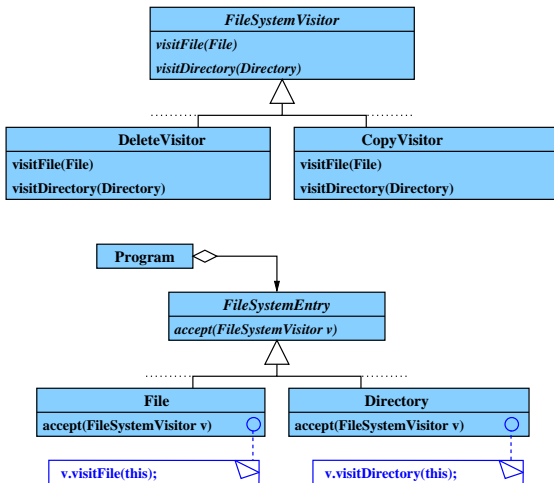


Problem: Für neue Operationen müssen `Directory` und `File` geändert werden

Beispiel mit Visitor



Beispiel mit Visitor



Übersicht

- Softwarearchitektur
- UML
- Design Pattern
- **Unit Tests**

Unit Tests

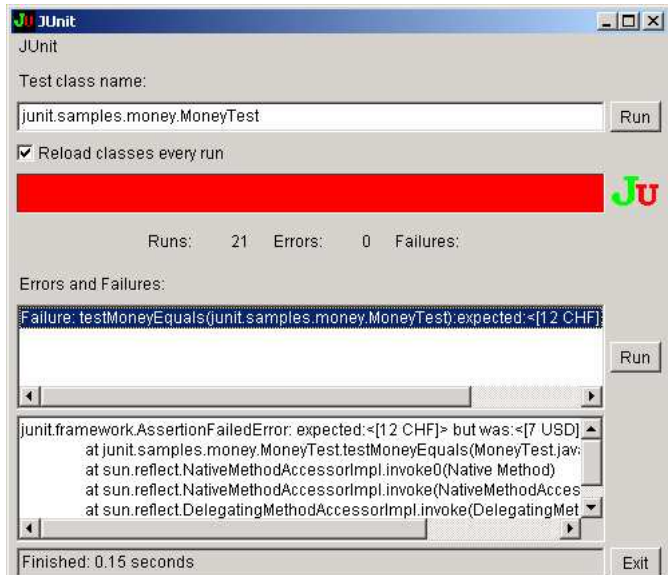
Idee

- 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

JUnit



The screenshot shows the JUnit GUI window. At the top, the title bar says "JUnit". Below it, the text "JUnit" is displayed. The "Test class name:" field contains "junit.samples.money.MoneyTest". A "Run" button is to the right of this field. Below the field, there is a checked checkbox labeled "Reload classes every run". A large red progress bar is visible, followed by the JUnit logo. Below the progress bar, the statistics "Runs: 21 Errors: 0 Failures:" are shown. The "Errors and Failures:" section contains a text area with the following text: "Failure: testMoneyEquals(junit.samples.money.MoneyTest): expected: <[12 CHF]". Below this text area is a "Run" button. At the bottom of the text area, the stack trace is visible: "junit.framework.AssertionFailedError: expected: <[12 CHF]> but was: <[7 USD] at junit.samples.money.MoneyTest.testMoneyEquals(MoneyTest.java: at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method) at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAcces at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMet". Below the text area is a scroll bar. At the bottom of the window, the text "Finished: 0.15 seconds" is displayed, and an "Exit" button is on the right.

JUnit

JUnit

Test class name:

junit.samples.money.MoneyTest

Run

Reload classes every run

Runs: 21 Errors: 0 Failures:

Errors and Failures:

Failure: testMoneyEquals(junit.samples.money.MoneyTest): expected: <[12 CHF]

Run

junit.framework.AssertionFailedError: expected: <[12 CHF]> but was: <[7 USD]
at junit.samples.money.MoneyTest.testMoneyEquals(MoneyTest.java:
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAcces
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMet

Finished: 0.15 seconds

Exit