

Projektgruppe
Entwurfsmuster für den Informatikunterricht (EMI)

**Dokumentation für die Lernumgebung
Pattern Park**

Eingereicht von:

Demian Franke, Lars Friedrich, Florian Haug,
Benjamin Klein, Rudolf Koslowski, Jonathan Ufer

Fachgruppe Didaktik der Informatik und E-Learning
Fachbereich 12, Universität Siegen

Betreuung durch:

Dipl.-Inf. Peer Stechert, Dipl.-Ing. Stefan Freischlad

Inhaltsverzeichnis

1 Einleitung.....	3
1.1 Einführung und Übersicht über die Lernsoftware	3
1.2 Beschreibung des Szenarios Freizeitpark.....	5
1.3 Beschreibung des Aufbaus der Module.....	6
1.4 Fachdidaktischer Ansatz dieser Lernsoftware.....	11
2 Modul Informationshaus	13
2.1 Funktionalität und Aufgaben des Moduls	13
2.2 Konfigurationsmöglichkeiten des Gesamtprogramms	13
2.3 Animation Entwurfsmuster.....	14
2.4 Animation Pattern Park.....	15
2.5 Glossar	16
2.6 Links	21
2.7 Musterkombinationsaufgabe	22
3 Modul Kompositum.....	25
4 Modul Beobachter	33
5 Modul Fassade	43
6 Modul Proxy.....	51
7 Modul Dekorierer	59
8 Modul Iterator	67
9 Modul Schablonenmethode.....	73
10 Modul Zustand.....	79
11 Technische Umsetzung.....	87
12. Zusammenfassung und Ausblick.....	93
Literaturverzeichnis	95

1 Einleitung

1.1 Einführung und Übersicht über die Lernsoftware

Diese Dokumentation beschreibt den Aufbau und die Funktionalität von Pattern Park, einer Lernsoftware, die primär an Schulen im Informatikunterricht der Sekundarstufe II eingesetzt werden soll, um dort das Verstehen von Informatiksystemen und die Lehre von objektorientierten Entwurfsmustern der Informatik zu unterstützen und zu fördern. Es handelt sich bei dieser Lernsoftware um eine Kombination aus deskriptiven, interaktiven und explorativen Programmteilen mit unterschiedlich hohem Freiheitsgrad.

Aus [GHJV95] wurden acht Entwurfsmuster ausgewählt. Ein sehr wichtiges Kriterium bei dieser Auswahl war die Frage, inwieweit in den Entwurfsmustern grundlegende informatische und objektorientierte Konzepte zu finden sind, da diese den Bildungswert erhöhen. Demzufolge sollen diese Konzepte auch einen Schwerpunkt dieser Lernsoftware bilden, während die Entwurfsmuster selber und ihre Verwendungszwecke nur nebenrangig sind. Ein weiteres wichtiges Auswahlkriterium war die Frage, ob sich das Entwurfsmuster anhand eines Beispiels veranschaulichen lässt. Dieses Beispiel sollte Schülern aus dem Alltag bekannt sein, in das Gesamtszenario Freizeitpark passen (vgl. Abschnitt 1.3) und nicht aus der Informatik-Welt stammen.

[A1.1.1] zeigt eine Übersicht über die ausgewählten Entwurfsmuster, ihnen inhärente informatische Konzepte und die jeweiligen Beispiele.

Entwurfsmuster	Beispiel	Konzepte
Kompositum	Grafik / Jugendgruppe	Rekursion, Datenstruktur Baum Aggregation, Komposition
Beobachter	1 Uhrzeit – mehrere Uhren	Assoziation (1-zu-n) Darstellung von Daten
Fassade	Pförtner	Datenkapselung Schnittstelle, Delegation
Proxy	Geldkarte	Zugriffsschutz, Schnittstelle
Dekorierer	Werkzeuge und Arbeitskleidung von Mitarbeitern	Rekursion, Vererbung
Iterator	Durchlaufen einer Besucherliste	Datenstruktur Liste Traversierung
Schablone	Aufbau von Bahnen	Vererbung
Zustand	Achterbahnfahrt	Zustandsdiagramm Zustände, Bedingungen, Aktionen

[A1.1.1] Übersicht über die ausgewählten Entwurfsmuster

Das Hauptmenü dieser Lernsoftware ähnelt einer Landkarte, der Benutzer (Alle Personenbezeichnungen gelten gleichermaßen für die weibliche und männliche Form.) betrachtet an-

fangs einen Freizeitpark aus der Vogelperspektive, eine genauere Beschreibung des Szenarios und der Karte findet man in Abschnitt 1.3.

Durch diese Übersicht kann man verschiedene Bereiche des Parks aufrufen, die jeweils ein Modul repräsentieren, in dem ein Entwurfsmuster behandelt wird. Wie die Module im Allgemeinen aufgebaut sind, wird in Abschnitt 1.4 beschrieben.

Eine besondere Rolle in der Karte des Freizeitparks nimmt das „Informationshaus“ ein, das sich etwa in der Mitte des Parks befindet. Dieses Modul wird in Abschnitt 2 ausführlich erläutert. Hierüber sind alle Elemente der Lernsoftware erreichbar, die sich keinem speziellen Entwurfsmuster zuordnen lassen. Dies sind Konfigurationsmöglichkeiten (siehe 2.2), eine allgemeine Einführung in Entwurfsmuster (siehe 2.3), eine Einführung in den Pattern Park (siehe 2.4), ein Glossar (siehe 2.5), Links zu Internetseiten zum Thema (siehe 2.6), sowie Metadaten der gesamten Lernsoftware. Über einen Wegweiser in der Nähe des Informationshauses gelangt man zu einer Übung, in der verschiedene Entwurfsmuster miteinander kombiniert werden (siehe 2.7).

Das Glossar und die Verknüpfungen dienen auch als Unterstützung zum autodidaktischen Lernen, um einen Einsatz der Lernsoftware auch in anderen Kontexten – außerhalb der Schule – zu ermöglichen.

Es werden für jedes Entwurfsmuster unterschiedliche Übungen für den Benutzer angeboten. Hier handelt es sich um Übungen ohne Unified Modelling Language (UML) und UML-Puzzle. Für die Entwurfsmuster Beobachter, Fassade, Proxy, Dekorierer und Zustand gibt es neben den UML-Puzzles zusätzlich eine weitere Übung mit UML.

Durch die Modularisierung – also die Aufteilung des Programms in Module für jedes Entwurfsmuster, zu denen es wiederum weitere Module gibt – können die bereits vorhandenen Entwurfsmuster überarbeitet und erweitert werden. Die Linkseite und der Glossar sind im HTML-Format und können daher leicht editiert werden.

Hinweise und Fehlermeldungen auf der Benutzungsoberfläche sind vom Quellcode entkoppelt und werden aus externen XML-Dateien eingelesen, wodurch eine Übersetzung in andere Sprachen und ein Überarbeiten der Meldungen leicht möglich ist.

Das zur Lernsoftware gehörige **Handbuch** enthält Hinweise zur Bedienung sowie alle Einstellungs- und Konfigurationsmöglichkeiten.

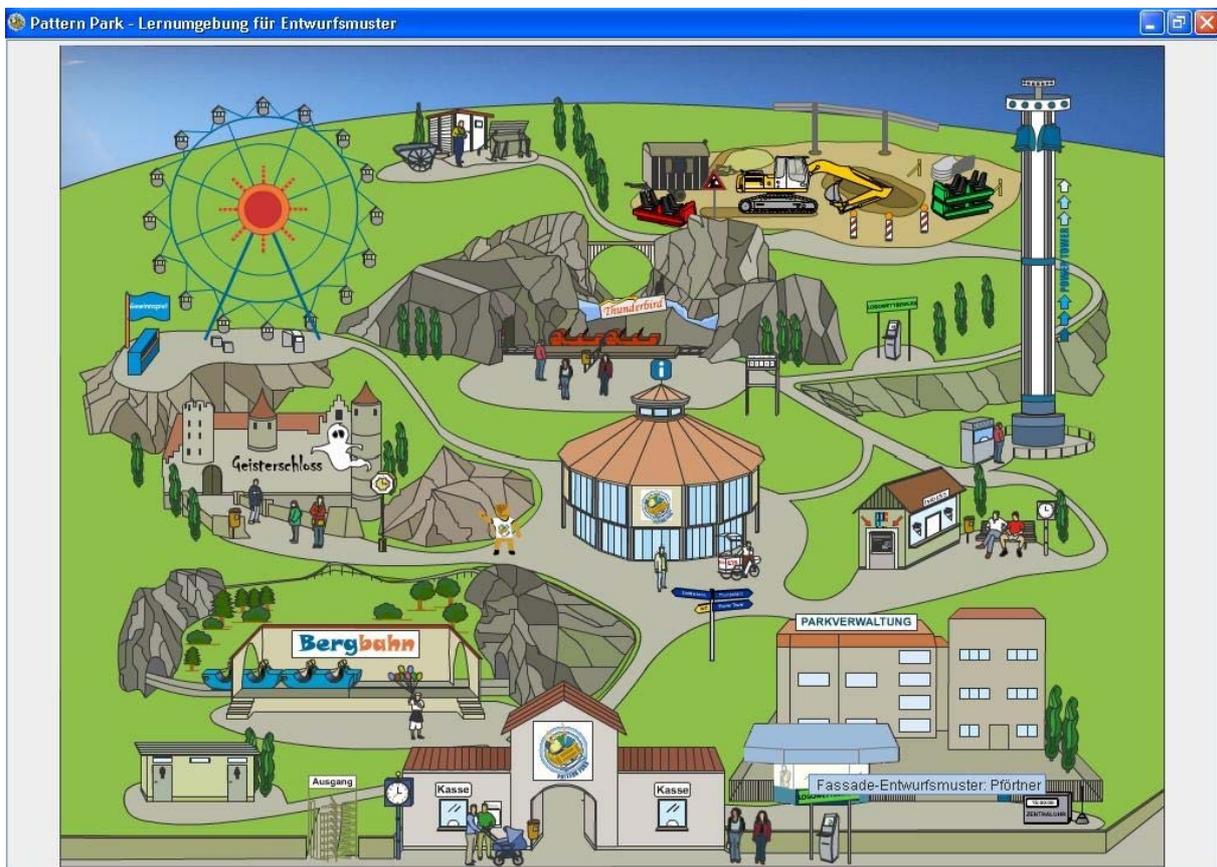
Nach dem Starten des Programms kann ein kurzer Einführungsfilm betrachtet oder übersprungen werden. Dieser Film kann später auch über das Informationshaus aufgerufen werden.

Für die Lernsoftware sind mindestens 256 MB Arbeitsspeicher (512 MB empfohlen) und mindestens 150 MB freier Festplattenspeicher für Programm- und Mediendateien sowie eine Bildschirmauflösung von mindestens 1024 x 768 Pixel erforderlich.

1.2 Beschreibung des Szenarios Freizeitpark

Um einen lebensweltnahen Bezug zur Thematik der Entwurfsmuster herzustellen, wurde ein Szenario gesucht, welches folgende Anforderungen erfüllen soll:

- Lebensweltbezug gemäß der Zielgruppe,
- Gendersensitivität,
- enthält möglichst viele Entwurfsmuster,
- veranschaulicht Fundamentale Ideen der Informatik, vgl. [Schwill93].



[A1.2.1] Die Übersichtskarte des Freizeitparks mit aktiviertem Kontextmenü zur Fassade

Ein Szenario, welches diesen Anforderungen weitestgehend gerecht wird, ist die Sicht auf einen Freizeitpark als abgeschlossenes System der Lebenswelt. Dabei werden die Fahrattraktionen und Ereignisse im Freizeitpark mit den Eigenschaften von Entwurfsmustern verknüpft. Auf einer Karte des Freizeitparks, die als Ausgangspunkt der gesamten Lernsoftware fungiert, werden diese einzelnen Objekte übersichtlich dargestellt. Dabei kann der Benutzer

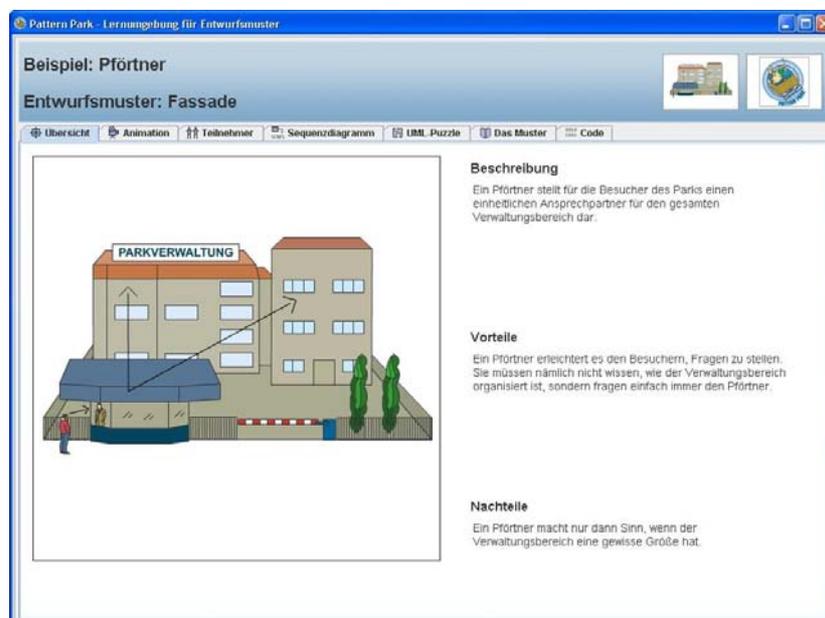
über diese Karte die einzelnen Bereiche im Freizeitpark, etwa den Verwaltungsbereich anwählen und gelangt so zu den Modulen, in dem ein bestimmtes Entwurfsmuster – in diesem Fall Fassade – anhand des Verwaltungsbereichs erklärt wird, vgl. [A1.2.1].

Wählt der Benutzer mit seiner Maus ein Gebäude an, wird ein Kontexthinweis angezeigt, der das angewählte Modul kurz beschreibt. Wird diese Verlinkung per Mausklick aktiviert, gelangt man zur Übersichtsseite des Entwurfsmusters.

1.3 Beschreibung des Aufbaus der Module

Wie in Abschnitt 1.1 bereits erwähnt, repräsentiert jedes Modul ein Entwurfsmuster, wobei dessen Aufbau und die Navigation innerhalb der einzelnen Module immer gleich sind.

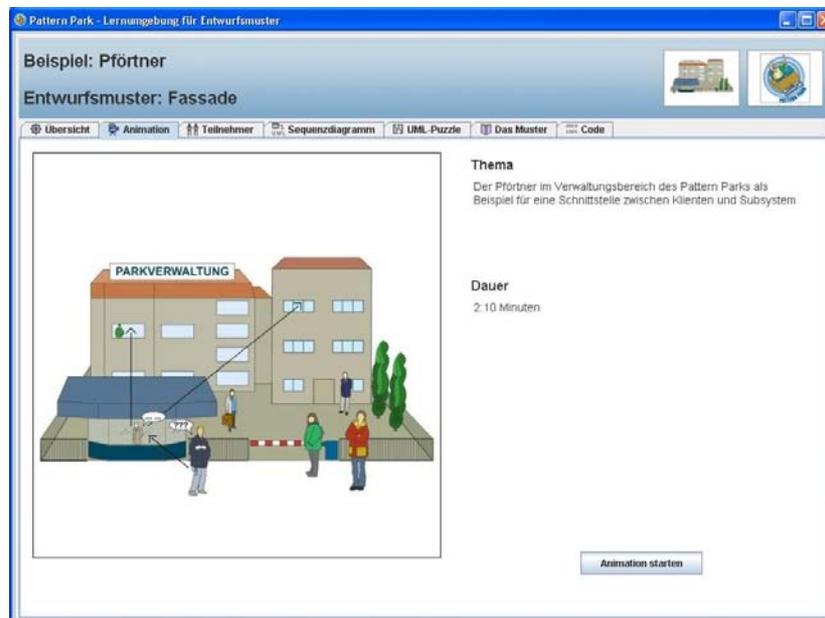
Nach dem Auswählen eines Elements auf der Freizeitparkkarte gelangt man in ein Modul, in dem das entsprechende Entwurfsmuster behandelt wird. Zu Beginn befindet man sich auf der Übersichtsseite, auf der das Entwurfsmuster anhand des Beispiels im Freizeitpark kurz erläutert wird, vgl. [A1.3.1]. Per Mausklick auf das Symbol des Pattern Parks rechts oben gelangt man zur Gesamtübersicht – der Parkkarte – zurück. Über die Reiter im oberen Bereich des Bildschirms können die weiteren Untermodule ausgewählt werden.



[A1.3.1] Screenshot der Übersichtsseite zur Fassade

Auf der **Übersichtsseite** wird das Muster anhand des Beispiels prägnant in einem Satz erklärt und seine Vor- und Nachteile werden benannt. Ein Bild veranschaulicht die Funktionsweise und fungiert als Logo des Musters, vgl. [A1.3.1]. Diese Übersichtsseiten sollen dem

Benutzer helfen, sich die Ideen der Entwurfsmuster zu einem späteren Zeitpunkt wieder ins Gedächtnis zu rufen. Die Inhalte der Übersichtsseiten sind in den Abschnitten 3.1 – 10.1 beschrieben.



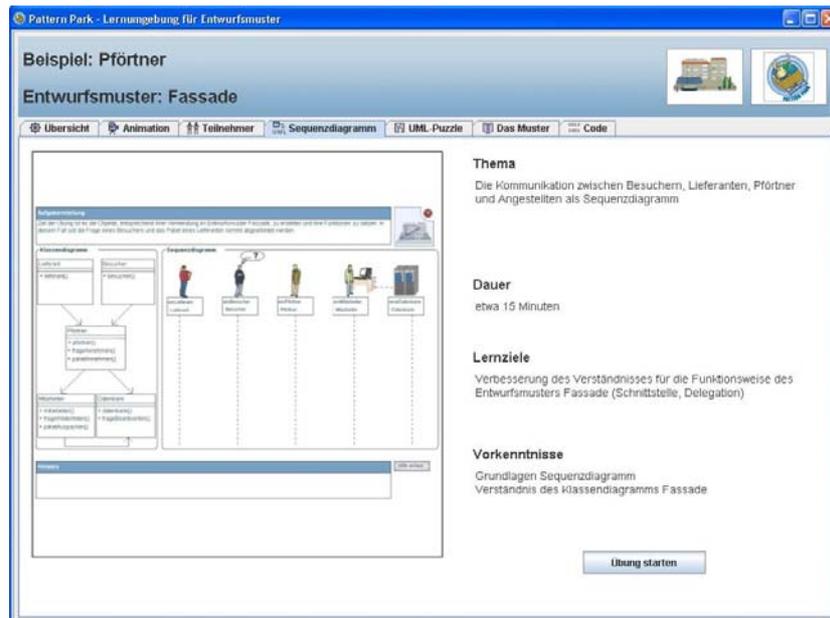
[A1.3.2] Screenshot der Übersichtsseite zur Animation zur Fassade

Von der nächsten Seite kann ein einführender **Animationsfilm** gestartet werden, der das Muster anhand des Beispiels aus dem Park anschaulich erklärt. Ein Screenshot, sowie Dauer und Thema des Films befinden sich auf dieser Startseite, vgl. [A1.3.2]. Es gibt jeweils zwei Animationen, eine mit Ton und eine ohne Ton und mit Untertiteln. Über das Informationshaus kann ausgewählt werden, welche Filme gestartet werden sollen. Die Animationen werden in den Abschnitten 3.3 – 10.3 beschrieben.

Es folgen die Startseiten der Übungen, auf denen Vorkenntnisse, Lernziele, Thema und ungefähre Dauer der Übung aufgelistet werden. Ein Screenshot gibt einen ersten Einblick in die Übung, vgl. [A1.3.3].

Innerhalb jedes Moduls gibt es eine **Übung ohne UML-Diagramme**. Diese soll die Funktionsweise des entsprechenden Entwurfsmuster anhand des Beispiels erklären. Häufig muss hierbei die richtige Auswahl aus verschiedenen Vorgaben getroffen werden. Die einzelnen Übungen ohne UML werden in den Abschnitten 3.4 – 10.4 beschrieben.

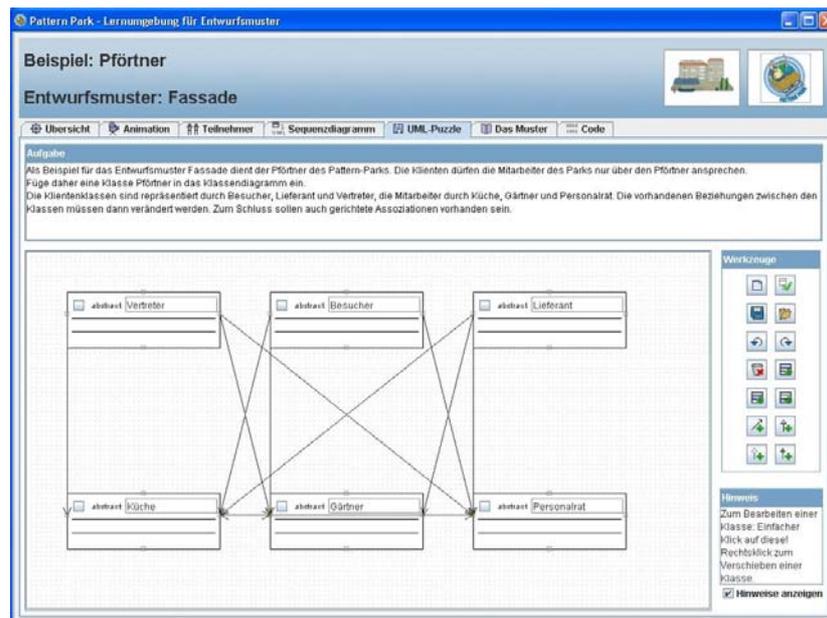
Zu einigen Entwurfsmustern gibt es auch **Übungen mit UML**, in denen die Funktionsweise des Musters aus objektorientierter Sicht mithilfe von Sequenz- oder Klassendiagrammen thematisiert wird. Im Modul Zustand wird ein Zustandsdiagramm zu einer Bahnfahrt behandelt. Die einzelnen Übungen mit UML werden in den Abschnitten 3.5 – 10.5 beschrieben.



[A1.3.3] Screenshot zur Übung mit UML zur Fassade

Zu jedem Entwurfsmuster gibt es ein **UML-Puzzle**, die jeweils einen UML-Editor für Klassendiagramme verwenden. In diesem gibt es Werkzeuge mit denen Klassen erzeugt, Attribute und Methoden benannt und Verbindungen zwischen Klassen (Assoziationen, Vererbungen, Kompositionen und Aggregationen) hergestellt und auch wieder entfernt werden können. Außerdem ist es möglich, Arbeitsschritte rückgängig zu machen und wiederherzustellen. In jedem Modul kann ein Zustand des UML-Puzzles persistent gespeichert und zu einem späteren Zeitpunkt (auch nach einem Neustart des gesamten Programms) wiederhergestellt werden.

Zu jedem Entwurfsmuster gibt es ein unvollständiges Klassendiagramm und eine Aufgabenstellung, in der beschrieben wird, wie das Diagramm an das jeweilige Muster angepasst werden kann. Die Ausgangssituation der Aufgabe lässt sich jederzeit wiederherstellen. Das fertige Klassendiagramm kann überprüft werden, woraufhin das Programm eine Rückmeldung erzeugt, die Hinweise zu Fehlern enthält oder feststellt, dass die Aufgabe gelöst wurde. Abbildung [A1.3.4] enthält einen Screenshot zum UML-Puzzle zur Fassade. In den Abschnitten 3.6-10.6 werden die speziellen Aufgaben zu den einzelnen Entwurfsmustern und die Algorithmen, die die Lösung überprüfen, beschrieben. Folgende Konventionen und Regeln müssen dabei eingehalten werden: Klassennamen werden groß geschrieben, Attribute haben einen Typ, Methoden haben einen Rückgabebetyp und enden mit einer Klammer.

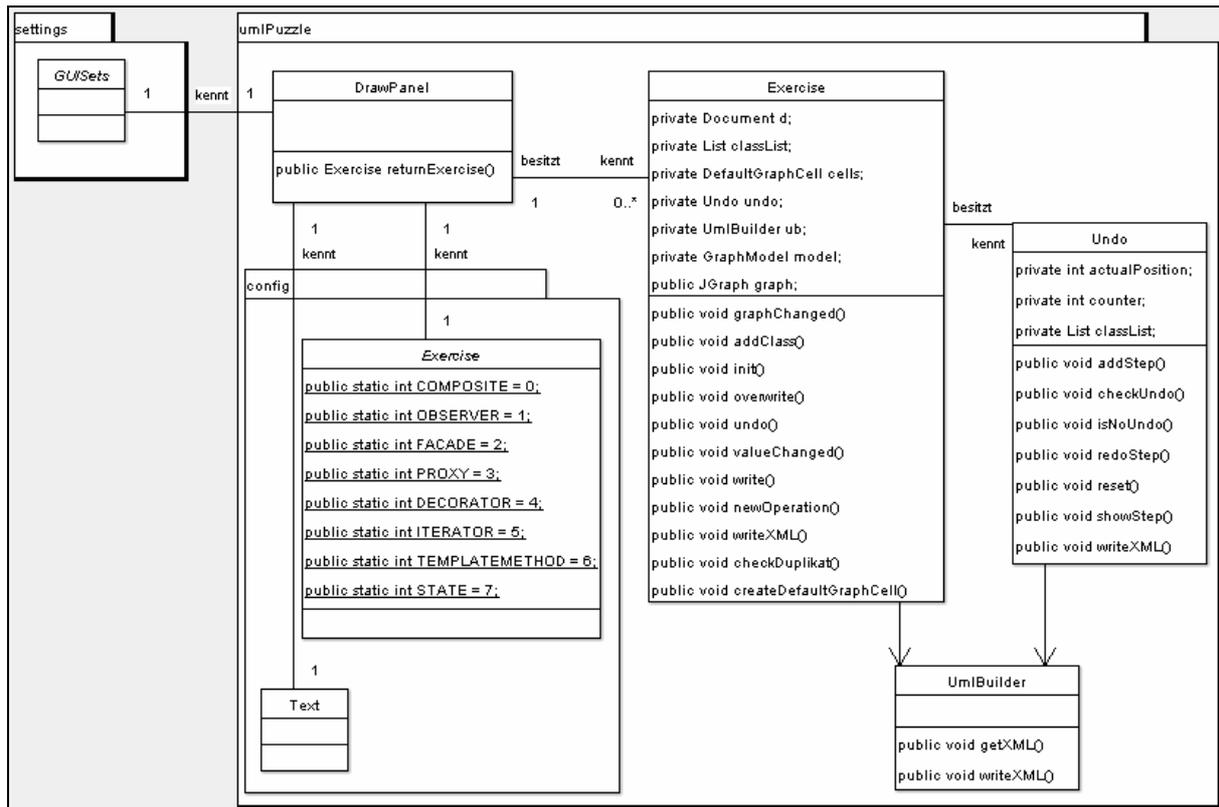


[A1.3.4] Screenshot des UML-Puzzles zur Fassade

Zentrale Klasse des UML-Puzzles ist die Klasse `Exercise`, vgl. Abbildung [A1.3.5]. Durch diese wird mit Hilfe von `JGraph` das Diagramm aufgebaut. Die Methode `graphChanged()` wickelt dabei die Aktionen des Benutzers innerhalb des Graphen ab, `writeXML(String)` schreibt Änderungen in XML und führt dabei gleichzeitig die `init(String)`-Methode aus, die sich um den Aufbau des Graphen kümmert. Dabei ist die Synchronisation der Datenstrukturen wichtig. Neben XML wird für Java die Datenstruktur `Liste` benutzt. Ebenfalls benutzt `JGraph` noch eine eigene Struktur, die synchron gehalten wird. In `evaluateExercise()` werden die Lösungen überprüft.

Das Klassendiagramm wird aus einer zur Übung passenden XML-Datei aufgebaut, die durch einen UML-Builder (Klasse `UMLBuilder`) gelesen und umgewandelt wird. Die Funktion zum Wiederherstellen wird in der Klasse `Undo` realisiert.

In der zentralen GUI-Klasse `DrawPanel` wird die jeweilige Übung durch die Methode `returnExercise()` die aktuelle Übung zurückgegeben, um in anderen Klassen Code zu sparen und damit effizienter zu programmieren.



[A1.3.5] Ausschnitt des Klassendiagramms zum UML-Puzzle

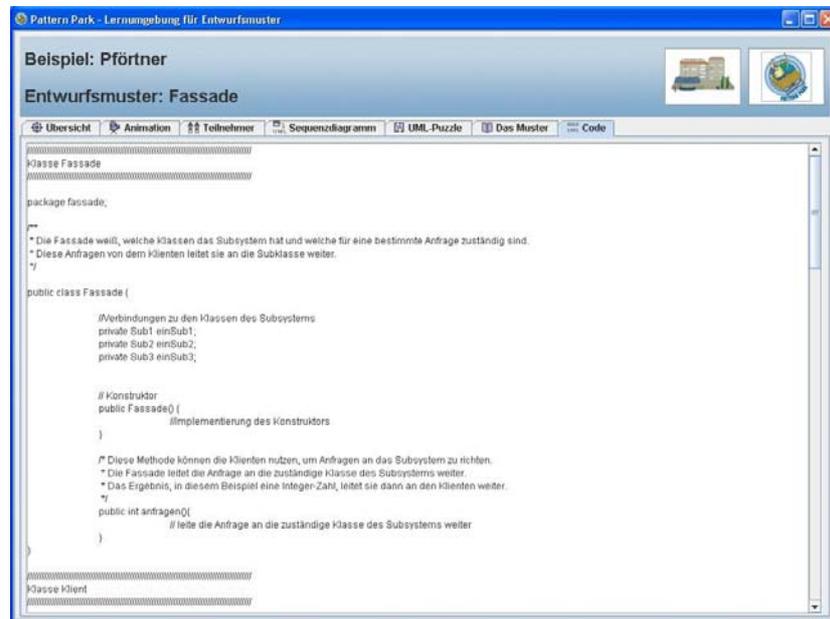
Zu jedem Entwurfsmuster gibt es außerdem eine Seite, in der **das Muster** anhand einer umgangssprachlichen Definition und eines Klassendiagramms allgemeingültig beschrieben wird, vgl. [A1.3.6]. Diese Inhalte befinden sich in diesem Dokument in den Abschnitten 3.2 – 10.2. Sie sind außerdem über das Informationshaus im Glossareintrag zum entsprechenden Muster zu finden.

The screenshot shows the 'Pattern Park' application window. The title bar reads 'Pattern Park - Lernumgebung für Entwurfsmuster'. The main content area is titled 'Beispiel: Pförtner' and 'Entwurfsmuster: Fassade'.

- Navigation:** Includes buttons for 'Übersicht', 'Animation', 'Teilnehmer', 'Sequenzdiagramm', 'UML-Puzzle', 'Das Muster', and 'Code'.
- Diagram:** A UML class diagram showing a **Klient** class with an arrow pointing to a **Fassade** class. The **Fassade** class has three arrows pointing to three sub-system classes: **Sub1**, **Sub2**, and **Sub3**.
- Beschreibung:**
 - Definition: 'Eine Fassade bietet Klienten eine einheitliche Schnittstelle für ein Subsystem an. Das Fassaden-Entwurfsmuster gehört zu der Familie der objektbasierten Strukturmuster.'
 - Vorteile:** 'Das Subsystem ist durch die Fassade einfacher zu nutzen. Die Klienten müssen nämlich keinerlei Kenntnisse über den Aufbau des Subsystems haben, sie können alle ihre Anfragen einfach an die Fassade richten. Subsysteme, die eine Fassade besitzen, sind unabhängig vom übrigen System und dadurch leichter austauschbar und veränderbar.'
 - Nachteile:** 'Eine Fassade macht nur Sinn, wenn das Subsystem eine gewisse Größe hat.'
- Teilnehmer:**
 - Fassade:** 'Die Fassade weiß, welche Klassen das Subsystem hat und welche für eine bestimmte Anfrage zuständig sind. Diese Anfragen von dem Klienten leitet sie an die Subklasse weiter.'
 - Subsystem:** 'In den Klassen des Subsystems ist die Subsystemfunktionalität implementiert. Die kennen die Fassade selbst nicht. In ihnen werden die von der Fassade weitergeleiteten Aufgaben bearbeitet.'
 - Klient:** 'Der Klient schickt Anfragen an das Subsystem über die Fassade.'

[A1.3.6] Screenshot zur allgemeinen Musterbeschreibung zur Fassade

Schließlich wird das allgemeine Klassendiagramm auf einer letzten Seite auch noch als Java-Code dargestellt, allerdings nur die Attribute und Methodenrumpfe, vgl. [A1.3.7].



[A1.3.7] Screenshot zur Code-Seite zur Fassade

1.4 Fachdidaktischer Ansatz dieser Lernsoftware

Diese Lernsoftware folgt dem fachdidaktischen Ansatz zum Informatiksystemverständnis nach Stechert, der diese Projektgruppe betreut hat. Laut Claus und Schwill ist ein System folgendermaßen definiert, vgl. Abschnitt 2.4.

„**System:** In der Informatik versteht man hierunter die Zusammenfassung mehrerer Komponenten zu einer als Ganzes aufzufassenden Einheit. (...)

Systeme lassen sich meist durch mindestens einen der folgenden Punkte charakterisieren:

- durch ein nach außen sichtbares Verhalten. Dies kann man versuchen, durch Tests und Experimente zu ermitteln.
- durch eine innere Struktur. Diese ist in der Regel nur den Entwicklern, nicht aber den Benutzern von Systemen bekannt. Sie lässt sich im Allgemeinen nicht durch Experimente, sondern nur durch eine genaue Analyse der Komponenten ermitteln.
- durch Eigenschaften. Diese erfasst man mithilfe einer Spezifikation, die zur Entwicklung einer konkreten Realisierung dienen kann.“ [Duden06, S.677]

Aus dieser Definition leitet Stechert eine top-down orientierte Dreiteilung des Lernprozesses zum Informatiksystemverständnis ab.

- „ S_A: Understanding of essential aspects of external behaviour of the informatics system,
- S_B: Understanding of essential aspects of the internal structure of the informatics system that are based on fundamental ideas,
- S_C: Understanding of construction details during the development of a concrete realization (implementation details).“ [StSch07, S.3]

Für Basiskompetenzen stehen dabei das nach außen sichtbare Verhalten (S_A) und die innere Struktur der Systeme (S_B) im Vordergrund, während spezielle Eigenschaften wie Implemen-

tierungsdetails (S_C) von geringerer Bedeutung sind, vgl. [Stech07 und Stech06c]. Demzufolge stellen das nach außen sichtbare Verhalten und die lebensweltlichen Beispiele (in den Animationen und den Übungen ohne und mit UML) sowie die innere Struktur eines System repräsentiert durch Entwurfsmuster (in den Übungen mit UML, den UML-Puzzles und den „Das Muster“-Seiten) die Schwerpunkte dieser Lernsoftware dar. Implementierungsdetails werden auf den „Code“-Seiten betrachtet.

Der systemorientierte Ansatz geht davon aus, dass es wichtig ist, diese verschiedenen Aspekte von Informatiksystemen miteinander zu verknüpfen, um Schülern Zusammenhänge zu verdeutlichen, vgl. [StSch07]. In den Animationen und den Übungen mit und ohne UML werden deshalb insbesondere die Beziehungen zwischen den Lebensweltbeispielen und den Entwurfsmustern thematisiert. Des Weiteren geht Stechert davon aus, dass fundamentale Ideen der Informatik und informatische Konzepte in Informatiksystemen miteinander vernetzt sind und dass diese Verbindungen mit Entwurfsmustern als Wissensrepräsentationsformen verdeutlicht werden können, vgl. [Stech06a und Stech06b].

„Object-oriented design patterns support the learning process for informatics system comprehension because they represent experts' knowledge, they are solutions to recurring design problems and carry networked fundamental ideas of informatics.“ [Stech07, S.1]

Deswegen werden in dieser Lernsoftware auch die den Mustern inhärenten informatischen Konzepte und fundamentalen Ideen behandelt, vgl. Abbildung [A1.1.1]. Entwurfsmuster sind außerdem sehr gut geeignet, um die innere Struktur von Software-Systemen (S_B) zu beschreiben. Hierzu bilden sich insbesondere Diagramme der UML an, die deshalb einen weiteren Schwerpunkt dieser Lernsoftware darstellen.

2 Modul Informationshaus

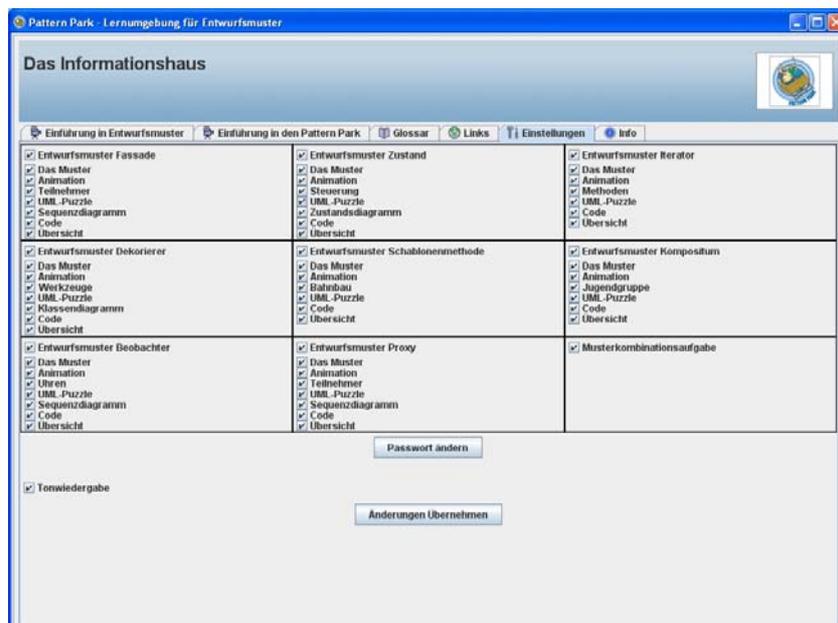
2.1 Funktionalität und Aufgaben des Moduls

Das Modul Informationshaus stellt in der Lernsoftware ein zentrales Konfigurationsobjekt dar. Hier können alle Module aktiviert und deaktiviert werden. Hier befinden sich eine Einführende Animation zum Thema Entwurfsmuster, die einführende Animation zum Pattern Park, die optional auch direkt nach dem Programmstart betrachtet werden kann, ein Glossar mit Einträgen zu allen behandelten Entwurfsmustern und kurzen Erläuterungen zu den in den Übungen behandelten informatischen Konzepten, eine Seite mit Verweisen zu interessanten Internetseiten, sowie eine Seite mit kurzen Informationen zur Entwicklung des Programms. Diese Module werden in den folgenden Abschnitten etwas ausführlicher erläutert.

2.2 Konfigurationsmöglichkeiten des Gesamtprogramms

Die Seite zur Konfiguration des Gesamtprogramms ist über das Modul Informationshaus erreichbar. Sie ist mit einem Passwort geschützt. Dieses lautet standardmäßig „pass“, kann aber über den Dialog geändert werden.

Die Tonausgabe des Programms kann ein- und ausgeschaltet werden. Ist die Tonausgabe deaktiviert, wird bei den Animationen der zugehörige Sprechertext eingeblendet. Einzelne Module oder sämtliche Module zu einzelnen Entwurfsmustern können hier deaktiviert werden. [A2.3.1] zeigt einen Screenshot der Konfigurationsseite.



[A2.3.1] Screenshot der Einstellungen

2.3 Animation Entwurfsmuster

Metadaten

Thema: Die Idee der Wiederverwendung von Christopher Alexander
 Die Übertragung dieser Idee auf die Informatik durch Erich Gamma
 Die Kategorisierung der 23 Entwurfsmuster

Dauer: 2:50 Minuten

Quellen der Bilder: Christopher Alexander:
<http://www.npr.org/templates/story/story.php?storyId=4469331>
 Erich Gamma:
<http://blogs.zdnet.com/Burnette/index.php?p=134>
 Titelbild des Entwurfsmuster-Buches:
<http://www.amiga-magazin.de/magazin/a04-02/oop4a/bild2-gross.jpg>
 Diese Seiten wurden zuletzt aufgerufen am 15.03.2007.



"Gang of Four", um Erich Gamma:

- Übertragung der Idee auf die Informatik
- "Entwurfsmuster - Elemente wiederverwendbarer Software" 23 Entwurfsmuster



		A u f g a b e n		
		Erzeugungsmuster	Strukturmuster	Verhaltensmuster
G E F Ü H R T V O N	K L A S S E N B A S I E R T	Fabrikmethode	Adapter	Interpreter Schablonenmethode
	O B J E K T B A S I E R T	Abstrakte Fabrik Erbauer Prototyp Singleton	Adapter Brücke Dekorierer Fassade Fliegengewicht Kompositum Proxy	Befehl Beobachter Besucher Iterator Memento Strategie Vermittler Zustand Zuständigkeitskette

[A2.3.2] Screenshot der Animation Entwurfsmuster

Sprechertext

„Der Architekt und Mathematiker Christopher Alexander befasste sich in den 1960er und 1970er Jahren mit der Planung von Städten und Gebäuden. Er versuchte schwierige Probleme in Teilprobleme zu unterteilen, um diese dann einfacher lösen zu können. Ihm fiel auf, dass dabei gewisse Probleme immer wieder auftreten. Für diese Probleme hat Alexander elegante Lösungen gesammelt und entwickelt. Diese hat er verallgemeinert, so dass sie als Vorlage immer wieder verwendet werden können. Man bezeichnet diese Lösungen als Entwurfsmuster. Ein Entwurfsmuster ist eine allgemeine und bewährte Lösung für häufig auftretende Probleme. Ein Beispiel: Ein Architekturbüro wird damit beauftragt, eine Wohnsiedlung zu bauen. Die Auftraggeber möchten, dass alle Häuser der Siedlung von außen gleich aussehen und auch alle die gleiche Raumaufteilung haben. Jedes Haus soll allerdings von innen

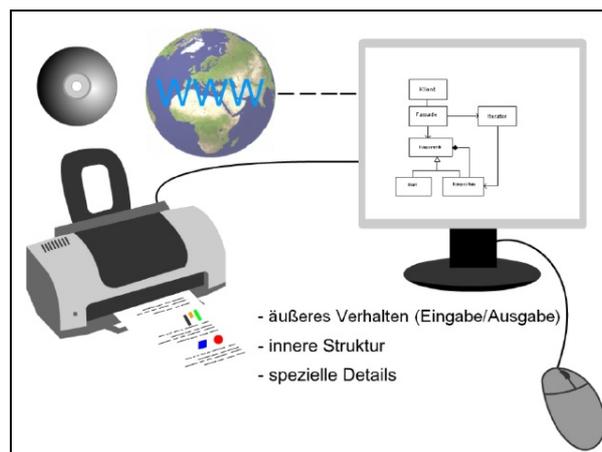
individuell gestaltet werden können. Das Architekturbüro fertigt also ein Entwurfsmuster für ein Haus an, welches dann für alle anderen Häuser ebenfalls verwendet werden kann. Somit muss nicht für jedes Haus ein eigener Entwurf erstellt werden. Diese Idee wurde in den 1990er Jahren von der Gang of Four um Erich Gamma aufgenommen und auf die Softwareentwicklung in der Informatik übertragen. Das Ergebnis ist das Buch Entwurfsmuster – Elemente wiederverwendbarer objektorientierter Software, es beschreibt 23 Entwurfsmuster. Da es so viele Entwurfsmuster gibt, müssen diese irgendwie gruppiert werden. Dieser Gruppierung liegen zwei Kriterien zu Grunde: Das erste Kriterium ist die Aufgabe des Musters: Erzeugungsmuster behandeln die Erzeugung der Objekte. Strukturmuster befassen sich mit der Zusammensetzung von Klassen und Objekten. Verhaltensmuster charakterisieren die Art und Weise, in der Klassen und Objekte zusammenarbeiten und Zuständigkeiten aufteilen. Das zweite Kriterium ist der Gültigkeitsbereich. In diese Tabelle lassen sich die 23 Entwurfsmuster von Gamma einordnen.“

2.4 Animation Pattern Park

Metadaten

Thema: Einführung in den Pattern Park
Die drei Eigenschaften von Informatiksystemen

Dauer: 0:55 Minuten



[A2.4.3] Screenshot der Animation zur Einführung in den Pattern Park

Sprechertext

„Willkommen im Pattern Park! Wir möchten uns einen Freizeitpark mit seinen verschiedenen Einrichtungen, den Mitarbeitern, den Besuchern und den Fahrattraktionen aus Sicht der Informatik anschauen. Hier können wir viele informatische Konzepte wieder finden. Informatik-

systeme bestehen aus Hardware, Software und Netzverbindungen. Um Informatiksysteme zu verstehen, sind drei Eigenschaften zu untersuchen: Das nach außen sichtbare Verhalten mit Eingabe und Ausgabe, die innere Struktur der Systeme und spezielle Details. Wir betrachten Entwurfsmuster, untersuchen die verschiedenen Informatikkonzepte und die Vernetzung der einzelnen Komponenten.“

2.5 Glossar

Im Glossar werden möglicherweise unbekannte Begriffe kurz erklärt. Außerdem enthält der Glossar zu jedem behandelten Entwurfsmuster das entsprechende allgemeine Klassendiagramm mit Erläuterungen. Diese Einträge befinden sich in den Abschnitten 3.2, 4.2, ... bis 10.2.

Das Glossar ist über das Informationshaus zu erreichen, die Beschreibungen der Entwurfsmuster sind auch direkt aus den entsprechenden Modulen über die Seiten „Das Muster“ erreichbar.

Neben den Einträgen zu den Entwurfsmustern befinden sich folgende Einträge im Glossar:

Algorithmus

Unter einem Algorithmus versteht man im Allgemeinen eine eindeutig definierte Handlungsvorschrift zur Lösung einer bestimmten Art von Problemen. Im weiteren Sinne sind also auch Kochrezepte oder Bedienungsanleitungen Algorithmen. In der Informatik bezeichnet man mit einem Algorithmus eine Verarbeitungsvorschrift, die so präzise formuliert ist, dass sie von einem mechanisch oder elektronisch arbeitenden Gerät ausgeführt werden kann. [Duden06, S.39] Beispiele hierfür sind der Euklidische Algorithmus zur Bestimmung des größten gemeinsamen Teilers zweier natürlicher Zahlen oder der Bubblesort Algorithmus zum Sortieren einer endlichen Menge von Elementen nach einem bestimmten Kriterium.

Assoziationen

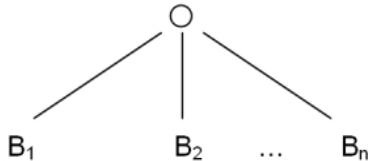
Beziehungen zwischen Klassen werden in der Objektorientierung als Assoziationen bezeichnet. Im Allgemeinen kann man sagen, dass dann eine „kennt“-Beziehung zwischen den entsprechenden Objekten besteht. In Klassendiagrammen werden Assoziationen durch Linien zwischen den Klassen dargestellt. Diese Linien können mit Kardinalitäten beschriftet werden, die angeben, mit wie vielen Objekten ein Objekt in Beziehung stehen kann. Eine Pfeilspitze gibt an, dass es sich um eine gerichtete Assoziation handelt, das heißt, nur die Objekte der einen Klasse kennen die der anderen Klasse, aber nicht umgekehrt. Teil-von-Beziehungen heißen Aggregationen, sie werden durch Linien mit einer Raute an der Klasse, die das Ganze darstellt, gekennzeichnet. Ein Sonderfall der Aggregation ist die Komposition, hierbei ist

die Komponente genau einem anderen Objekt zugeordnet. In der graphischen Darstellung wird die Raute schwarz ausgefüllt. [siehe Kompositum]

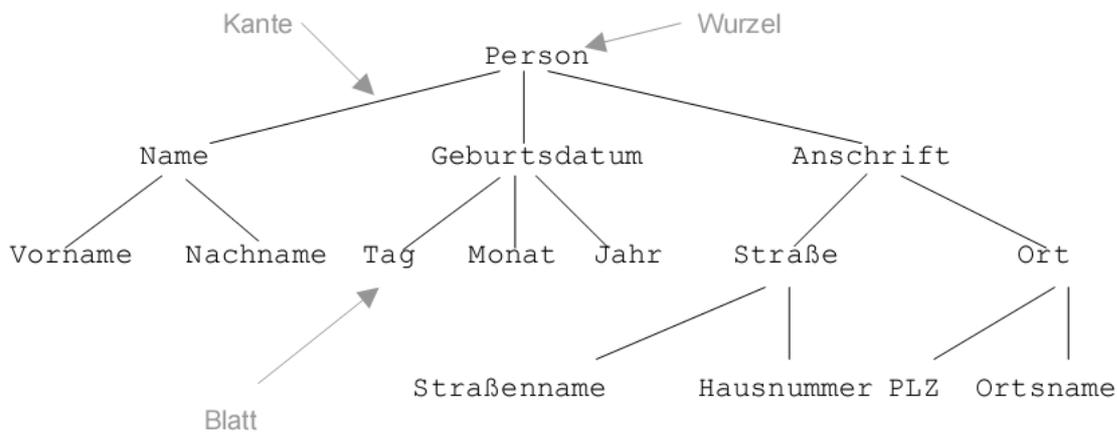
Baum

Unter einem Baum versteht man in der Informatik eine wichtige dynamische Datenstruktur, die man rekursiv folgendermaßen definieren kann:

Ein einzelner Knoten O ist ein Baum. Wenn B_1, B_2, \dots, B_n für eine natürliche Zahl n Bäume sind, dann ist auch



ein Baum. Die Verbindungslinien heißen Kanten. Die Knoten ohne auslaufenden Kanten heißen Blätter. Der Knoten ohne einlaufende Kante heißt Wurzel. Die Knoten und die Kanten können Informationen tragen. Das folgende Beispiel zeigt einen Baum zur Charakterisierung von Personen.



Vgl. [Duden06, S.69ff]

Binärsystem (Dualsystem)

Das Binärsystem ist ein Zahlensystem, das nur zwei Ziffern (0 und 1) zur Darstellung von Zahlen benutzt. Der Stellenwert einer Ziffer steigt von rechts nach links in Potenzen zur Basis 2 an. Beispiel: Die Binärzahl 10110 hat die Dezimaldarstellung

$$1 * 2^4 + 0 * 2^3 + 1 * 2^2 + 1 * 2^1 + 0 * 2^0 = 16 + 0 + 4 + 2 + 0 = 22.$$

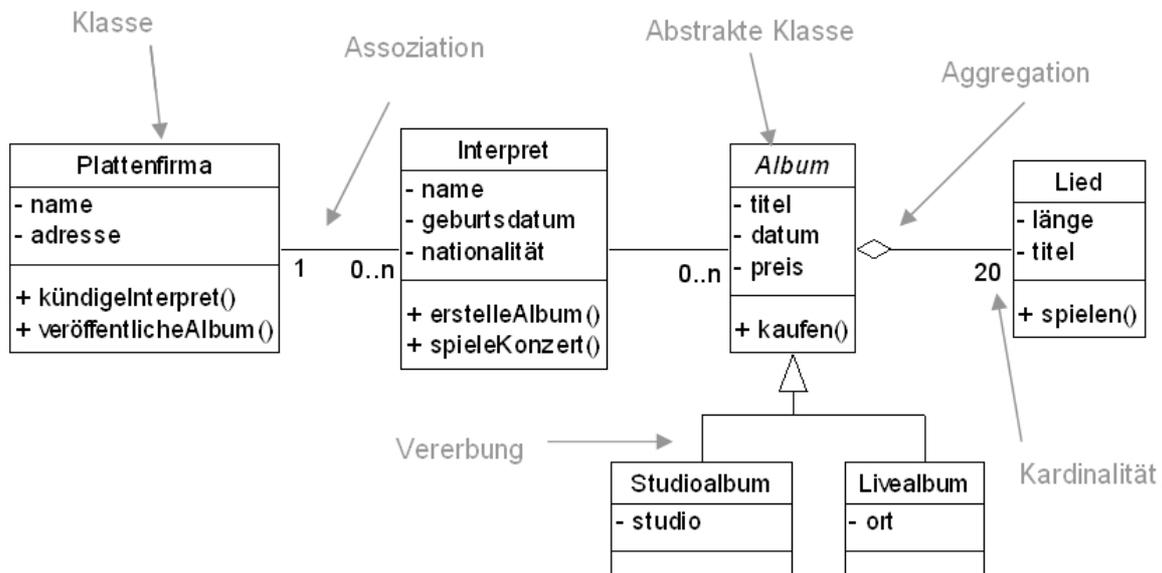
Das Binärsystem hat für die Informatik eine große Bedeutung, da es möglich ist, Zeichen als Binärzahl zu codieren und auf Speicherelementen, die über genau zwei Zustände (Magnet geladen / ungeladen, Spannung hoch / niedrig, ...) verfügen, zu speichern.

Klasse

In der objektorientierten Programmierung ist eine Klasse eine abstrakte Beschreibung einer Menge von Objekten, die die gleiche Struktur besitzen. In einer Klasse werden die Eigenschaften der Objekte (Attribute), ihr Verhalten (Methoden) und ihre Beziehungen zu anderen Klassen [Vererbung und Assoziationen] beschrieben.

Klassendiagramm

Ein Klassendiagramm ist eine der Diagrammart der Unified Modelling Language (UML). Es beschreibt die Eigenschaften (Attribute) und das Verhalten (Methoden) einer Klasse und ihre Beziehungen zu anderen Klassen. Eine Klasse wird dabei als dreigeteiltes Rechteck dargestellt. Oben steht der Klassenname, in der Mitte die Attribute, unten die Methoden, wobei die letzteren beiden jeweils auch weggelassen werden können. Beziehungen [Assoziationen und Vererbungsstrukturen] werden als Linien zwischen den Klassen dargestellt:

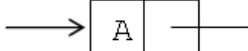


Rekursion (rekursiv)

Als Rekursion bezeichnet man ein Prinzip bei dem ein Verfahren oder eine Definition sich selbst wieder aufruft. Hierbei muss es für mindestens einen Wert ein bekanntes Ergebnis geben (Abbruchbedingung), damit der Algorithmus terminiert (d. h. beendet wird).

Beispiele:

Eine Liste ist leer (Abbruchbedingung). 

Oder eine Liste ist ein Element und eine weitere Liste. 

In der Mathematik ist die Fakultätsfunktion „!“ für alle natürlichen Zahlen n folgendermaßen definiert: 0! = 1 (Abbruchbedingung) n! = n * (n-1)! siehe Baum

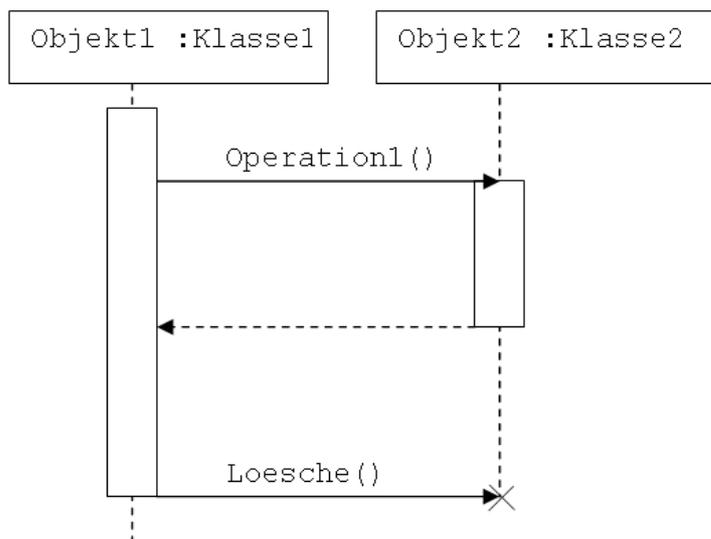
Schnittstelle

Eine Schnittstelle ist eine Verbindungsstelle zum Zweck des Informationsaustausches. Sie fasst alle von außen benötigten und abrufbaren Größen für die Verwendung des Systems zusammen. Sie legt außerdem fest, wie Informationen ausgetauscht werden. Innerhalb eines Informatiksystems gibt es Schnittstellen zwischen den einzelnen Hardwarekomponenten (Hardwareschnittstellen), zwischen den einzelnen Programmen (Softwareschnittstellen) und zwischen Hard- und Software (Hardware-Software-Schnittstellen). Die Benutzungsschnittstelle umfasst die Sprachen, Programme und Geräte die dem Benutzer eines Informatiksystems zur Verfügung stehen. [siehe Fassade]

Vgl. [Duden06, S.599ff]

Sequenzdiagramm

Ein Sequenzdiagramm ist eine der Diagrammarten der Unified Modelling Language [UML]. Es beschreibt die zeitliche Abfolge der Interaktionen zwischen einer Menge von Objekten. Die Zeitachse verläuft dabei von oben nach unten. Ganz oben sind von links nach rechts die Objekte als Rechtecke angeordnet, die in dem Szenario eine Rolle spielen. Von diesen gehen Lebenslinien aus, die gestrichelt gezeichnet sind. Wenn ein Objekt gelöscht wird, so wird dieser Zeitpunkt mit einem großen X gekennzeichnet und die Lebenslinie endet. Zwischen diesen Lebenslinien werden Operationsaufrufe als Pfeil mit durchgezogener Linie eingetragen und mit dem Namen der Operation beschriftet. Der Rückkehrpfeil wird gestrichelt gekennzeichnet. Während ein Objekt eine Operation ausführt ist es aktiv, dies wird durch ein Rechteck auf der Lebenslinie gekennzeichnet. Wenn ein Objekt eine eigene Operation aufruft, zeigt ein Pfeil auf die Lebenslinie des Objekts und das Rechteck wird durch ein weiteres Rechteck überdeckt.



UML

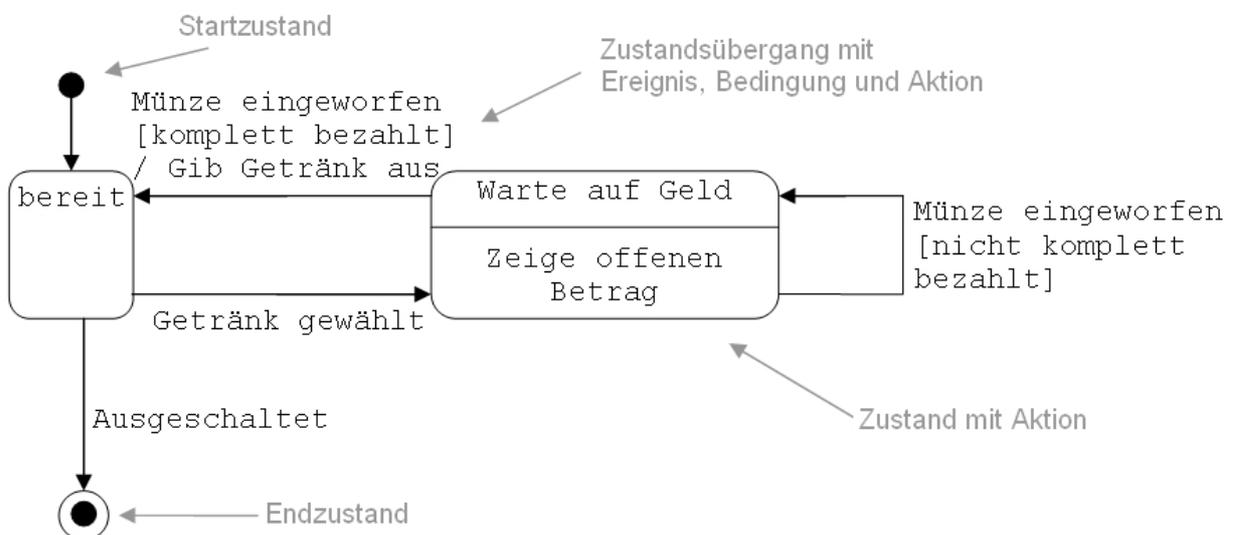
Die Unified Modelling Language (deutsch: vereinheitlichte Modellierungssprache) ist ein Standard zur Modellierung von Software und anderen Systemen. Sie wurde von der Object Management Group (OMG) entwickelt. In der UML werden Begriffe und graphische Notationen festgelegt. [siehe Sequenzdiagramm, Klassendiagramm, Zustandsdiagramm]

Vererbung (Generalisierung)

Vererbung ist ein Konzept in der Objektorientierung, das eine „ist ein“-Beziehung zwischen Klassen beschreibt. Eine Unterklasse erbt dabei von einer Oberklasse, das heißt sie besitzt die gleichen Attribute und Methoden. Diese können aber überschrieben werden (Polymorphie). Im Klassendiagramm geht ein Pfeil mit weißer Pfeilspitze von der Unterklasse hin zur Oberklasse. Solche Oberklassen können abstrakt sein, dann können von ihnen keine Objekte erzeugt werden. Der Klassenname wird dann *kursiv* geschrieben.

Zustandsdiagramm

Ein Zustandsdiagramm ist eine der Diagrammart der Unified Modelling Language [UML]. Es veranschaulicht, was für Zustände ein Objekt einnehmen kann, und wie es seinen Zustand ändern kann. Zustände werden dabei als zweigeteiltes Rechteck mit abgerundeten Ecken abgebildet. Oben steht der Name des Zustands. Unten können Aktionen stehen, die in dem Zustand ausgeführt werden. Diese Aktionen können beim Eintritt in den Zustand (entry), beim Verlassen des Zustands (exit) oder während des Zustands (do) ausgeführt werden. Start- und Endzustand sind wie im Beispiel besonders zu kennzeichnen. Ein Zustandsübergang wird als Pfeil zwischen den Zuständen dargestellt. Er kann mit dem auslösenden Ereignis, einer Bedingung und der auszuführenden Aktion beschriftet werden. Das Beispiel zeigt ein Zustandsdiagramm für einen Getränkeautomaten. [Siehe Zustand]



2.6 Links

Mit Hilfe von Internetverweisen sollen dem Benutzer des Lernangebotes externe Quellen zur Thematik der Entwurfsmuster zur Verfügung gestellt werden. Dadurch kann der Lernprozess des Benutzers unterstützt werden. Das Starten eines Web-Browsers darf hierzu nicht von der Firewall blockiert werden. Diese Seite liegt im HTML-Format vor und kann daher leicht bearbeitet werden. Folgende Verlinkungen befinden sich auf dieser Seite:

Fachgruppe Didaktik der Informatik und E-Learning der Universität Siegen

<http://www.die.informatik.uni-siegen.de>

weitere Lernumgebungen

Homepage der Lernumgebung für objektorientiertes Modellieren im Informatikunterricht

<http://www.die.informatik.uni-siegen.de/pgleo>

Lernumgebung für objektorientiertes Modellieren mit UML

http://www.die.informatik.uni-siegen.de/DIE_BIB/Lehre/ele/sose05/gepra

Seiten über Entwurfsmuster

Codebeispiele für Entwurfsmuster in Java und C++ (englisch)

<http://home.earthlink.net/~huston2/dp/patterns.html>

UML-Diagramme zu Entwurfsmustern (englisch)

<http://www.tml.tkk.fi/~pnr/GoF-models/html/>

Lebensweltliche Beispiele für Entwurfsmuster (englisch)

<http://www2.ing.puc.cl/~jnavon/IIC2142/patexamples.htm>

Übersicht der Entwurfsmuster (englisch)

http://www.mindspring.com/~mgrand/pattern_synopses.htm

Homepage über Softwaremuster (englisch)

<http://hillside.net/patterns/>

Vorlesungsskripte

Vorlesungsskript zu Softwaremustern der Technischen Universität München

<http://www.bruegge.in.tum.de/teaching/ss02/muster/skriptDruckversion.pdf>

Auszug aus dem Vorlesungsskript Software Engineering II der Universität Münster

<http://danae.uni-muenster.de/lehre/kuchen/SS04/SE2k2b.pdf>

Vorlesungsskript zu Entwurfsmustern der Universität Kassel

<http://www.se.eecs.uni-kassel.de/se/fileadmin/se/courses/DesignPattern/DPAZPart1.Version2.pdf>

2.7 Musterkombinationsaufgabe

Über den Wegweiser in der Nähe des Informationshauses gelangt der Benutzer zu einer Übung, die in ihrer Form den UML-Puzzlen aus den Modulen zu den einzelnen Entwurfsmustern ähnelt, jedoch mehrere Muster miteinander verknüpft.

Aufgabenstellung:

„In dieser Übung sollen einige der Entwurfsmuster, die Du im Pattern Park kennen gelernt hast, miteinander kombiniert werden. Dies sind: Kompositum, Iterator, Dekorierer, Fassade. Vielleicht erinnerst Du Dich noch an den Logo-Wettbewerb, der im Einführungsfilm zum Kompositum vorgestellt wurde. Wir verwenden dieses Beispiel auch für diese Aufgabe.“

Kompositum

Eine Grafik kann demnach aus einfachen Elementen wie Linien, Kreisen und Buchstaben oder aus zusammengesetzten Elementen (Logos) bestehen. Vervollständige das Klassendiagramm um diese einfachen Elemente entsprechend dem Kompositum-Entwurfsmuster.

Dekorierer

Mit einem Dekorierer kann man die Funktionalität eines Objekts erweitern. Wir möchten das zusammengesetzte Logo um eine Scrollleiste erweitern. Füge eine Klasse Scrollleiste in das Klassendiagramm ein.

Iterator

Mit dem Iterator-Entwurfsmuster kann man nacheinander auf die einzelnen Elemente eines zusammengesetzten Objekts zugreifen. In unserem Fall benötigen wir einen speziellen Itera-

tor, um auf die einzelnen Elemente Linien, Kreise und Buchstaben des Logos zuzugreifen. Außerdem benötigen wir einen speziellen Iterator, um auf ein zusammengesetztes Objekt zuzugreifen. Es greifen aber beide Iteratoren über die Klasse Logo auf die Elemente zu. Sie haben außerdem beide eine Verbindung zur vorhandenen Klasse Iterator.

Fassade

Du hast außerdem das Entwurfsmuster Fassade kennen gelernt, mit dem man ein Subsystem nach außen abkapseln kann. Wir möchten unser Klassendiagramm nun so abkapseln, dass man nur über eine Klasse LogoSchnittstelle darauf zugreifen kann. Füge diese Klasse und eine Klasse Parkbesucher, die darauf zugreift, in das Klassendiagramm ein.

Abschluss

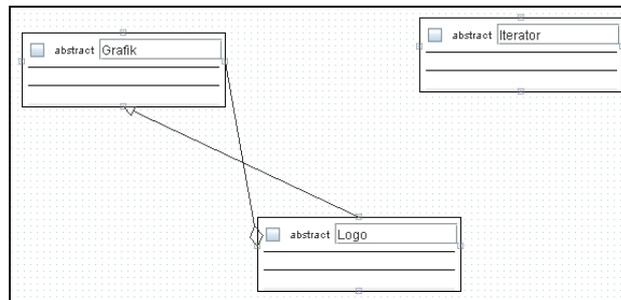
Wir haben jetzt vier Entwurfsmuster (Kompositum, Iterator, Dekorierer und Fassade) in einem Klassendiagramm zusammengefasst. Du kannst versuchen, die Teilnehmer dieser Muster in unserem Klassendiagramm wieder zu finden.

Beim Kompositum: Komponente, Blatt, Kompositum

Beim Iterator: Iterator, konkreter Iterator, konkretes Aggregat, Aggregat

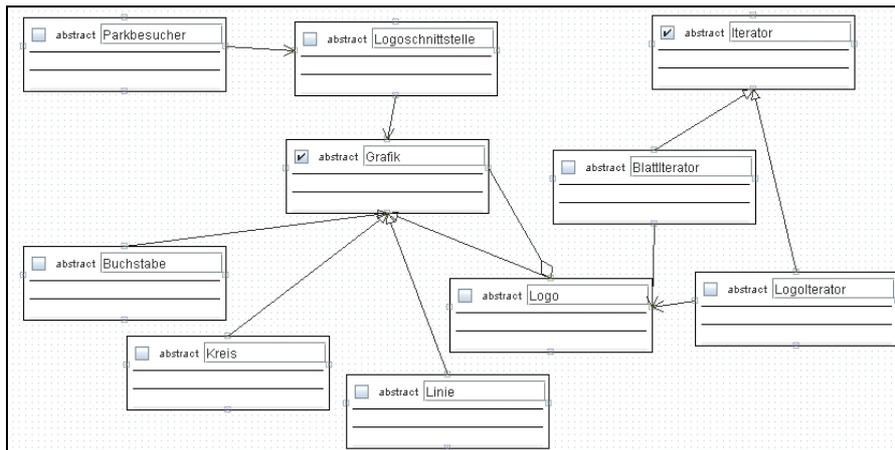
Bei der Fassade: Fassade, Klient“

Abbildung [A2.7.1] zeigt die Ausgangssituation der Musterkombinationsaufgabe.



[A2.7.1] Ausgangssituation der Musterkombinationsaufgabe

Abbildung [A2.7.2] zeigt die Musterlösung zu dieser Aufgabe. Aufgrund des großen Umfangs gibt es hierzu keinen Überprüfungsalgorithmus.



[A2.7.2] Musterlösung zur Musterkombinationsaufgabe

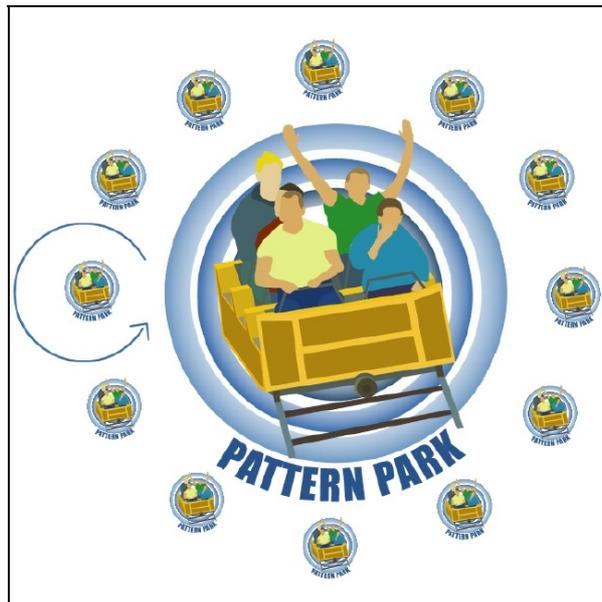
3 Modul Kompositum

3.1 Inhalt der Übersichtsseite

Beispiel:

Logo des Pattern Parks

Logo:



[A3.1.1] Logo Kompositum

Beschreibung:

Eine Grafik kann aus einfachen Objekten – zum Beispiel Linien, Buchstaben oder Kreisen – und aus zusammengesetzten Objekten – zum Beispiel Logos – bestehen. Diese Logos können wiederum aus Buchstaben oder Linien zusammengesetzt sein.

Vorteile:

- Wenn man Teile der Grafik verschieben oder vergrößern will, muss man nicht wissen, ob es sich um ein einfaches Objekt (z.B. eine Linie) handelt, oder ob es ein zusammengesetztes Objekt (z.B. ein Logo) ist.
- Linien, Buchstaben, Kreise usw. können zusammengefügt werden, so dass ein Logo entsteht.

Nachteile:

- Es besteht die Gefahr, dass man den Aufbau eines Logos zu allgemein beschreibt.

3.2 Glossareintrag

Beschreibung:

Mit einem Kompositum kann die Struktur von zusammengesetzten Objekten beschrieben werden, diese können dabei wiederum aus zusammengesetzten Objekten bestehen. Es gehört zu der Familie der objektbasierten Strukturmuster.

Vorteile:

- Klienten müssen nicht wissen, ob es sich um ein einzelnes oder um ein zusammengesetztes Objekt handelt.
- Einfache Objekte können zu umfangreicheren Objekten zusammengefügt werden.

Nachteile:

- Es besteht die Gefahr, dass bei großen Projekten die Struktur von Objekten zu allgemein beschrieben wird.

Kompositum

Die Klasse `Kompositum` definiert das Verhalten von `Komponente` und umfasst Attribute, die auf die Kindobjekte referenzieren. Die Klasse implementiert zusätzlich die kindobjektbezogenen Operationen der Schnittstelle von `Komponente`, vgl. [A3.2.1].

Komponente

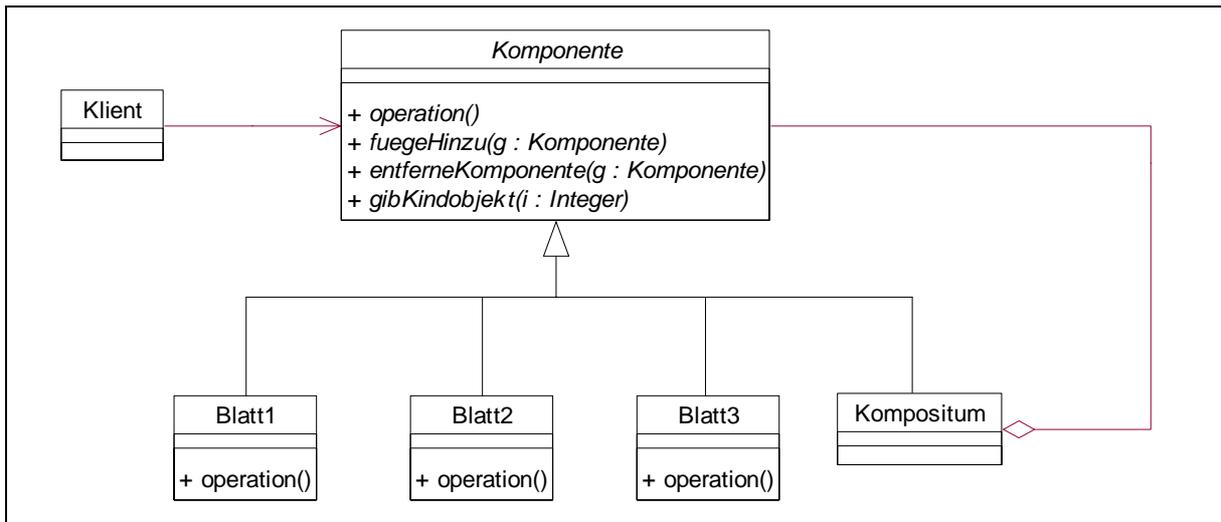
Die Klasse `Komponente` definiert die Schnittstelle für die Objekte in der zusammengesetzten Struktur von `Kompositum`. Hier wird ein mögliches Standardverhalten der Klasse `Blatt` implementiert. Der Zugriff auf die Objekte und deren Verwaltung wird in dieser Klasse geregelt. Optional wird auch der Zugriff auf das Elternobjekt deklariert und ggf. implementiert, vgl. [A3.2.1].

Blatt

Die Klassen `Blatt1`, `Blatt2`, `Blatt3` repräsentieren primitive Objekte in `Komposition`. Ein `Blatt` darf keine Kindobjekte haben. In der Klasse werden die konkreten Eigenschaften und Verhalten der Objekte in `Kompositum` gespeichert, vgl. [A3.2.1].

Klient

Durch `Klient` werden die einzelnen Objekte manipuliert, vgl. [A3.2.1].



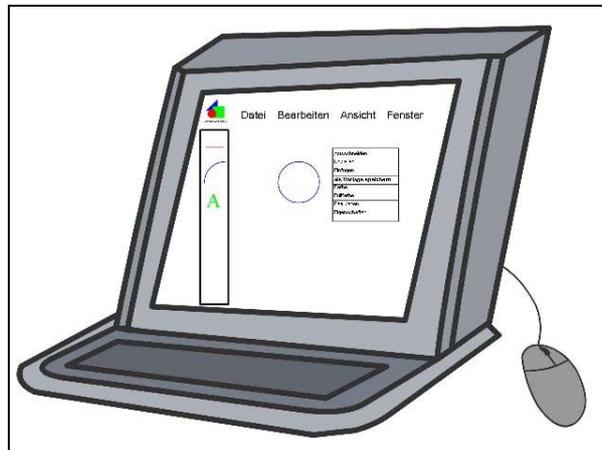
[A3.2.1] UML-Klassendiagramm des Kompositums

3.3 Animation

Metadaten

Thema: Das Logo des Pattern Parks als Beispiel für eine zusammengesetzte Grafik, bestehend aus zusammengesetzten und primitiven Elementen

Dauer: 1:20 Minuten



[A3.3.1] Screenshot der Animation zum Kompositum

Sprechertext

„Im Pattern Park können die Besucher bei einem Wettbewerb mitmachen. Die Besucher haben im Eingangsbereich die Möglichkeit, am PC ein Logo zu entwerfen. In einem Grafikprogramm kann das Logo gezeichnet werden. Folgende Möglichkeiten hat der Besucher in diesem Programm: Er kann einen Strich, einen Kreisbogen oder einen Buchstaben zeichnen. Wenn ihm ein erstelltes Element gut gefällt, kann er dieses als Vorlage zur späteren Ver-

wendung in die Bearbeitungs-Leiste einfügen. Hier ein Beispiel: der Besucher zeichnet vier Kreisbögen. Diese dreht er und schiebt er so zusammen, dass er einen Kreis erhält. Diesen Kreis fügt er nun in die Leiste ein, indem er die Grafik als Vorlage speichert. Jetzt kann er den Kreis noch einmal verwenden, ohne die Teile einzeln zu erstellen. Danach kann der Kreis so behandelt werden, wie ein primitives Objekt, z.B. kann er verschoben werden. Diese Idee findet sich verallgemeinert im Entwurfsmuster Kompositum wieder: Objekte können aus einfachen Objekten zusammengesetzt sein. Einzelne Objekte können dann genauso behandelt werden wie zusammengesetzte Objekte.“

3.4 Übung ohne Einbindung von UML-Diagrammen

Metadaten

- Thema: Die Aufteilung einer Jugendgruppe als Baum
1. Erstelle einen Baum zu einer vorgegebenen Jugendgruppe
 2. Teile die Jugendgruppe entsprechend dem vorgegebenen Baum auf
 3. Teile die Jugendgruppe beliebig auf und erstelle einen entsprechenden Baum
- Dauer: etwa 25 Minuten
- Lernziele: Verbesserung des Verständnisses für die Datenstruktur Baum
- Vorkenntnisse: Grundlagen Datenstruktur Baum

The screenshot displays a software application for visualizing a tree structure. It is divided into several sections:

- Modellsicht - Klassendiagramm:** Shows a class hierarchy where 'Jugendgruppe' is the base class, and 'Jugendlicher' and 'Teilgruppe' are subclasses.
- Aufgabenstellung:** A text box containing instructions in German about describing a youth group's division and providing a specific example scenario.
- Modellsicht - Baum:** A tree diagram representing the 'Jugendgruppe Mustardorf'. The root node has three children: '2', 'PONDORFER', and 'Riesernd'. '2' has children 'Max' and 'Tim'. 'PONDORFER' has children 'Lisa' and 'Ralf'. 'Riesernd' has children 'Lena', 'Tom', and 'Jule'. Further sub-nodes like 'Tina' and 'Mark' are also visible.
- Realsicht:** A colorful scene with various elements: a boat with 'Tina', 'Tim', and 'Max'; a 'Bergbahn' (cable car) with 'Anna'; a 'POWER TOWER' with 'Lisa', 'Tina', and 'Ralf'; and a curved path with numbered nodes (1-6) and characters 'Jule' and 'Mark'.
- Control Panel:** Located on the right, it includes 'Modus' options (Real > Baum, Baum > Real, eigene Eingabe), buttons for 'Hilfen', 'Aufgabe', and 'Liste anzeigen'.

[A3.4.1] Screenshot der Übung ohne UML zum Kompositum

In diesem Modul geht es darum, die rekursive Objektstruktur des Entwurfsmusters Kompositum mit Hilfe der Datenstruktur Baum zu verstehen. Der allgemeine Aufbau eines Baums wird im entsprechenden Glossareintrag kurz erläutert. Das Beispiel einer Jugendgruppe, die den Freizeitpark besucht und sich dort in Kleingruppen und Einzelpersonen aufteilt, soll dem Benutzer zu Beginn der Übung exemplarisch verdeutlichen, wie man aus einem gegebenen Klassendiagramm und einer Beschreibung der Gruppenstruktur einen entsprechenden Baum ableiten kann, vgl. [A3.4.1].

Die Realsicht ist derartig zu verstehen, dass eine Jugendgruppe von 10 Personen den Park besucht und sich in Teilgruppen aufgliedert. Beispielsweise bilden Lena, Tom, Mark und Jule eine Teilgruppe, da sie alle das Riesenrad besuchen. Teilgruppen können aus weiteren Teilgruppen – zum Beispiel bilden Mark und Jule eine solche in Gondel 5 – oder aus einzelnen Jugendlichen – in diesem Beispiel Lena und Tom – bestehen. Es sind keine Schnittmengen zwischen den Teilgruppen erlaubt, ein Jugendlicher sitzt zu einem Zeitpunkt in genau einem Wagen. Entsprechend handelt es sich im Klassendiagramm um eine Komposition und nicht um eine Aggregation. Dies ist also ein Sonderfall des Entwurfsmusters Kompositum.

Die dargestellte Realsicht ist äquivalent zu einer Baumstruktur. Dabei entspricht eine einzelne Person einem Blatt und eine Gruppe von Personen (innerhalb einer Fahrattraktion oder eines Wagens) einem Teilbaum. Die maximale Rekursionstiefe ist hierbei 3.

Sofern ein Jugendlicher alleine eine Teilgruppe bildet, soll der Knoten des Baums mit dem Namen des Jugendlichen beschriftet werden. Wenn zwei Jugendliche eine Teilgruppe innerhalb eines Wagens bilden, soll der Knoten mit der Nummer des Wagens (die eindeutig einer Fahrattraktion zuzuordnen ist) und die Knoten darunter mit den Namen der Jugendlichen beschriftet werden. Wenn sich mehrere Jugendliche auf mehrere Wagen einer Fahrattraktion verteilen, soll der obere Knoten mit dem Namen der Attraktion, die Knoten darunter mit der Nummer der Wagen und den Namen der Jugendlichen beschriftet werden.

Im Beispiel ist der erste Teilbaum mit Wagennummer 2 beschriftet, nicht etwa mit „Thunderbird“. Der dritte Teilbaum ist mit „Anne“ beschriftet, nicht mit „Bergbahn“ oder der Wagennummer 3 usw.

Lernziel dieses Moduls ist es, ein Verständnis für den prinzipiellen rekursiven Aufbau der Gruppenstruktur und des Baums zu entwickeln und die Äquivalenz zwischen diesen Darstellungsformen zu erkennen. Dabei wird auf die fundamentalen Ideen Rekursion, Komposition und Baumstruktur, die in dem Entwurfsmuster enthalten sind, eingegangen. Der Benutzer soll lernen, aus der Abbildung, die den Freizeitpark und die sich darin aufhaltenden Personen beschreibt, den entsprechende Baum abzuleiten und umgekehrt. Die Inhalte der beiden Sichten sollen gleichgesetzt werden.

Es gibt drei Bearbeitungsmodi:

Real → Baum

Per Zufallsgenerator werden die zehn Jugendlichen auf die Plätze der Fahrattraktionen verteilt, diese Verteilung wird in der Realsicht angezeigt. Die Aufgabe besteht darin, den entsprechenden Baum in der Modellsicht zu erstellen und die Knoten zu beschriften. Es wird eine Musterlösung angezeigt.

Baum → Real

Per Zufallsgenerator werden die zehn Jugendlichen auf die Plätze der Fahrattraktionen verteilt, diese Verteilung wird in der Modellsicht als Baum angezeigt. Die Aufgabe besteht darin, die Jugendlichen in der Realsicht so auf die Plätze in den Fahrattraktionen zu verteilen, dass die Verteilung dem vorgegebenen Baum entspricht.

Es ist zu beachten, dass es zu einem Baum in der Regel mehrere richtige Lösungen gibt: Wenn ein Jugendlicher zum Beispiel einen Teilbaum bildet, legt dies fest, dass er alleine eine Fahrattraktion besucht, nicht aber, welchen Platz er in welchem Wagen einnimmt.

Erst wenn alle Jugendlichen einen Platz haben, kann die Verteilung überprüft werden. Falsche Einträge werden dann rot markiert und können korrigiert werden.

Eigene Eingabe

In diesem Modus müssen die Jugendlichen zuerst in der Realsicht auf die Fahrattraktionen verteilt werden. Anschließend muss – wie im Modus Real → Baum – ein entsprechender Baum erstellt werden. Die Jugendlichen können also so verteilt werden, dass ein bestimmter Baum entsteht, zum Beispiel einer mit Tiefe=2 oder einer mit zwei Teilbäumen.

3.5 Übung mit Einbindung von UML-Diagrammen

Im Modul Kompositum gibt es keine Übung mit UML zusätzlich zum UML-Puzzle.

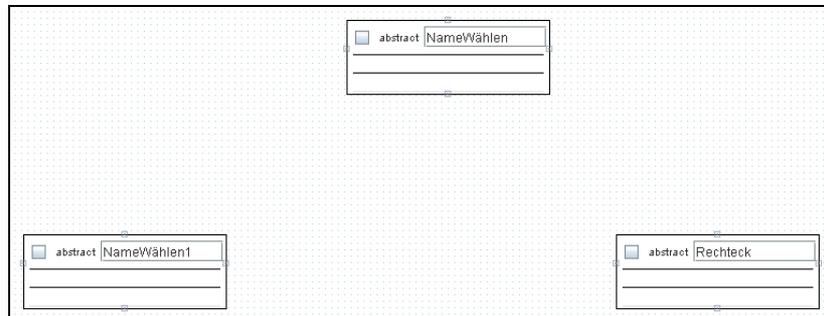
3.6 UML-Puzzle

Aufgabenstellung:

„Du konntest in der Animation zum Entwurfsmuster Kompositum das Beispiel zu einer Grafik sehen. Dieses Beispiel kannst du dir hier genauer anschauen. Deine Aufgabe ist, das Klassendiagramm zum Kompositum zu vervollständigen. Benenne zuerst die Klassen sinnvoll, nutze dabei drei der folgenden Vorgaben: Logo, Park, Grafik, Linie, Schild. Füge dann folgende Methoden sinnvoll den einzelnen Klassen hinzu: void zeichne(), void fuegeHinzu(Grafik), void entferne(Grafik), Grafik gibKindobjekt(int). Vervollständige anschließend das

UML-Diagramm mit Beziehungen zwischen den einzelnen Klassen. Kennzeichne zuletzt die abstrakte Klasse.“

Abbildung [A3.6.1] zeigt die Ausgangssituation des UML-Puzzles zum Kompositum.

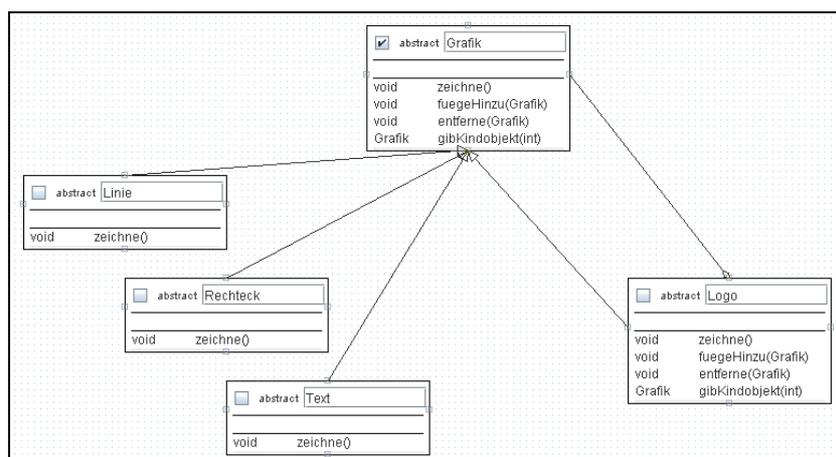


[A3.6.1] Ausgangssituation des UML-Puzzles zum Kompositum

Beschreibung des Algorithmus zur Überprüfung der Lösung

Es wird überprüft, ob die Klassen Grafik, Logo, Linie, Rechteck und Text vorhanden sind. Fehlt eine der Klassen, bzw. ist noch nicht umbenannt worden, wird ein entsprechender Fehlertext bei der Überprüfung ausgegeben. Ebenso werden die Vererbungen und die Aggregation zwischen Grafik und Logo überprüft. Die Methoden `zeichne()` (in allen Klassen), `fuegeHinzu(Grafik)`, `entferne(Grafik)` und `gibKindobjekt(int)` (in den Klassen Grafik und Logo) werden auf Existenz überprüft. Erst wenn alle Kriterien erfüllt wurden, wird die Aufgabe als gelöst gekennzeichnet. Alles was darüber hinaus noch hinzugefügt wird, wird bei der Überprüfung nicht berücksichtigt.

Abbildung [A3.6.2] zeigt die Musterlösung zum UML-Puzzle Kompositum.



[A3.6.2] Musterlösung zum UML-Puzzle zum Kompositum

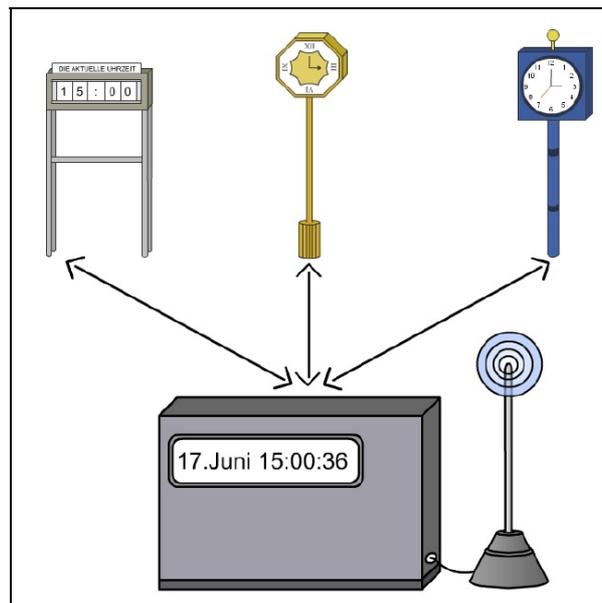
4 Modul Beobachter

4.1 Inhalt der Übersichtsseite

Beispiel:

Eine Uhrzeit – Mehrere Uhren

Logo:



[A4.1.1] Logo Beobachter

Beschreibung:

Mehrere Uhren werden benachrichtigt und aktualisiert, wenn sich die Uhrzeit ändert.

Vorteile:

- An vielen Stellen im Park ist die aktuelle Uhrzeit zu sehen.
- Das zentrale Uhrwerk und die vielen Uhren können unabhängig voneinander verändert werden.
- Das zentrale Uhrwerk muss nicht genau wissen, wie die vielen Uhren im Einzelnen funktionieren. Sie werden einfach nur benachrichtigt.

Nachteile:

- Es ist sehr aufwändig, ständig alle Uhren im Park zu benachrichtigen.
- Möglicherweise werden Uhren benachrichtigt, die gar nicht aktualisiert werden müssen, zum Beispiel weil sie keinen Sekundenzeiger haben.

4.2 Glossareintrag

Beschreibung:

Mehrere Beobachter werden benachrichtigt und aktualisiert, wenn das beobachtete Subjekt seinen Zustand ändert. Es gehört zu der Familie der objektbasierten Verhaltensmuster.

Vorteile:

- Wenn sich an einer Stelle im System etwas ändert, kann dies an vielen Stellen sichtbar werden.
- Das Subjekt und die Beobachter sind lose gekoppelt. Sie können also unabhängig voneinander variiert werden.
- Das Subjekt muss keine detaillierten Kenntnisse über seine Beobachter haben. Wenn das Subjekt seinen Zustand ändert, benachrichtigt es einfach nur eine Schnittstelle.

Nachteile:

- Die Änderung des Subjekts kann viele Benachrichtigungen verursachen, wenn es viele Beobachter gibt.
- Möglicherweise werden Beobachter benachrichtigt, die gar nicht aktualisiert werden müssen.

Beobachter

Die Klasse `Beobachter` definiert eine Aktualisierungsschnittstelle für Objekte, die über Änderungen von `Subjekt` benachrichtigt werden wollen, vgl. [A4.2.1].

KonkreterBeobachter

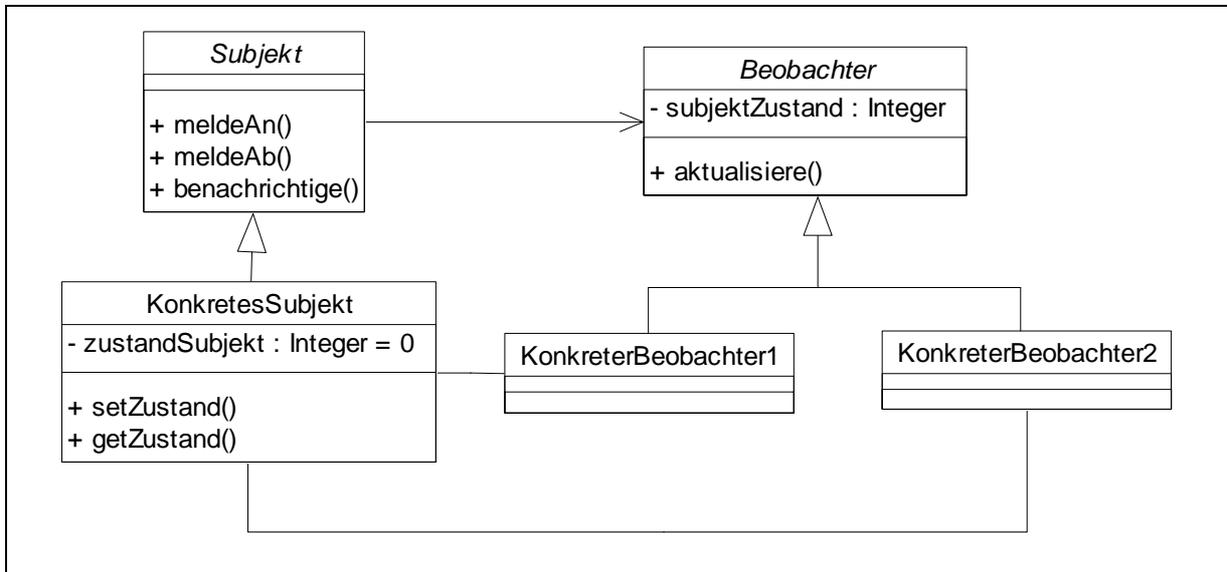
Die Klasse `KonkreterBeobachter` verwaltet eine Referenz auf `KonkretesSubjekt`. Hier wird der Zustand gespeichert, der zum passenden `Subjekt` gehört. Die Aktualisierungsschnittstelle der Klasse `Beobachter` wird hier implementiert, damit der Zustand mit dem von `Subjekt` konsistent bleibt, vgl. [A4.2.1].

Subjekt

Das `Subjekt` kennt alle `Beobachter`. Eine beliebige Anzahl von `Beobachter` kann ein `Subjekt` beobachten, vgl. [A4.2.1].

KonkretesSubjekt

Die Klasse `KonkretesSubjekt` benachrichtigt seine `Beobachter` bei Änderungen des Zustands (d.h. seiner Attribute und Daten). In dieser Klasse wird der Zustand für die Klasse `KonkreterBeobachter` gespeichert, vgl. [A4.2.1].



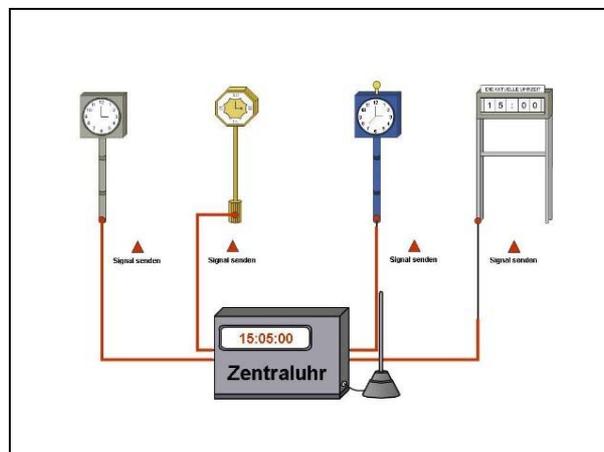
[A4.2.1] UML-Klassendiagramm des Beobachters

4.3 Animation

Metadaten

Thema: Die Kommunikation zwischen der Zentraluhr und verschiedenen Uhren im Pattern Park als Beispiel für eine Broadcast-Kommunikation

Dauer: 1:40 Minuten



[A4.3.1] Screenshot der Animation zum Beobachter

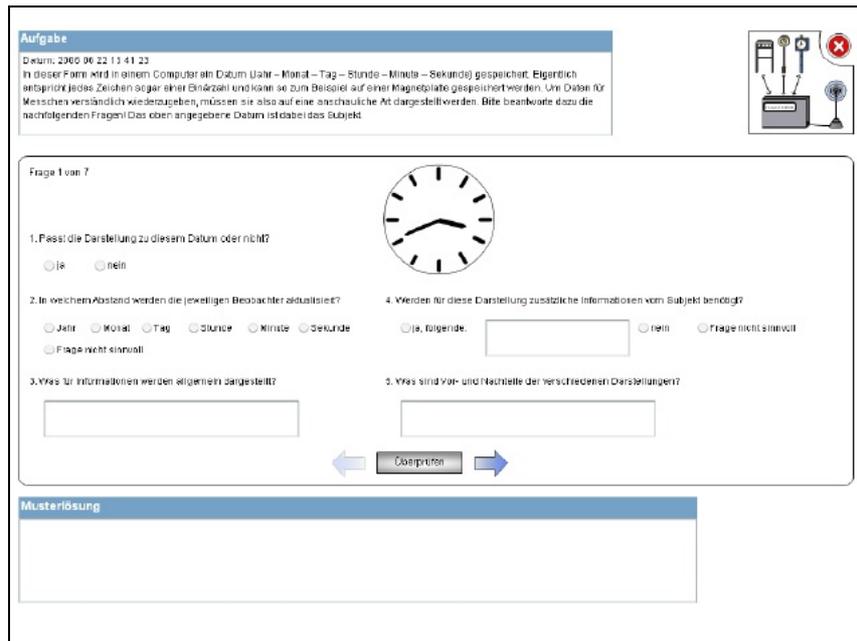
Sprechertext

„Im Pattern Park sind verschiedene Arten von Uhren aufgestellt: Analoguhren mit und ohne Sekundenanzeige sowie Digitaluhren. Im Verwaltungsbereich des Parks gibt es ein zentrales Uhrwerk. Damit nicht alle Uhren nachgestellt werden müssen, werden diese durch Leitungen mit dem zentralen Uhrwerk verbunden. Sie melden sich bei diesem an und teilen ihm mit, dass sie benachrichtigt werden wollen, wenn sich die Uhrzeit ändert. Sobald sich nun im zentralen Uhrwerk die Uhrzeit, also der Zustand ändert, werden durch die Leitungen Signale an die Uhren im Park versendet. Dies nennt man Broadcast-Kommunikation, bei der ein Sender, welchen wir im folgenden Subjekt nennen, an eine unbestimmte Anzahl von Empfängern Nachrichten versendet. Diese nennen wir Beobachter, da sie das zentrale Uhrwerk abhören, ob dieses ein Signal sendet. Alle Uhren im Park gehen somit gleich. Eine der Uhren muss gewartet werden. Sie reagiert nicht auf die Synchronisationsnachricht des zentralen Uhrwerks. Um den Kommunikationsaufwand möglichst gering zu halten, meldet der Techniker die Uhr am zentralen Uhrwerk ab. Solange sie abgemeldet ist, empfängt sie keine Nachrichten mehr.“

4.4 Übung ohne Einbindung von UML-Diagrammen

Metadaten

Thema:	Verschiedene Darstellungsmöglichkeiten eines Datums
Dauer:	etwa 10 Minuten
Lernziele:	Vor- und Nachteile verschiedener Darstellungsmöglichkeiten eines Datums
Vorkenntnisse:	keine



[A4.4.1] Screenshot der Übung ohne UML zum Beobachter

Mithilfe des Beobachters kann ein Datensatz auf verschiedene Arten repräsentiert werden. In dieser einfachen Übung geht es darum, verschiedene Darstellungsformen einem Datum zuzuordnen und sich über Vor- und Nachteile der unterschiedlichen Darstellungsformen Gedanken zu machen. Gegeben sind ein Datum und sieben Darstellungsmöglichkeiten, zu denen jeweils fünf Fragen beantwortet werden müssen.

Der Benutzer soll dabei lernen, dass Daten, die in einem Informatiksystem gespeichert sind, auf eine für Menschen verständliche Art dargestellt werden müssen. Unterschiedliche Darstellungsformen haben Vor- und Nachteile, sind für unterschiedliche Zwecke geeignet und können unterschiedlich genau sein.

Das vorgegebene Datum beinhaltet das Jahr, den Monat, den Tag und die sekundengenaue Uhrzeit.

Zu sieben verschiedenen Abbildungen, die teilweise dem vorgegebenen Datum entsprechen, sollen jeweils die folgenden Fragen beantwortet werden:

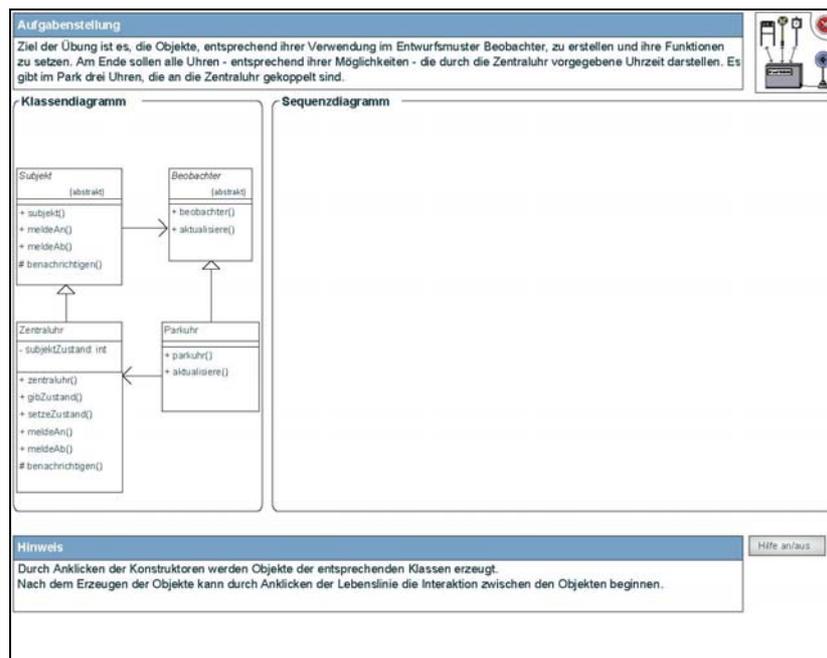
- Frage 1: Passt die Darstellung zu diesem Datum oder nicht?
Ja / Nein Ankreuzen
- Frage 2: In welchem Abstand werden die jeweiligen Beobachter aktualisiert?
1 aus n (Jahr, Monat, Tag, Stunde, Minute, Sekunde)
- Frage 3: Was für Informationen werden allgemein dargestellt?
Freitexteingabe
- Frage 4: Werden für diese Darstellung zusätzliche Informationen vom Subjekt benötigt?
Ja (welche?) / Nein
- Frage 5: Was sind Vor- und Nachteile der verschiedenen Darstellungen?
Freitexteingabe

Wenn alle Fragen beantwortet wurden, können die Eingaben überprüft werden und die Übung kann mit der nächsten Darstellung fortgesetzt werden. Abbildung [A4.4.1] zeigt einen Screenshot.

4.5 Übung mit Einbindung von UML-Diagrammen

- Thema:** Die Kommunikation zwischen der Zentraluhr und den verschiedenen Uhren im Park als Sequenzdiagramm
1. Melde alle Uhren an der Zentraluhr an
 2. Aktualisiere alle Uhren
- Dauer:** etwa 15 Minuten
- Lernziele:** Verbesserung des Verständnisses für die Funktionsweise des Entwurfsmusters Beobachter (Broadcast-Kommunikation)
- Vorkenntnisse:** Grundlagen Sequenzdiagramm
Verständnis des Klassendiagramms Beobachter

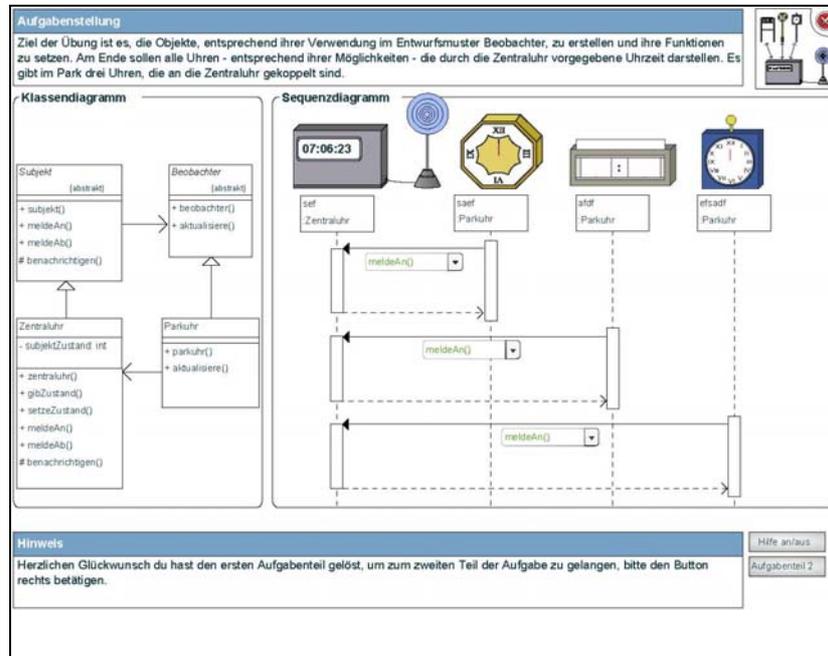
Für das Modul zum Beobachter-Entwurfsmuster ist eine Übung unter Einbindung eines Sequenzdiagramms implementiert worden. Dem Benutzer wird zu Beginn der Aufgabe die Aufgabenbeschreibung dargestellt und ein Klassendiagramm angezeigt, um ihm die Funktionalitäten der einzelnen Klassen anzuzeigen, vgl. [A4.5.1].



[A4.5.1] Startbild der Übung mit UML zum Beobachter

Durch Anklicken der Konstruktoren im Klassendiagramm werden die entsprechenden Objekte im Sequenzdiagramm erzeugt, die dann durch Auswählen des Benutzers untereinander kommunizieren.

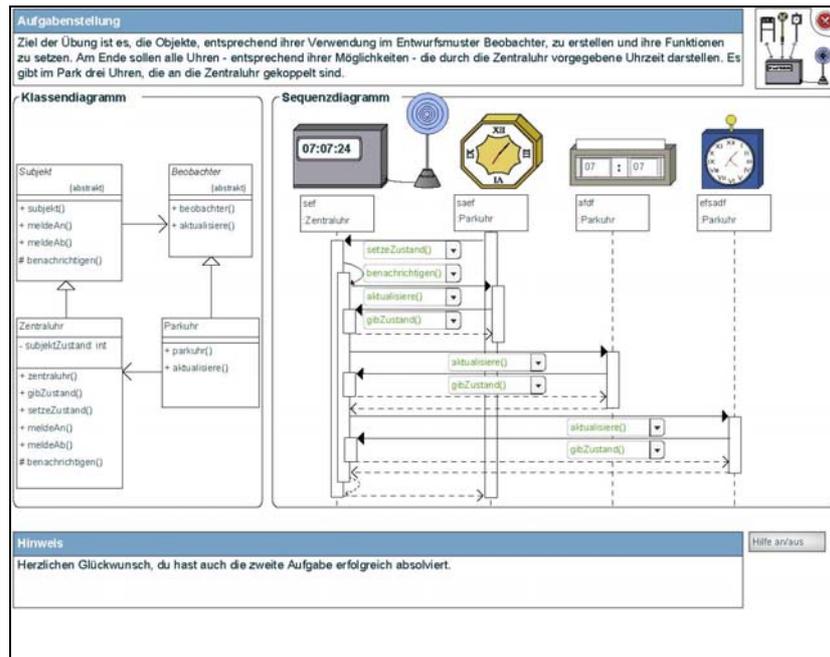
Dabei werden die in der Animation eingeführten Rollen Subjekt und Beobachter aufgegriffen und in dem Sequenzdiagramm dargestellt, vgl. [A4.5.2].



[A4.5.2] Anmelden der Beobachter an der Zentraluhr in der Übung mit UML zum Beobachter

Der Benutzer muss nun aus einer Auswahl von Methodenaufrufen auswählen, um das Subjekt, in der Übung handelt es sich dabei um die Zentraluhr des Pattern Parks, mit konkreten Beobachtern (Uhren die im Park verteilt stehen) verbinden. Beobachter melden sich beim Subjekt jeweils durch die Methode `meldeAn()` an oder durch `meldeAb()` wieder ab, vgl. [A4.5.2].

Das Subjekt, die Zentraluhr, wird von einer Parkuhr, durch die Methode `setzeZustand()` dazu aufgefordert, seinen aktuellen Stand preiszugeben. Daraufhin ruft die Zentraluhr ihre eigene Methode `benachrichtigen()` auf. Die Zentraluhr fordert dann, durch Aufrufen der Methode `aktualisiere()`, den Beobachter auf, seinen Status zu erneuern. Im Folgenden muss der Beobachter den aktuellen Zustand bei der Zentraluhr abholen, dies wird durch die Methoden `gibZustand()` realisiert. Diese übermittelt immer Stunden, Minuten und Sekunden, obwohl manche Beobachter diesen Zustandswechsel nicht explizit darstellen können (Uhren ohne Sekundenanzeige), vgl. [A4.5.3]. Das jeweilige Uhrenobjekt ändert seine Anzeige, je nachdem, in welchem Intervall (Sekunden, Minuten etc.) es eine Aktualisierung ausführen kann.



[A4.5.3] Aktualisierung der Uhrzeit in der Übung mit UML zum Beobachter

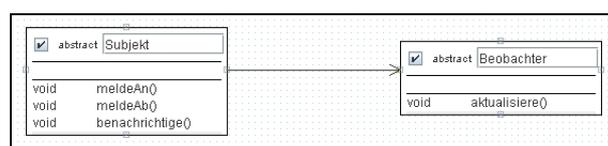
Der Benutzer kann nun im Verlauf der Übung verschiedene Beobachter mit dem Subjekt interagieren lassen.

4.6 UML-Puzzle

Aufgabenstellung:

„Als Beispiel für das Beobachtermuster hast du die verschiedenen Uhren im Park und die Zentraluhr kennen gelernt. Füge dem Klassendiagramm die Klassen `ZentraleParkUhrzeit` und `FreizeitParkUhr` hinzu. Die Klasse `FreizeitParkUhr` kann zusätzlich noch durch konkrete Uhren, wie z.B. `AnalogUhr` oder `DigitalUhr`, erweitert werden. Füge darüber hinaus noch die Attribute `uhrzeit` (vom Typ `Date`) und `standort` (vom Typ `String`) dort ein, wo sie dir sinnvoll erscheinen. Trage außerdem Beziehungen zwischen den Klassen ein.“

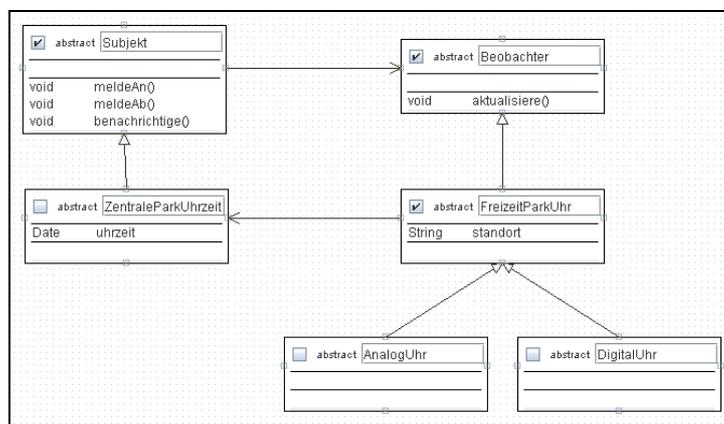
Abbildung [A4.6.1] zeigt die Ausgangssituation des UML-Puzzles zum Beobachter.



[A4.6.1] Ausgangssituation des UML-Puzzles zum Beobachter

Beschreibung des Algorithmus zur Überprüfung der Lösung

Es wird überprüft, ob die Klassen `ZentralParkUhrzeit`, `FreizeitParkUhr`, `AnalogUhr` und `DigitalUhr` vorhanden sind. Fehlt eine von den Klassen wird ein entsprechender Fehlertext bei der Überprüfung ausgegeben. Die Klasse `FreizeitParkUhr` wird ebenfalls überprüft, ob sie `abstract` gesetzt wurde. Ebenso werden alle Vererbungen und die Assoziation zwischen `FreizeitParkUhr` und `ZentralParkUhrzeit` überprüft. Die Attribute `uhrzeit` (in der Klassen `ZentraleParkUhrzeit`) und `standort` (in der Klasse `FreizeitParkUhr`) werden auf Existenz überprüft. Erst wenn alle Kriterien erfüllt werden, wird die Aufgabe als gelöst gekennzeichnet. Abbildung [A4.6.2] zeigt die Musterlösung zum UML-Puzzle zum Beobachter.



[A4.6.2] Musterlösung zum UML-Puzzle zum Beobachter

5 Modul Fassade

5.1 Inhalt der Übersichtsseite

Beispiel:

Pförtner

Logo:



[A5.1.1] Logo Fassade

Beschreibung:

Ein Pförtner stellt für die Besucher des Parks einen einheitlichen Ansprechpartner für den gesamten Verwaltungsbereich dar.

Vorteile:

- Das erleichtert es den Besuchern, Fragen zu stellen. Sie müssen nämlich nicht wissen, wie der Verwaltungsbereich organisiert ist, sondern fragen einfach immer den Pförtner.
- Im Verwaltungsbereich kann ein Mitarbeiter leicht durch einen anderen ersetzt werden, ohne dass die Besucher dies bemerken.

Nachteile:

- Ein Pförtner macht nur dann Sinn, wenn der Verwaltungsbereich eine gewisse Größe hat. Ansonsten ist es zu umständlich, immer erst den Pförtner fragen zu müssen.

5.2 Glossareintrag

Beschreibung:

Eine Fassade bietet Klienten eine einheitliche Schnittstelle für ein Subsystem an. Es gehört zu der Familie der objektbasierten Strukturmuster.

Vorteile:

- Das Subsystem ist durch die Fassade einfacher zu nutzen. Die Klienten müssen nämlich keinerlei Kenntnisse über den Aufbau des Subsystems haben, sie können alle ihre Anfragen einfach an die Fassade richten.
- Subsysteme, die eine Fassade besitzen, sind leichter austauschbar und veränderbar. Sie haben nämlich keine Verbindungen nach außen und sind dadurch unabhängig.

Nachteile:

- Eine Fassade macht nur Sinn, wenn das Subsystem eine gewisse Größe hat.

Fassade

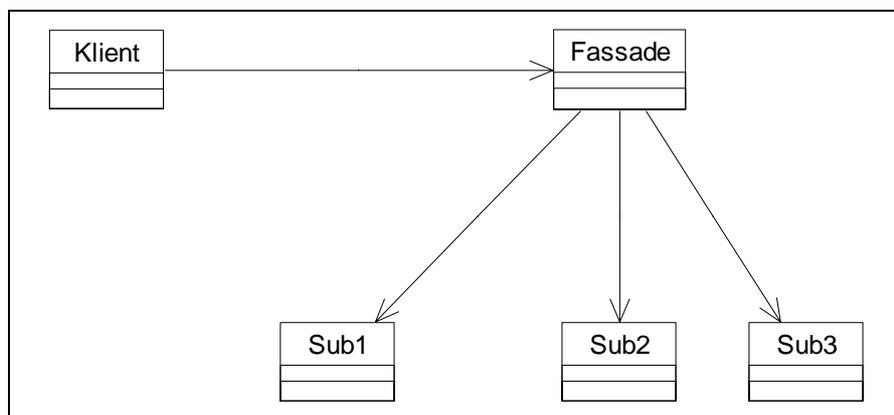
Die `Fassade` weiß, welche Klassen das Subsystem hat und welche für eine bestimmte Anfrage zuständig sind. Diese Anfragen von dem Klienten leitet sie an die Subklasse weiter, vgl. [A5.2.1].

Subsystem

In den Klassen des Subsystems ist die Subsystemfunktionalität implementiert. Die kennen die `Fassade` selbst nicht. In ihnen werden die von der `Fassade` weitergeleiteten Aufgaben bearbeitet, vgl. [A5.2.1].

Klient

Der `Klient` schickt Anfragen an das Subsystem an die `Fassade`, vgl. [A5.2.1].



[A5.2.1] UML-Klassendiagramm der Fassade

5.3 Animation

Metadaten

Thema: Der Pförtner im Verwaltungsbereich des Pattern Parks als Beispiel für eine Schnittstelle zwischen Klienten und Subsystem

Dauer: 2:10 Minuten



[A5.3.1] Screenshot der Animation zur Fassade

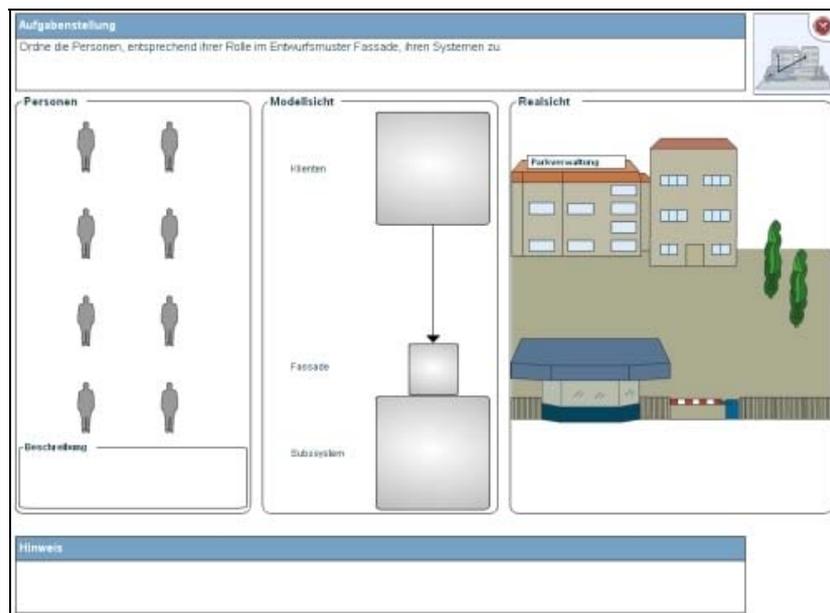
Sprechertext

„Die Angestellten des Pattern Parks haben ihre Arbeitsplätze in einem eigenen Verwaltungsbereich. Andere Personen, wie zum Beispiel die Besucher des Parks, haben keinen direkten Zugang zu diesem Bereich. Aus diesem Grund gibt es einen Pförtner, der nur dafür zuständig ist, Anfragen von außen anzunehmen und weiterzuleiten. Dies kann zum Beispiel eine Frage eines Besuchers oder eine Paketlieferung sein. Eine Menge von Klassen, die in enger Beziehung zueinander stehen, bezeichnet man als Subsystem. In unserem Beispiel sind das alle Klassen, die dem Verwaltungsbereich zuzuordnen sind, also zum Beispiel die Angestellten, die Arbeitsgeräte, die sie verwenden, wie Drucker und Kopierer oder auch Datenbanken. Als Klienten bezeichnet man im Allgemeinen Klassen, die von außen auf die angebotenen Dienste zugreifen. In unserem Beispiel sind das die Besucher und die Lieferanten. Diese sind also nicht Teil des Subsystems und sollen auch nicht einfach so darauf zugreifen können. Die Daten des Subsystems sind nach außen gekapselt. Deswegen hat der Verwaltungsbereich einen Pförtner, an den sich alle Klienten wenden können. Er ist eine Fassade, er fungiert als Schnittstelle. Der Pförtner kann natürlich nicht alle Anfragen selbst beantworten. Er hat aber Verbindungen zu den entsprechenden Klassen und kann daher die Anfragen weiterleiten. Dieses Prinzip wird als Delegation bezeichnet.“

5.4 Übung ohne Einbindung von UML-Diagrammen

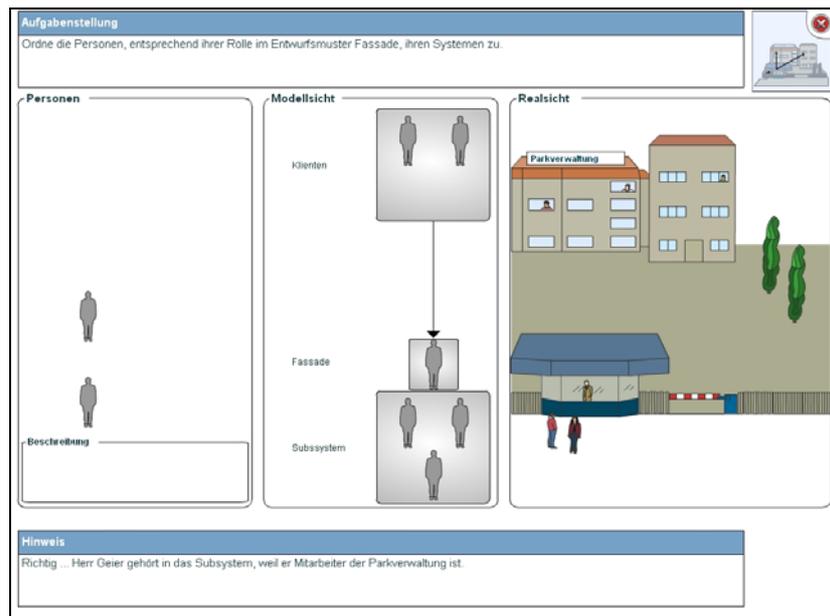
- Thema:** Zuordnung von Personen im Park zu den Gruppen Klienten, Subsystem und Fassade
- Dauer:** etwa 5 Minuten
- Lernziele:** Verständnis für die Aufteilung in Klienten, Subsystem und Fassade
- Vorkenntnisse:** Einführungsanimation Fassade

Das Entwurfsmuster Fassade wird ohne Verwendung von UML-Diagrammen wie folgt behandelt: Der Benutzer muss eine sinnvolle Zuordnung von Klient, Fassade und Subsystem vornehmen. Gegeben ist der Pförtner, Angestellte anderer Firmen, Besucher des Parks und Mitarbeiter der Verwaltung. Der Benutzer muss nun diese Rollen den Bezeichnungen Klient, Fassade und Subsystem zuordnen, vgl. [A5.4.1].



[A5.4.1] Startbild der Übung ohne UML zur Fassade

Durch Hineinziehen der Figuren in die entsprechenden Kästen, erlernt der Benutzer, Rollen einem Schema zuzuordnen. Durch Bewegen des Mauszeigers über eine Figur werden dem Benutzer die entsprechende Rolle und eine kurze Beschreibung der Rolle, in dem dazu vorgesehenen Beschreibungskasten, angezeigt. Sobald eine Person richtig gesetzt wurde, wird diese auch, entsprechend ihrer Systemzugehörigkeit, in der Realsicht angezeigt, vgl. [A5.4.1].



[A5.4.2] angeordnete Personen in der Übung ohne UML zur Fassade

Der Benutzer bemerkt im Verlauf, dass die Fassade nur durch eine Person realisiert werden kann. Diese soll in dieser Übung nicht durch die Rolle des Pförtners, der in der Animation eingeführt wurde, dargestellt werden. Alternativ wird diese Rolle durch einen „Telefonist“ eingenommen, welcher Anfragen an das Subsystem weiterleitet. Sobald eine Zuordnung fehlerhaft ist, wird dies dem Benutzer entsprechend durch Einblenden eines Hinweises gemeldet werden. Die Übung ist beendet, sobald alle Figuren richtig gesetzt sind.

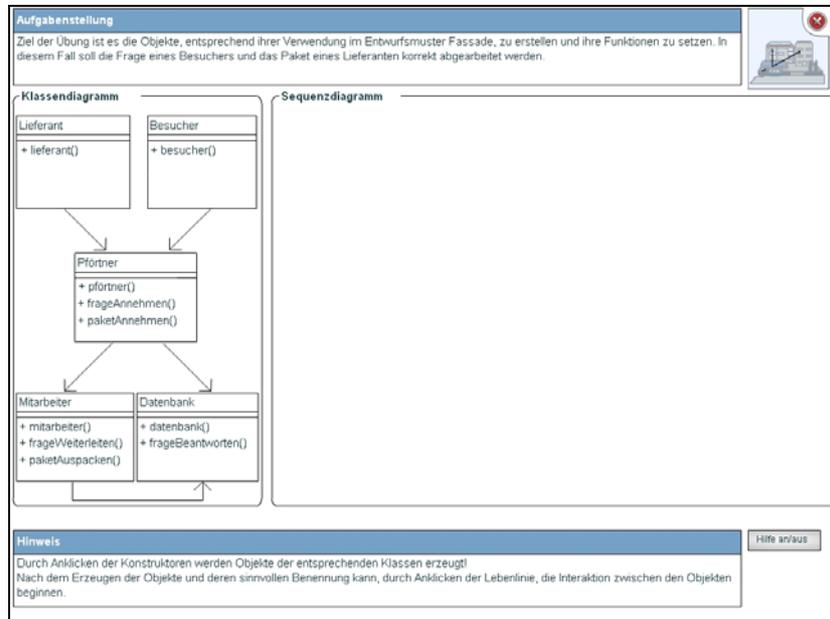
5.5 Übung mit Einbindung von UML-Diagrammen

- Thema:** Die Kommunikation zwischen Besuchern, Lieferanten, Pförtner und Angestellten als Sequenzdiagramm
- Dauer:** etwa 15 Minuten
- Lernziele:** Verbesserung des Verständnisses für die Funktionsweise des Entwurfsmusters Fassade (Schnittstelle, Delegation)
- Vorkenntnisse:** Grundlagen Sequenzdiagramm
 Verständnis des Klassendiagramms Fassade

In dieser Übung wird mithilfe eines Sequenzdiagramms die Kommunikationsstruktur der Fassade thematisiert. Es soll insbesondere deutlich werden, dass Klienten nur mit der Fassade kommunizieren können und dass diese die Anfragen in der Regel nicht selbst beantwortet, sondern an entsprechende Stellen des Subsystems delegiert, dadurch wird auf die

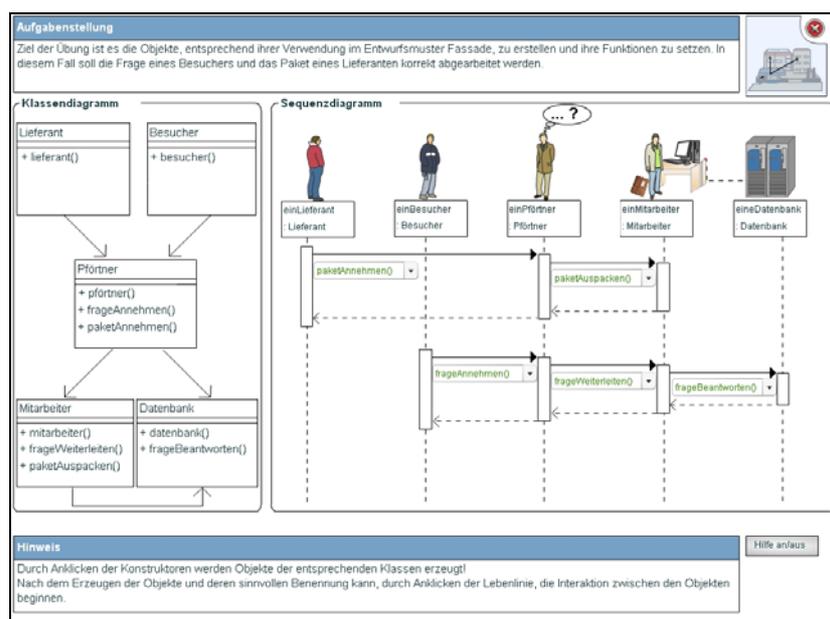
fundamentalen Ideen des Entwurfsmusters – Datenkapselung, Delegation und Schnittstelle – eingegangen.

Zu Beginn ist ein Klassendiagramm zu dem durch die Animation bekannten Szenario gegeben, vgl. Kap. 5.3 und [A5.5.1].



[A5.5.1] Startbild der Übung mit UML zur Fassade

Durch Aktivieren der Konstruktoren im Klassendiagramm werden die entsprechenden Objekte im Sequenzdiagramm erzeugt, die dann durch Auswählen des Benutzers miteinander kommunizieren können.



[A5.5.2] Musterlösung zum 1. Handlungsablauf der Übung mit UML zur Fassade

Dem Benutzer wird nach dem Erzeugen der Objekte und dem Auswählen der Kommunikationspfade eine Reihe von Methoden zum Auswählen angeboten. Er wird damit aufgefordert, die Handlungsabläufe, die in der Aufgabenstellung beschrieben sind, im Sequenzdiagramm nachzustellen, vgl. [A5.5.2]. Bei unerlaubten Vorgängen (z.B. wenn ein Besucher mit einem Mitarbeiter direkt kommunizieren will) oder bei Vorgängen, die nicht dem vorgegebenen Handlungsablauf entsprechen, erfolgt eine erläuternde Rückmeldung in einem dafür reservierten Fenster.

Handlungsabläufe:

1)

Ein Besucher stellt dem Pförtner die Frage, wann der neue Wasserbob fertig gestellt ist. Der Pförtner kann diese Frage nicht beantworten, weiß aber, dass ein Mitarbeiter den Bau des Wasserbobs plant und ruft ihn deshalb an, dieser leitet die Frage an die Datenbank weiter, da dort der aktuelle Stand des Baus hinterlegt ist, vgl. [A5.5.2].

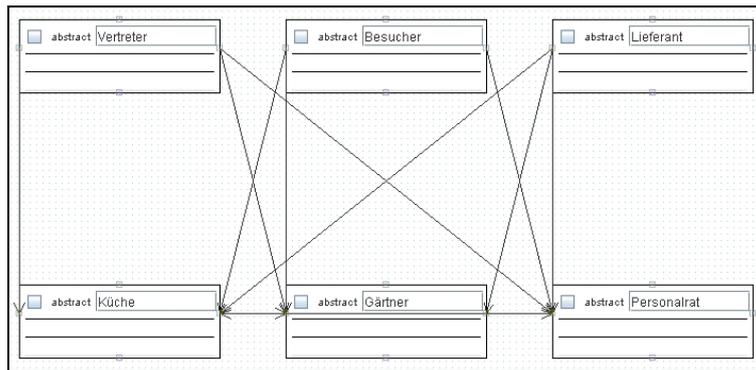
2)

Ein Lieferant hat ein Paket für den Mitarbeiter abzugeben. Der Pförtner nimmt das Paket entgegen und leitet es wenig später an den Mitarbeiter weiter.

5.6 UML-Puzzle

Aufgabenstellung:

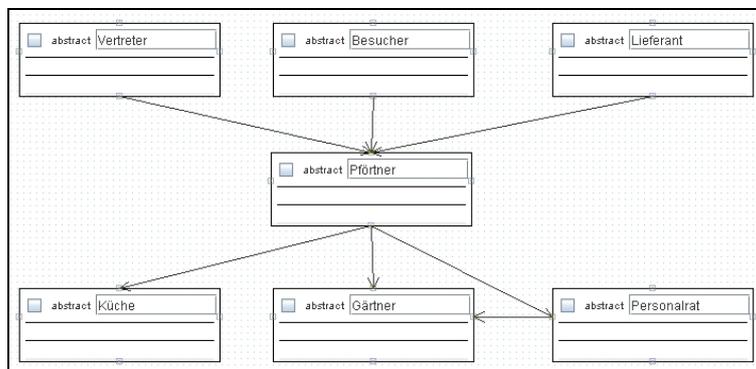
„Als Beispiel für das Entwurfsmuster Fassade dient der Pförtner des Pattern Parks. Die Klienten dürfen die Mitarbeiter des Parks nur über den Pförtner ansprechen. Füge daher eine Klasse Pförtner in das Klassendiagramm ein. Die Klientenklassen sind repräsentiert durch Besucher, Lieferant und Vertreter, die Mitarbeiter durch Küche, Gärtner und Personalrat. Die vorhandenen Beziehungen zwischen den Klassen müssen dann verändert werden. Zum Schluss sollen auch gerichtete Assoziationen vorhanden sein. Abbildung [A5.6.1] zeigt die Ausgangssituation des UML-Puzzles zur Fassade.“



[A5.6.1] Ausgangssituation des UML-Puzzles zur Fassade

Beschreibung des Algorithmus zur Überprüfung der Lösung

Es wird überprüft, ob die Klasse `Pförtner` eingefügt wurde. Fehlt diese wird ein entsprechender Fehlertext bei der Überprüfung ausgegeben. Ebenso werden alle Assoziationen sowohl zwischen `Pförtner` und den drei Klassen des Subsystems als auch zwischen den Klienten-Klassen und `Pförtner` überprüft. Erst wenn alle Kriterien erfüllt werden, wird die Aufgabe als gelöst gekennzeichnet. Alles was darüber hinaus noch hinzugefügt wird, wird bei der Überprüfung nicht berücksichtigt. Abbildung [A5.6.2] zeigt die Musterlösung des UML-Puzzles zur Fassade.



[A5.6.2] Musterlösung zum UML-Puzzle zur Fassade

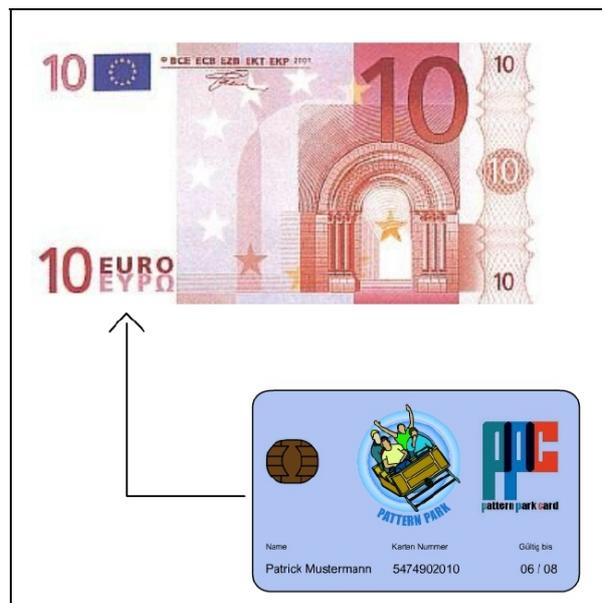
6 Modul Proxy

6.1 Inhalt der Übersichtsseite

Beispiel:

Gutschein

Logo:



[A6.1.1] Logo Proxy

Beschreibung:

Eine Geldkarte ist ein Stellvertreter, mit dem man den Zugriff auf das Guthaben kontrollieren kann.

Vorteile:

- Die Geldkarte kann zusätzliche Aktionen ausführen, zum Beispiel das Kaufverhalten des Besuchers speichern.
- Über eine persönliche Identifikationsnummer (PIN) kann man sein Guthaben vor fremden Zugriffen schützen.

Nachteile:

- Keine

6.2 Glossareintrag

Beschreibung:

Ein Proxy ist ein Stellvertreter, mit dem man den Zugriff auf ein Objekt kontrollieren kann. Es gehört zu der Familie der objektbasierten Strukturmuster.

Vorteile:

- Der Proxy kann zusätzliche Aktionen ausführen („Smart-Reference“).
- Mit einem Schutzproxy kann man unterschiedliche Zugriffsrechte kontrollieren.

Proxy

Die Klasse `Proxy` verwaltet eine Referenz, die es ermöglicht, auf die Klasse `EchtesSubjekt` zuzugreifen. Das `Proxy` kann die Schnittstelle der Klasse `Subjekt` verwenden, wenn sie mit der von `EchtesSubjekt` identisch ist. Die Klasse `Proxy` ist dafür zuständig `EchtesSubjekt` aufzurufen, und ggf. sichtbar werden zu lassen, vgl. [A6.2.1].

Subjekt

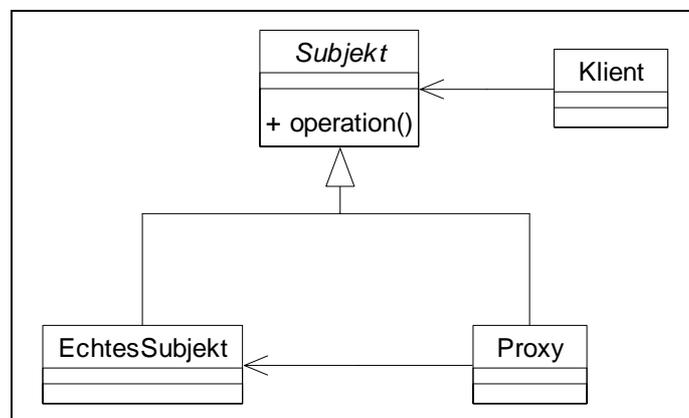
Das `Subjekt` definiert die gemeinsame Schnittstelle von `EchtesSubjekt` und `Proxy`, so dass `Proxy` überall benutzt werden kann, wo `EchtesSubjekt` erwartet wird, vgl. [A6.2.1].

EchtesSubjekt

Die Klasse `EchtesSubjekt` definiert das eigentliche Objekt, das durch `Proxy` repräsentiert wird, vgl. [A6.2.1].

Klient

Von `Klient` geht eine Anfrage an `Subjekt`, vgl. [A6.2.1].



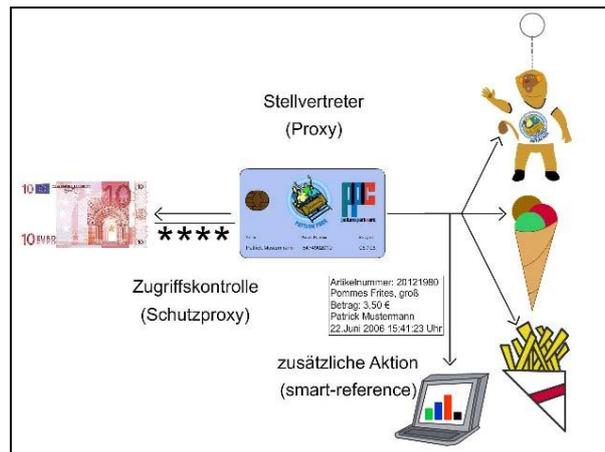
[A6.2.1] UML-Klassendiagramm des Entwurfsmusters Proxy

6.3 Animation

Metadaten

Thema: Eine Geldkarte im Pattern Park als Beispiel für einen Stellvertreter mit Zugriffsschutz und zusätzlichen Aktionen

Dauer: 1:25 Minuten



[A6.3.1] Screenshot der Animation zum Proxy

Sprechertext

„An einem Automaten im Pattern Park können die Besucher eine Guthabekarte erwerben. Diese Pattern Park Card kann innerhalb des Parks als Zahlungsmittel verwendet werden, zum Beispiel für Souvenirs, Eis oder Snacks. Ein Besucher erwirbt eine Karte im Wert von 10 Euro. Die Pattern Park Card kann nun stellvertretend für den eingezahlten Betrag verwendet werden. So einen Stellvertreter nennt man auch Proxy. Zum Schutz vor Missbrauch muss der Benutzer bei jedem Bezahlvorgang eine 4 stellige Nummer eingeben, die nur ihm bekannt ist. Der Zugriff auf das eingezahlte Geld wird somit kontrolliert, damit nicht jeder die Karte benutzen kann. Ein Proxy mit dieser Funktion wird Schutzproxy genannt. Was der Besucher aber nicht weiß, ist, dass jedes Mal bei der Benutzung der Karte eine zusätzliche Aktion ausgeführt wird. Die Nummer des bezahlten Artikels wird an einen zentralen Computer gesendet, um Statistiken über das Kaufverhalten der Besucher zu erstellen. Ein Proxy, der zusätzliche Aktionen ausführt, wird als Smart-Reference-Proxy bezeichnet.“

6.4 Übung ohne Einbindung von UML-Diagrammen

Thema: Lebensweltliche Beispiele für Proxy, Klienten und Subjekte

Dauer: etwa 5 Minuten

Lernziele: Verständnis für die Aufteilung in Proxy, Klienten und Subjekte

Vorkenntnisse: Einführungsanimation Proxy

Aufgabenstellung
Die Aufgabe besteht darin die aufgeführten Objekte entsprechend ihrer Rolle nach dem Proxy-Entwurfsmuster zuzuordnen. Der Klient benutzt den Proxy, um das eigentliche Objekt benutzen oder auf dieses zugreifen zu können.

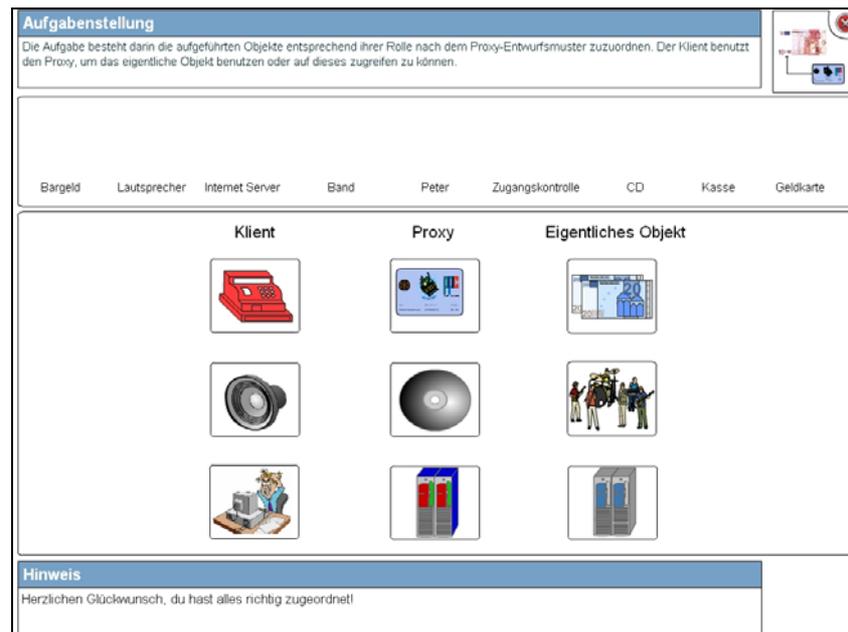
Bargeld Lautsprecher Internet Server Band Peter Zugangskontrolle CD Kasse Geldkarte

Klient	Proxy	Eigentliches Objekt
<input type="text"/>	<input type="text"/>	<input type="text"/>
<input type="text"/>	<input type="text"/>	<input type="text"/>
<input type="text"/>	<input type="text"/>	<input type="text"/>

Hinweis
Klicke einmal auf eines der Symbole und setze es dann in das richtige Feld.

[A6.4.1] Startbild der Übung ohne UML zum Proxy

Um dem Lernenden das Stellvertreter-Prinzip des Proxy näher zu bringen, werden in diesem Lernmodul Objekte aus dem Alltag, hinsichtlich ihrer Rolle innerhalb des Proxy-Entwurfsmusters verwendet. Abbildung [A6.4.1] zeigt das Startbild der Übung. In dieser Übung geht es darum, lebensweltliche Beispiele den Teilnehmern des Proxy-Entwurfsmusters zuzuordnen. Hierzu wird das Beispiel aus der Einführungsanimation zum Proxy mit Geldkarte, Bargeld und Kasse aufgegriffen. Die weiteren Beispiele beinhalten eine CD, eine Musikband und Lautsprecher sowie einen Internet-Server, einen Computer-Benutzer und eine Zugangskontrolle. Abbildung [A6.4.2] zeigt die Musterlösung zu dieser Übung.



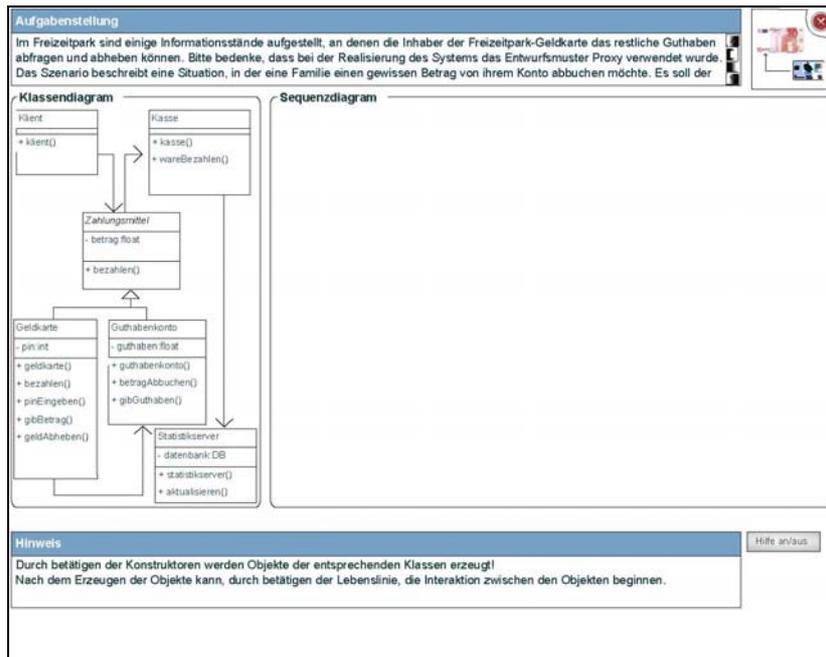
[A6.4.2] Musterlösung der Übung ohne UML zum Proxy

6.5 Übung mit Einbindung von UML-Diagrammen

- Thema:** Die Kommunikation zwischen Besuchern, Geldkarte, Konto, Kasse und Statistik-Server als Sequenzdiagramm
- Dauer:** etwa 15 Minuten
- Lernziele:** Verbesserung des Verständnisses für die Funktionsweise des Entwurfsmusters Proxy (Zugriffsschutz und Schnittstelle)
- Vorkenntnisse:** Grundlagen Sequenzdiagramm
Verständnis des Klassendiagramms Proxy
Einführungsanimation Proxy

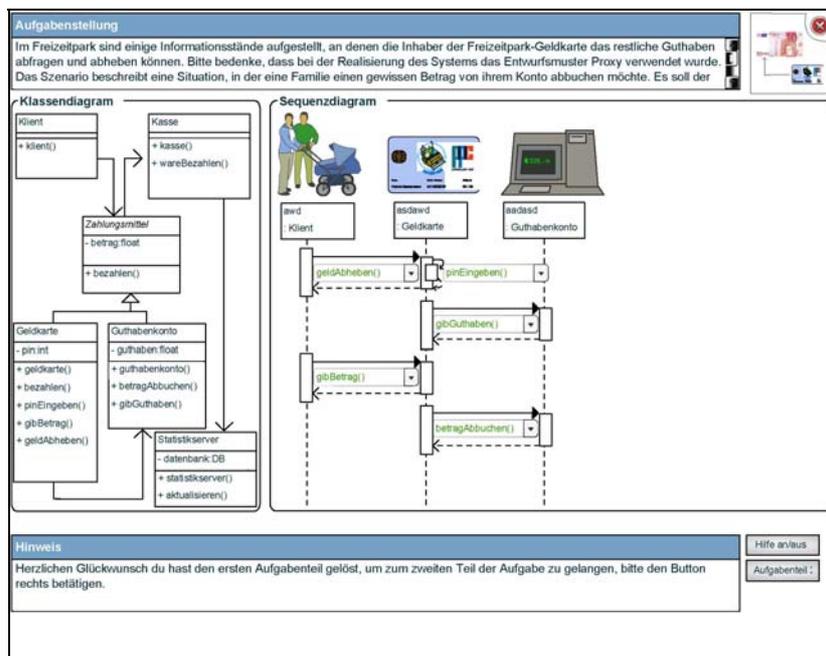
In dieser Übung wird die Funktion des Proxy-Entwurfsmusters mithilfe eines Sequenzdiagramms thematisiert. Der Benutzer soll die in der Aufgabenstellung beschriebenen Handlungsabläufe im Sequenzdiagramm modellieren, dabei wird auf die wichtigen Konzepte des Entwurfsmusters – Zugriffsschutz und Schnittstelle – eingegangen. Die Übung ist an die Animation und die dort vorgestellten verschiedenen Proxyarten angelehnt.

Dem Benutzer steht zu Beginn ein Klassendiagramm zur Verfügung, welches die Funktionalität der einzelnen Klassen erläutert, vgl. [A6.5.1].



[A6.5.1] Startbild der Übung mit UML zum Proxy

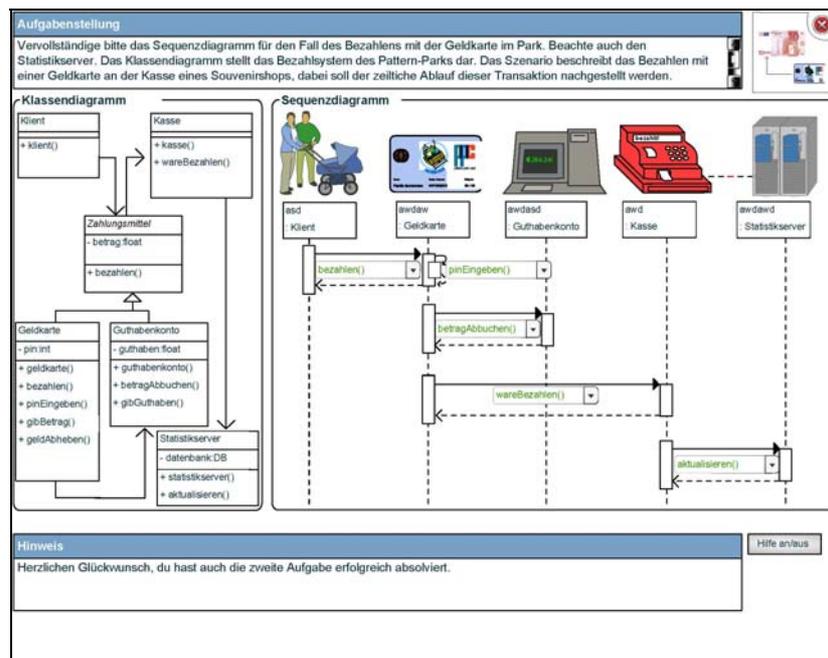
Durch Auswahl der Konstruktoren im Klassendiagramm werden die entsprechenden Objekte im Sequenzdiagramm erzeugt, die dann durch Auswählen des Benutzers miteinander kommunizieren. Nachdem der Benutzer den Kommunikationspfad erzeugt hat, kann er aus einer Liste von Methoden auswählen und so den Ablauf der zwei implementierten Handlungsabläufe nachbauen.



[A6.5.2] erste Aufgabe, Geldabbuchen, der Übung mit UML zum Proxy

Aufgabenstellung 1:

Im Freizeitpark sind einige Informationsstände aufgestellt, an denen die Inhaber der Freizeitpark-Geldkarte das restliche Guthaben abfragen und abheben können. Bitte bedenke, dass bei der Realisierung des Systems das Entwurfsmuster Proxy verwendet wurde. Das Szenario beschreibt eine Situation, in der eine Familie einen gewissen Betrag von ihrem Konto abbuchen möchte, vgl. [A6.5.2].



[A6.5.3] zweite Aufgabe, Bezahlen in einem Souvenirshop, der Übung mit UML zum Proxy

Aufgabenstellung 2:

Vervollständige bitte das Sequenzdiagramm für den Fall des Bezahls mit der Geldkarte im Park. Das Klassendiagramm stellt das Bezahlsystem des Pattern Parks dar. Das Szenario beschreibt das Bezahlen mit einer Geldkarte an der Kasse eines Souvenirshops, dabei soll der zeitliche Ablauf dieser Transaktion nachgestellt werden, vgl. [A6.5.3].

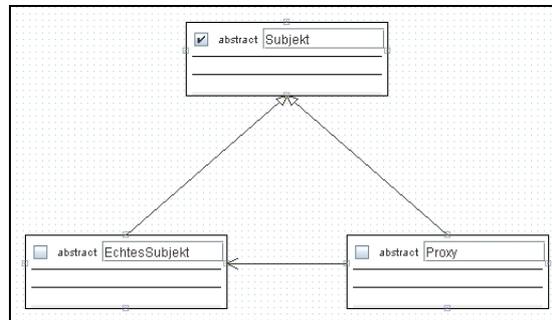
6.6 UML-Puzzle

Aufgabenstellung:

„Das Klassendiagramm zeigt ein allgemeines Proxy-Entwurfsmuster. Dieses soll nun, an das Beispiel im Pattern Park angelehnt, zu einem konkreten Diagramm mit den Klassen Gutscheine, Geld und Geldbetrag geändert werden. Benenne die Klassen entsprechend um. Nun werden spezielle Varianten des Proxy gefordert, und zwar die des Schutzproxy. Füge für die spezielle Variante des Schutzproxys eine Klasse Identität mit einer Beziehung zu der richti-

gen Klasse im Diagramm ein. Der Gutschein soll dabei als Schutzproxy fungieren und soll eine entsprechende Methode (überprüfe) zur Überprüfung erhalten.“

Abbildung [A6.6.1] zeigt die Ausgangssituation des UML-Puzzles zum Proxy.

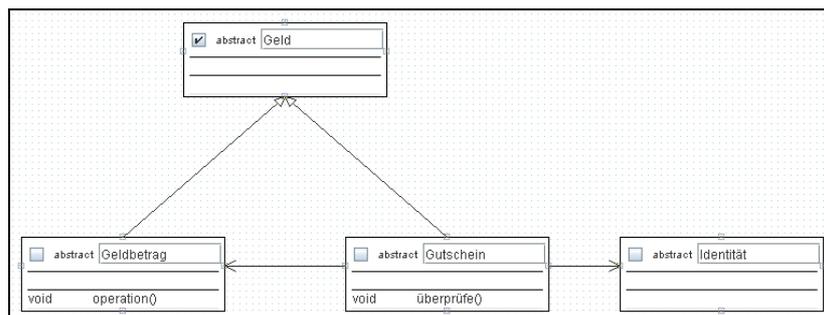


[A6.6.1] Ausgangssituation des UML-Puzzles zum Proxy

Beschreibung des Algorithmus zur Überprüfung der Lösung

Es wird überprüft, ob die Klassen Geld, Geldbetrag, Gutschein und Identität vorhanden sind. Fehlt eine von diesen Klassen, bzw. ist noch nicht umbenannt worden, wird ein entsprechender Fehlertext bei der Überprüfung ausgegeben. Ebenso werden die Anordnung der Klassen und die Assoziation zwischen Gutschein und Identität überprüft. Die Operation überprüfe() (in der Klassen Gutschein) muss vorhanden sein. Erst wenn alle Kriterien erfüllt werden, wird die Aufgabe als gelöst gekennzeichnet. Alles was darüber hinaus noch hinzugefügt wird, wird bei der Überprüfung nicht berücksichtigt.

Abbildung [A6.6.2] zeigt die Musterlösung zum UML-Puzzle zum Proxy.



[A6.6.2] Musterlösung zum UML-Puzzle zum Proxy

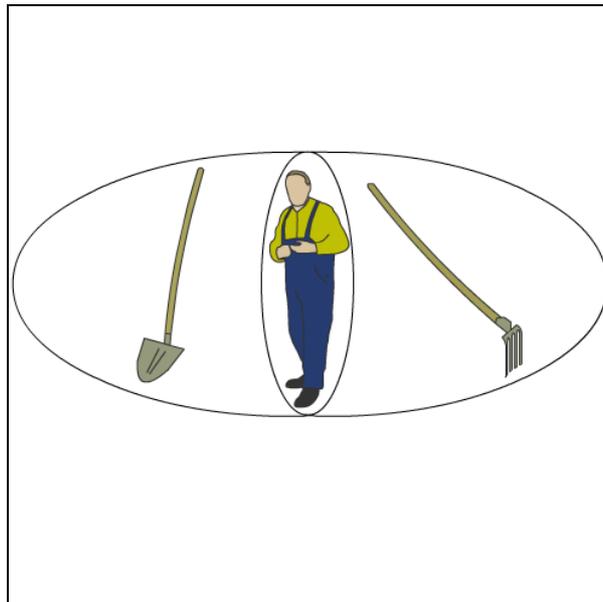
7 Modul Dekorierer

7.1 Inhalt der Übersichtsseite

Beispiel

Werkzeuge und Arbeitskleidung von Mitarbeitern im Park

Logo



[A7.1.1] Logo Dekorierer

Beschreibung:

Ein Mitarbeiter des Freizeitparks kann seinen Tätigkeitsbereich erweitern, wenn er bestimmte Werkzeuge benutzt oder bestimmte Kleidung trägt.

Vorteile:

- Der Mitarbeiter selbst wird dadurch nicht verändert.
- Der Mitarbeiter kann seine Tätigkeiten im Laufe eines Tages ändern, indem er Werkzeuge austauscht oder seine Arbeitskleidung wechselt.

Nachteile:

- Ein Mitarbeiter ohne die richtige Arbeitskleidung kann nicht dasselbe machen, wie derselbe Mitarbeiter mit Arbeitskleidung.

7.2 Glossareintrag

Beschreibung

Eine Komponente kann in ihrer Funktionalität erweitert werden, indem sie von einem Dekorierer umschlossen wird. Es gehört zu der Familie der objektbasierten Strukturmuster.

Vorteile

- Die Klasse der Komponente ändert sich nicht.
- Die Funktionalität kann zur Laufzeit hinzugefügt werden, im Gegensatz zur Vererbung.
- Klienten werden durch die Funktionalitätserweiterung der Komponente nicht beeinträchtigt.

Nachteile

- Die dekorierte Komponente ist nicht identisch mit der ursprünglichen Komponente.
- Es entstehen viele, kleine, ähnliche Objekte.

Dekorierer

Der Dekorierer verwaltet eine Referenz auf ein Komponentenobjekt und definiert eine Schnittstelle, die der Schnittstelle der Komponente entspricht, vgl. [A7.2.1].

KonkreterDekorierer

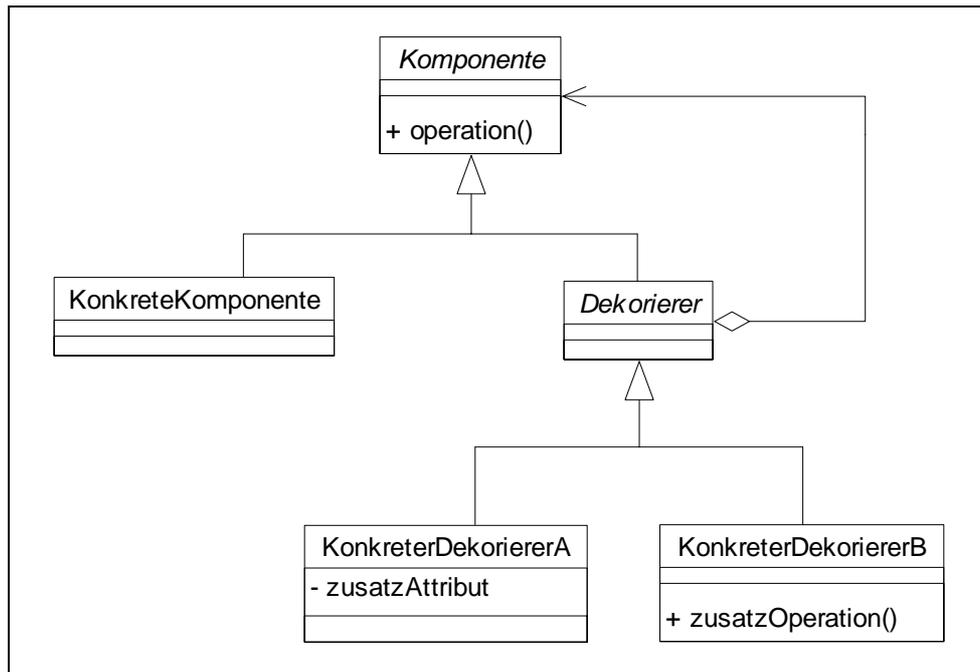
Der `konkreterDekorierer` fügt der Komponente Funktionalitäten hinzu, vgl. [A7.2.1].

Komponente

Die Klasse `Komponente` definiert die Schnittstelle für die Objekte in der zusammengesetzten Struktur von `Kompositum`. Hier wird ein mögliches Standardverhalten der Klassen `KonkreteKomponente` implementiert. Der Zugriff auf die Objekte und dessen Verwaltung wird in dieser Klasse geregelt. Optional wird auch der Zugriff auf das Elternobjekt deklariert und ggf. implementiert, vgl. [A7.2.1].

KonkreteKomponente

Die `konkreteKomponente` definiert ein Objekt, das um zusätzliche Funktionalität erweitert werden kann, vgl. [A7.2.1].



[A7.2.1] UML-Klassendiagramm des Dekorierers

7.3 Animation

Metadaten

Thema: Die Arbeitskleidung von Mitarbeitern im Pattern Park als Beispiel für die funktionale Erweiterung von Objekten

Dauer: 1:10 Minuten

Arbeitsplan

Aufgabe:	Reinigungskraft	Techniker	Maskottchen	Eisverkäufer	Ballonverkäufer
Benötigt:	4	3	1	2	1
Anwesend:	5	4	0	1	1
Status:	+1	+1	-1	-1	✓

[A7.3.1] Screenshot der Animation zum Dekorierer

Sprechertext

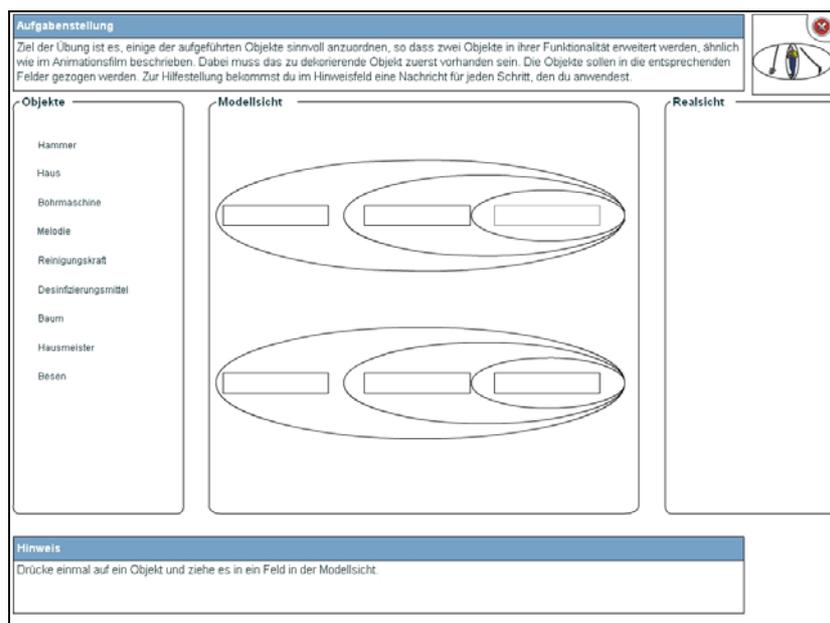
„Eine Übersichtstabelle zeigt die Anzahl der anwesenden Mitarbeiter für den heutigen Arbeitstag an. Es fällt auf, dass ein Mitarbeiter für die Rolle des Park-Maskottchens und ein Eisverkäufer fehlen. Außerdem sind heute eine Reinigungskraft und ein Techniker zuviel zur

Arbeit erschienen. So ist kein optimaler Parkbetrieb möglich! Mitarbeiter können mit anderer Arbeitskleidung eine neue Tätigkeit im Park ausüben. Diese Idee findet sich im Entwurfsmuster Dekorierer wieder: Die Funktionalität eines Objekts kann durch einen Dekorierer sinnvoll erweitert werden. Die Tätigkeiten der Parkmitarbeiter sind nun an den aktuellen Personalbedarf angepasst worden. Es können auch mehrere Dekorierer mit einem Objekt kombiniert werden, um dessen Funktionalität mehrfach zu erweitern.“

7.4 Übung ohne Einbindung von UML-Diagrammen

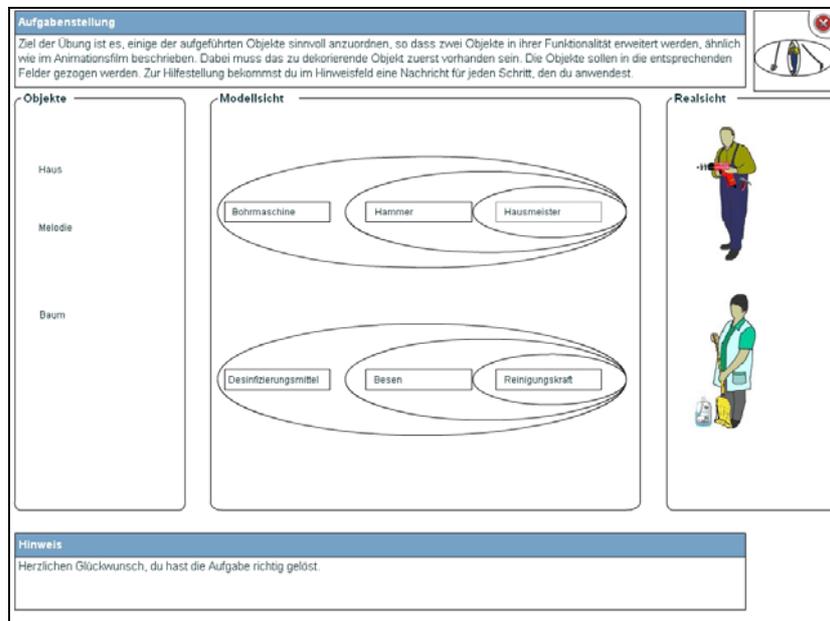
- Thema:** Erweiterung der Einsatzmöglichkeiten von Mitarbeitern im Park durch Werkzeuge
- Dauer:** etwa 5 Minuten
- Lernziele:** Verständnis für die funktionale Erweiterung von Objekten
- Vorkenntnisse:** Einführungsanimation Dekorierer

Die Übung des Dekorierers ohne Einbindung von UML verdeutlicht visuell und mit Bezug zur Realwelt die Funktionalität dieses Entwurfsmusters. Dabei stehen dem Benutzer drei Sichten zur Verfügung: In der Objektsicht befinden sich die Aufgabenelemente, darunter sowohl die konkreten Komponenten wie auch die Dekoriererkomponenten. Die Modellsicht verdeutlicht den Zusammenhang der Objekte untereinander. In der Realsicht werden dann die jeweiligen konkreten Komponenten mit ihren Dekorationen dekoriert und dargestellt, vgl. [A7.4.2].



[A7.4.1] Startbild der Übung ohne UML zum Dekorierer

Die Aufgabe des Benutzers besteht nun darin, die aufgeführten Objekte sinnvoll anzuordnen, dazu muss er diese in die Modellsicht ziehen und dort entsprechend ihrer Funktion ablegen. Im Falle eines falschen und nicht sinnvollen Ablegens bekommt der Nutzer einen entsprechenden Hinweis in dem dafür vorgesehenen Feld.



[A7.4.2] Beispiel einer sinnvollen Anordnung zur Übung ohne UML zum Dekorierer

7.5 Übung mit Einbindung von UML-Diagrammen

Metadaten

- Thema: Die funktionale Erweiterung des Hausmeisters durch seine Werkzeuge als Klassendiagramm
- Dauer: etwa 10 Minuten
- Lernziele: Verbesserung des Verständnisses für die Funktionsweise des Entwurfsmusters Dekorierer
- Vorkenntnisse: Grundlagen Klassendiagramm (Vererbung und Assoziation)

In dieser Übung wird das Entwurfsmuster Dekorierer mithilfe eines UML-Puzzles thematisiert.

Dem Benutzer stehen dazu drei Sichten zur Verfügung. Die Klassensicht beinhaltet die anzuordnenden Klassen. In der Klassendiagrammsicht wird das Dekorierer UML-Klassendiagramm skizziert, welches durch den Benutzer dann ausgefüllt werden soll. In der Realsicht werden die Schritte durch Bilder verdeutlicht, vgl. [A7.5.1].

Aufgabenstellung
 Ziel der Übung ist es, das dargestellte Klassendiagramm so zu vervollständigen, dass der Hausmeister seine Aufgaben erfüllen kann. Um dies zu überprüfen, kannst du deine Angaben auswerten. Aufgaben: Eingang fegen.

Klassen

Hammer	Hausmeister
+ typ	+ alter
+ benutzen()	+ benutzen()

Zange	Werkzeug
+ typ	+ typ
+ benutzen()	+ benutzen()

Besen	Schraubendreher
+ typ	+ typ
+ benutzen()	+ benutzen()

Messer	Leiter
+ typ	+ länge
+ schneiden()	+ benutzen()

Bohrmaschine	ReparaturKomponente
+ leistung	+ typ
+ bohren()	+ benutzen()

Klassendiagramm

Realsicht

Hinweis

auswerten
 Werkzeuge neu setzen

[A7.5.1] Startbild der Übung mit UML zum Dekorierer

Die Ausgangsposition ist die, dass der Hausmeister drei Aufgaben zu erledigen hat. Der Benutzer wird nun aufgefordert, das Klassendiagramm so zu vervollständigen, dass der Hausmeister diese Aufgaben erledigen kann. Dazu können die Klassen in die Klassendiagrammsicht gezogen und dort abgelegt werden, vgl. [A7.5.2]. Bei einem falschen oder nicht sinnvollen Ablegen der Klasse erscheint eine entsprechende Ausgabe im Hinweisfeld.

Sollte der Benutzer alle Klassen korrekt abgelegt haben, sich aber bei den Werkzeugen vertan haben, so dass der Hausmeister seine Aufgabe nicht erledigen kann, besteht die Möglichkeit durch Aktivieren der „Werkzeuge neu setzen“-Schaltfläche, diese neu anzuordnen.

Aufgabenstellung
 Ziel der Übung ist es, das dargestellte Klassendiagramm so zu vervollständigen, dass der Hausmeister seine Aufgaben erfüllen kann. Um dies zu überprüfen, kannst du deine Angaben auswerten. Aufgaben: Eingang fegen.

Klassen

Besen	Schraubendreher
+ typ	+ typ
+ benutzen()	+ benutzen()

Messer	Leiter
+ typ	+ länge
+ schneiden()	+ benutzen()

Klassendiagramm

Realsicht

Hinweis
 Der Hausmeister konnte mit dieser Kombination nur die zweite und dritte Aufgabe erledigen.

auswerten
 Werkzeuge neu setzen

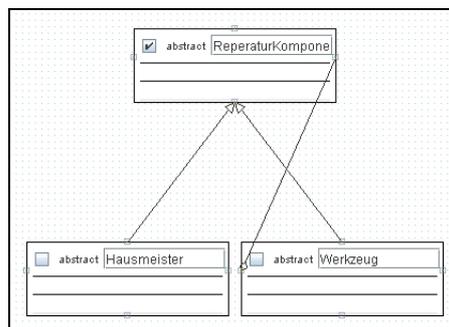
[A7.5.2] falsche Anordnung der Werkzeuge in der Übung mit UML zum Dekorierer

7.6 UML-Puzzle

Aufgabenstellung:

„Im Pattern Park gibt es einen Hausmeister, der – je nach dem welches Werkzeug er trägt – eine andere Rolle innehat. Die Austauschbarkeit dieser Werkzeuge soll mittels des Entwurfsmusters Dekorierer realisiert werden. Die gewünschten Werkzeuge müssen – entsprechend der allgemeinen Beschreibung zum Entwurfsmuster Dekorierer – als Klassen dem Diagramm hinzugefügt werden.“

Abbildung [A7.6.1] zeigt die Ausgangssituation des UML-Puzzles zum Dekorierer.

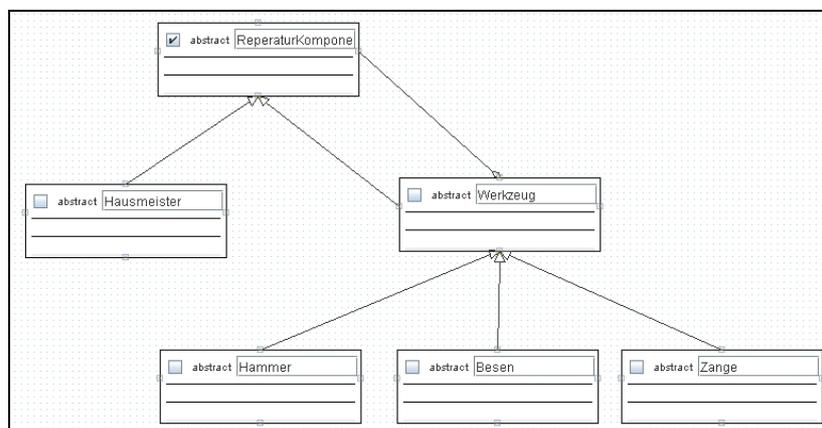


[A7.6.1] Ausgangssituation des UML-Puzzles zum Dekorierer

Beschreibung des Algorithmus zur Überprüfung der Lösung

Es wird überprüft, ob neue Klassen vorhanden sind. Diese müssen dann mit einer Vererbung zur Klasse Werkzeug verbunden werden. Fehlt eine Vererbung, bzw. ist noch keine neue Klasse hinzugefügt worden, wird ein entsprechender Fehlertext bei der Überprüfung ausgegeben. Erst wenn beide Kriterien erfüllt werden, wird die Aufgabe als gelöst gekennzeichnet. Alles was darüber hinaus noch hinzugefügt wird, wird bei der Überprüfung nicht berücksichtigt.

Abbildung [A7.6.2] zeigt die Musterlösung zum UML-Puzzle zum Dekorierer.



[A7.6.2] Musterlösung zum UML-Puzzle zum Dekorierer

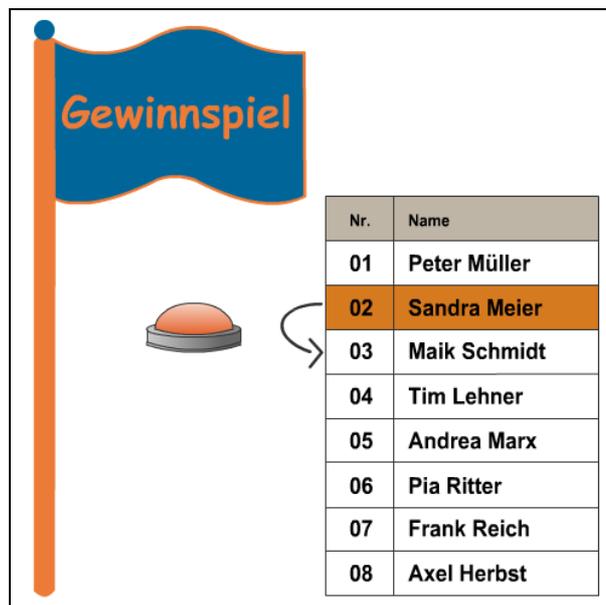
8 Modul Iterator

8.1 Inhalt der Übersichtsseite

Beispiel:

Durchlaufen einer Besucherliste

Logo:



[A8.1.1] Logo Iterator

Beschreibung:

Mit einem Iterator kann man nacheinander auf die Einträge einer Besucherliste zugreifen.

Vorteile:

- Für den Iterator ist es egal, wie die Besucherliste aufgebaut ist.
- Eine Besucherliste kann gleichzeitig mehrfach auf verschiedene Arten durchlaufen werden, zum Beispiel in alphabetischer und in quasi zufälliger Reihenfolge.

Nachteile:

- keine

8.2 Glossareintrag

Beschreibung:

Mit einem Iterator kann man nacheinander auf die Elemente eines zusammengesetzten Objekts zugreifen. Es gehört zu der Familie der objektbasierten Verhaltensmuster.

Vorteile:

- Die innere Struktur des zusammengesetzten Objekts bleibt verborgen.
- Das zusammengesetzte Objekt kann gleichzeitig mehrfach auf verschiedene Arten durchlaufen werden.
- Man kann auf Objekte unterschiedlicher Datenstrukturen einheitlich zugreifen und sie durchlaufen.

Nachteile:

- Der Einsatz des Iterators ist nur dann sinnvoll, wenn die Datenstruktur verschiedene Typen haben kann (Liste, Graph, Baum, Array).

Iterator

Der `Iterator` definiert eine Schnittstelle zum Zugriff auf und zum Durchlauf von Elementen, vgl. [A8.2.1].

KonkreterIterator

Der `konkreterIterator` implementiert die Schnittstelle aus dem `Iterator`. Die Klasse verwaltet die aktuelle Position während der Traversierung des Aggregats, vgl. [A8.2.1].

Aggregat

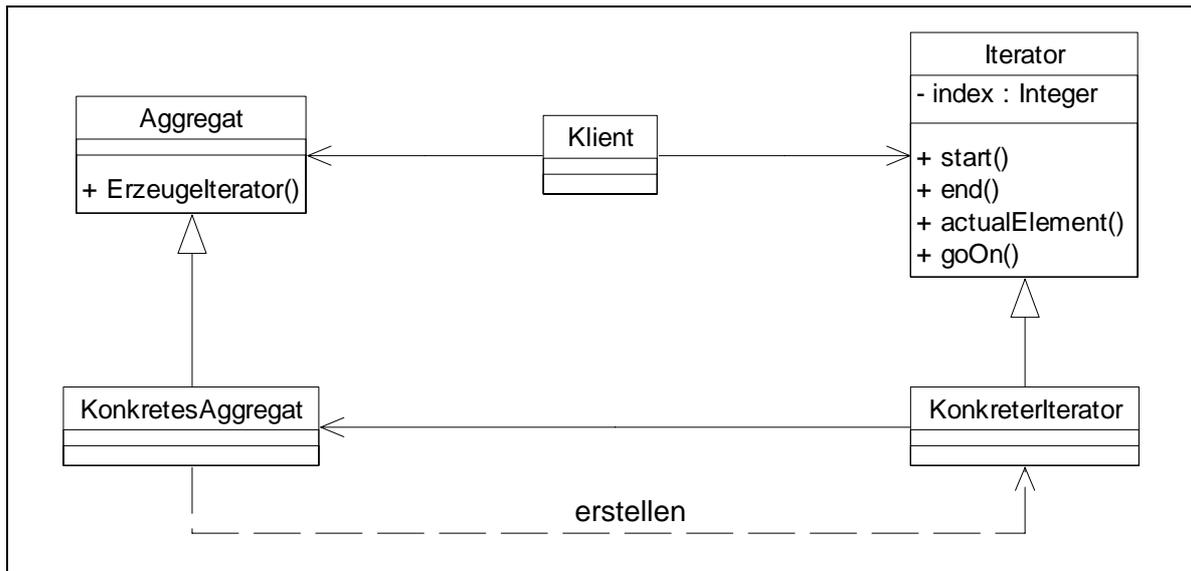
Das `Aggregat` definiert eine Schnittstelle zum Erzeugen eines Objekts der Klasse `Iterator`, vgl. [A8.2.1].

KonkretesAggregat

Das `konkretesAggregat` implementiert die Operation zum Erzeugen eines konkreten Iterators, indem es ein Objekt der passenden konkreten Iterator-Klasse zurückgibt, vgl. [A8.2.1].

Klient

Der `Klient` greift auf den `Iterator` und auf das `Aggregat` zu, vgl. [A8.2.1].



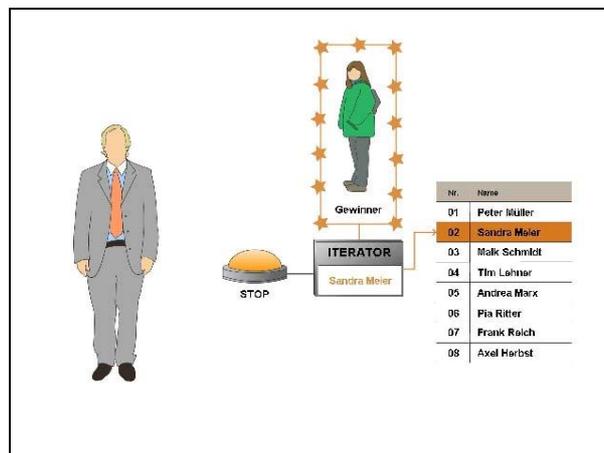
[A8.2.1] UML-Klassendiagramm des Iterators

8.3 Animation

Metadaten

Thema: Das Durchlaufen einer Liste von Besuchern mithilfe des Iterators

Dauer: 1:25 Minuten



[A8.3.1] Screenshot der Animation zum Iterator

Sprechertext

„Die Leitung des Pattern Parks überlegt sich, unter den Besuchern des Riesenrads eine Reise nach Hollywood zu verlosen. Hierfür werden von jedem Fahrgast, der in der Warteschlange steht, Name und eine Nummer in eine Liste eingetragen. Um nun den Gewinner zu ermitteln, muss es eine Möglichkeit geben, die Liste zu durchlaufen und das aktuelle Element

wiederzugeben. Dazu wird ein Zeiger auf dieses gesetzt. Nun kommt der Iterator ins Spiel. Er übernimmt die Aufgabe des Durchlaufens der Liste und liefert jeweils das aktuelle Element oder setzt den Zeiger auf das nachfolgende Element um. Der Iterator durchläuft die Liste. Ist er am Ende angelangt, fängt er wieder am Anfang an. Um den Iterator irgendwann zu stoppen, wird ein Schalter angeschlossen. Während der Verlosung drückt der Direktor des Parks den Schalter und stoppt damit den Iterator, der auf dem aktuellen Element der Liste stehen bleibt. Der Iterator zeigt nun den Gewinner an. Man kann den Iterator für das Durchlaufen einer beliebigen Liste verwenden.“

8.4 Übung ohne Einbindung von UML-Diagrammen

Thema:	Iteratoren für die Datenstrukturen Liste und Binärbaum
Dauer:	etwa 10 Minuten
Lernziele:	Verbesserung des Verständnisses für die Funktionsweise des Iterators
Vorkenntnisse:	Datenstrukturen Liste und Binärbaum, Pseudocode

Der Benutzer bekommt zwei Aufgabenteile präsentiert, die jeweils eine Datenstruktur behandeln, auf die das Iteratormuster angewendet werden soll. Zum einen ist dies eine einfach verkettete Liste, zum anderen die Datenstruktur Feld (Array).

Der Benutzer findet für die verschiedenen Methoden, welche für die jeweilige Datenstruktur verwendet werden Methodenrumpfe vor [Abb. 8.4.1]. Diese enthalten knapp formulierten Pseudocode für die Implementierung der jeweiligen Methode. Der Benutzer muss nun aus einer Combobox die Iterator-Methode (z.B. `weiter()`, `start ()`, `istFertig ()`) auswählen, welche zu dem Pseudocode passend ist und die Funktionalität der Methode der Datenstruktur übernimmt. Dies wird überprüft und ggf. werden Hilfestellungen in einem Hinweisfenster ausgegeben.

Die Anzahl der zu bearbeitenden Methodenrumpfe hängt von der Datenstruktur ab. Es werden jedoch immer soviel Methodenrumpfe wie Iterator-Methoden angezeigt. Zusätzlich können auch datenstruktureigene Methoden, wie `erzeugeElement ()` als Coderumpf angezeigt werden, welche jedoch nicht mit einer Iterator-Methode verknüpft werden können.

Die Übung ist erfolgreich abgeschlossen, wenn allen Methodenrumpfen entweder eine Iterator-Methode oder „keine Iterator-Methode“ zugeordnet wurde. Neben der beschriebenen Methodensicht wird in der Realsicht bei richtiger Auswahl einer Iterator-Methode eine grafi-

sche Ausgabe, etwa bei `weiter()` der Sprung von einem Listenelement auf das Nächste, angezeigt.

Der Wert der Übung liegt darin, zu verdeutlichen, dass ein Iterator auf einer Datenstruktur arbeitet, diese aber nach Außen hin nicht preisgibt. Er setzt eine vorher spezifizierte Schnittstelle zu einer Datenstruktur um. Des Weiteren arbeitet der Lernende die Methoden durch und ordnet die Pseudocodeausschnitte einer Funktionalität zu. Voraussetzung für die Übung ist demnach, dass dem Lernenden die Datenstrukturen bekannt sein müssen, welche in der Übung behandelt werden. In diesem Fall sind es die Datenstrukturen Feld (Array) und Liste.

Aufgabenstellung
 Die Aufgabe besteht darin, Code-Ausschnitte einer Datenstruktur eine Iteratormethode zuzuordnen. Diese wählst Du aus den jeweiligen Ausklappenmenüs aus. Dabei ist zu beachten, dass ein Iterator nur bestimmte Methoden einer Datenstruktur nach Außen hin weitergibt. Andere werden wiederum nur von der Datenstruktur implementiert. Die innere Struktur der Datenstruktur wird somit nicht preisgegeben.

Realsicht
 Diagramm zur Darstellung der Datenstruktur (Liste) und des Iterators. Ein Pfeil zeigt auf den aktuellen Element, der durch den Iterator angesprochen wird.

Methodensicht
 Datenstruktur: einfach verkettete Liste

<code>start()</code>	<code>isFertig()</code>	<code>weiter()</code>	<code>listHead=listFoot=NULL;</code>
<pre>{ currentElement=listHead; return currentElement; }</pre>	<pre>{ if(currentElement==NULL) { return true; } else { return false; } }</pre>	<pre>{ currentElement= currentElement.next; return currentElement; }</pre>	<pre>{ listHead=listFoot=NULL; }</pre>

Hinweis
 Richtig! Mit der Methode `start()` wird das aktuelle Listenelement auf den Listenkopf gesetzt.

[A8.4.1] Screenshot der Übung ohne UML zum Iterator

8.5 Übung mit Einbindung von UML-Diagrammen

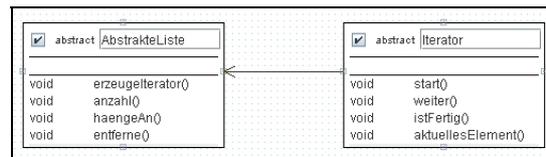
Im Modul Iterator gibt es keine Übung mit UML zusätzlich zum UML-Puzzle.

8.6 UML-Puzzle Iterator

Aufgabenstellung:

Das Klassendiagramm zeigt das Gerüst eines allgemeinen Iterator-Entwurfsmusters. Füge nun die Klassen `BesucherListe` und `BesucherListeIterator` hinzu und verbinde sie untereinander und mit den vorhandenen Klassen.

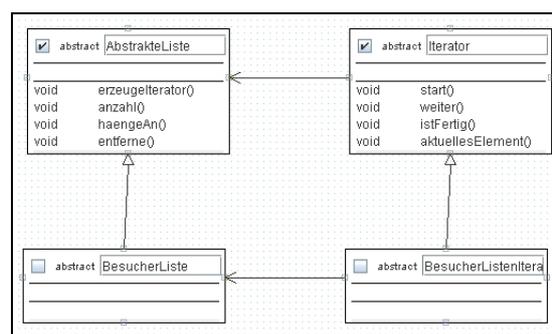
Abbildung [A8.6.1] zeigt die Ausgangssituation des UML-Puzzles zum Iterator.



[A8.6.1] Ausgangssituation des UML-Puzzles zum Iterator

Beschreibung des Algorithmus zur Überprüfung der Lösung

Es wird überprüft, ob die Klassen `BesucherListe` und `BesucherListeIterator` vorhanden sind. Fehlt eine von beiden Klassen wird ein entsprechender Fehlertext bei der Überprüfung ausgegeben. Ebenso wird die Assoziation zwischen `BesucherListe` und `BesucherListeIterator` überprüft. Die beiden Vererbungen werden auf Existenz und korrekte Anordnung überprüft. Erst wenn alle Kriterien erfüllt werden, wird die Aufgabe als gelöst gekennzeichnet. Alles was darüber hinaus noch hinzugefügt wird, wird bei der Überprüfung nicht berücksichtigt. Abbildung [A8.6.2] zeigt die Musterlösung zum UML-Puzzle zum Iterator.

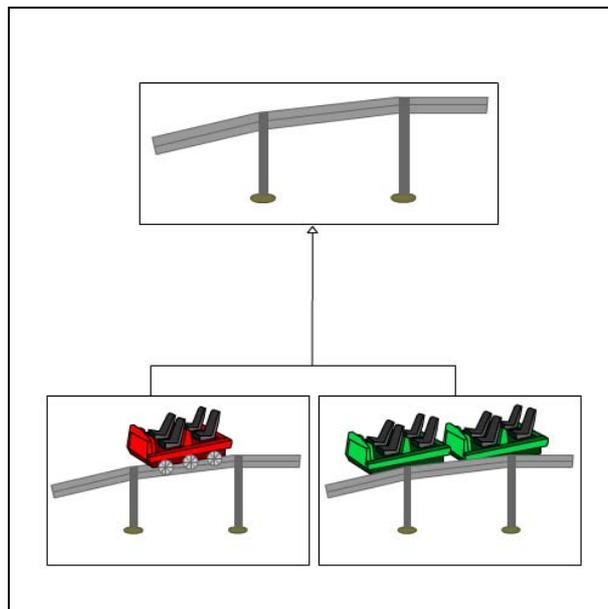


[A8.6.2] Musterlösung zum UML-Puzzle zum Iterator

9 Modul Schablonenmethode

9.1 Inhalt der Übersichtsseite

Logo:



[A9.1.1] Logo Schablonenmethode

Beschreibung:

Eine Schablonenmethode legt die Arbeitsschritte fest, die beim Aufbau einer Achterbahn immer durchgeführt werden müssen, zum Beispiel Fundament gießen oder Sicherheitsvorschriften überprüfen. Einzelne Schritte, die je nach Achterbahn verschieden sind, werden speziell festgelegt.

Vorteil:

- Die gemeinsamen Schritte müssen nur einmal festgelegt und beschrieben werden und können dann für alle Achterbahnen wiederholt werden.

Nachteil:

- Eine Schablonenmethode macht natürlich nur Sinn, wenn es mehrere Arbeitsschritte gibt, die beim Aufbau von verschiedenen Achterbahnen gleich sind.

9.2 Glossareintrag

Beschreibung:

Eine Schablonenmethode legt das Grundgerüst eines Algorithmus fest, einzelne Schritte werden in verschiedenen Unterklassen beschrieben. Es gehört zu der Familie der klassenbasierten Verhaltensmuster.

Vorteil:

Code kann wieder verwendet werden. Die gemeinsamen Teile eines Algorithmus müssen nämlich nur einmal festgelegt und können von mehreren Klassen benutzt werden.

Nachteil:

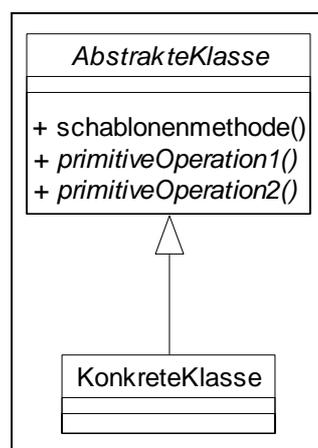
Eine Schablonenmethode sollte nur verwendet werden, wenn mehrere Methoden viele Gemeinsamkeiten haben. Ansonsten besteht die Gefahr der Unübersichtlichkeit.

Abstrakte Klasse

Die `AbstrakteKlasse` definiert abstrakte primitive Operationen, die von den konkreten Klassen implementiert werden müssen, damit die Schritte eines Algorithmus genau definiert werden. In dieser Klasse gibt es die Schablonenmethode, in der das Skelett des Algorithmus definiert wird. Dabei werden auch die primitiven Operationen aufgerufen, vgl. [A9.2.1].

Konkrete Klasse

Die `KonkreteKlasse` implementiert die primitiven Operationen, die die unterklassenspezifischen Schritte des Algorithmus ausführen, vgl. [A9.2.1].



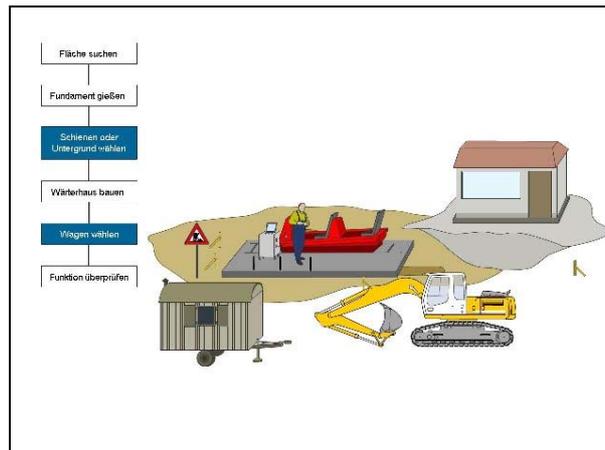
[A9.2.1] UML-Klassendiagramm der Schablonenmethode

9.3 Animation

Metadaten

Thema: Die gemeinsamen Schritte beim Aufbau verschiedener Bahnen als Beispiel für das Vererbungsprinzip

Dauer: 1:40 Minuten



[A9.3.1] Screenshot der Animation zur Schablonenmethode

Sprechertext

„Beim Aufbau einer Fahrattraktion gibt es immer einige Arbeitsschritte, die bei jedem Bau durchgeführt werden. Andere Arbeitsschritte sind für jede Bahn speziell. Für jede Bahn muss zuerst eine Fläche ausgesucht werden, auf der die Bahn gebaut wird. Ist diese gefunden, wird ein Fundament gegossen. Jetzt gibt es die erste Unterscheidung: Es müssen spezielle Schienen oder ein Untergrund ausgewählt werden, auf der die Bahn später fährt. In unserem Beispiel wählen wir Metallschienen. Als nächstes wird immer ein Wärterhaus für die Elektronik gebaut. Dann kommt die nächste Unterscheidung: Es müssen spezielle Wagen ausgewählt werden. In unserem Beispiel wählen wir Wagen mit Metallrädern für die entsprechenden Schienen aus. Zum Schluss wird bei allen Bahnen die Funktionstüchtigkeit überprüft, bevor die Bahn eröffnet werden kann. Dieser Algorithmus zum Aufbau einer Bahn kann durch ein Entwurfsmuster vereinfacht werden. Die gemeinsamen Arbeitsschritte werden in der so genannten Schablonenmethode in der richtigen Reihenfolge festgehalten. Diese Arbeitsschritte werden durch Vererbung an jeden Aufbau einer Bahn weiter gegeben. Die speziellen Arbeitsschritte werden in den jeweiligen unteren Klassen überschrieben.“

9.4 Übung ohne Einbindung von UML-Diagrammen

- Thema:** Schablonenmethode und primitive Operationen beim Aufbau verschiedener Bahnen
- Dauer:** etwa 5 Minuten
- Lernziele:** Unterschiede zwischen Schablonenmethode und primitiven Operationen
- Vorkenntnisse:** Animation Schablonenmethode

Klassendiagramm

```
classDiagram
    class Bahn {
        <<abstrakt>>
        +<<abstrakt>> aufbauen()
        +<<abstrakt>> fahrzeuge()
        +<<abstrakt>> strecke()
    }
    class Golfbahn {
        +<<abstrakt>> aufbauen()
        +<<abstrakt>> fahrzeuge()
        +<<abstrakt>> strecke()
    }
    class Wasserbahn {
        +<<abstrakt>> aufbauen()
        +<<abstrakt>> fahrzeuge()
        +<<abstrakt>> strecke()
    }
    class Wasserrutsche {
        +<<abstrakt>> aufbauen()
        +<<abstrakt>> fahrzeuge()
        +<<abstrakt>> strecke()
    }
    Bahn <|-- Golfbahn
    Bahn <|-- Wasserbahn
    Bahn <|-- Wasserrutsche
```

Aufgabe

Beim Aufbau einer neuen Fahrrastraktion treten oft die gleichen Aufgaben auf. Diese werden in der so genannten Schablonenmethode (hier: `aufbauenBahn()`) implementiert. Unterschiedliche Aufgaben werden separat in den speziellen Klassen in so genannte primitive Operationen geschrieben (hier: `verlegeStrecke()`, `fahrzeuge()`). In dieser Übung soll der Unterschied zwischen Schablonenmethode und den primitiven Operationen erlernt werden. Ordne die Aufgaben der entsprechenden Methode zu!

Hinweis

[A9.4.1] Startbild der Übung ohne UML zur Schablonenmethode

In dieser Übung soll verdeutlicht werden, dass es beim Aufbau von verschiedenen Bahnen im Freizeitpark spezielle Schritte und gemeinsame Schritte gibt. Ein Klassendiagramm soll verdeutlichen, dass die gemeinsamen Schritte in einer Oberklasse zusammengefasst werden könnten. Die Aufgabe in der Übung besteht darin, die Arbeitsschritte entweder der allgemeinen Schablonenmethode zum Aufbau einer Bahn oder einer speziellen Methode der Unterklassen zuzuordnen. Abbildung [A9.4.1] zeigt das Startbild der Übung.

9.5 Übung mit Einbindung von UML-Diagrammen

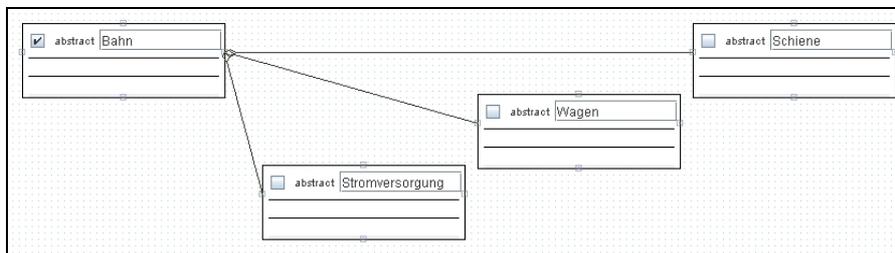
Im Modul Schablonenmethode gibt es keine Übung mit UML zusätzlich zum UML-Puzzle.

9.6 UML-Puzzle

Aufgabenstellung:

„Als Beispiel für das Entwurfsmuster Schablonenmethode hast du den Aufbau einer Fahrtraktion kennen gelernt. Dafür wird ein allgemeiner Bauplan benutzt, mit dessen Hilfe verschiedene Achterbahnen geplant werden können. Es gibt eine Schablonenmethode `baueBahn()`, in der die für alle Bahnen einheitlichen Arbeitsschritte festgelegt werden. Füge diese Methode der Klasse `Bahn` hinzu. Außerdem benötigt man drei Methoden, für den Bau einer Wasserbahn, einer Achterbahn und einer Rutschbahn. Füge auch diese drei Methoden in das Klassendiagramm ein. In unserem Fall soll eine konkrete Rutschbahn mit dem Namen `Fichtenflitzer` gebaut werden. Erstelle diese Klasse und verbinde sie mit der Klasse `Bahn`. In der Klasse `Fichtenflitzer` wird eine entsprechende Methode der Klasse `Bahn` überschrieben. Füge diese Methode hinzu. Die neue Bahn soll nun als Wagen `Sommerrodelbobs` benutzen, als Schienen eine Rutschbahn verwenden und eine mittelstarke Stromversorgung erhalten. Füge auch diese drei Klassen in das Diagramm ein und erstelle sinnvolle Verbindungen zu den anderen Klassen.“

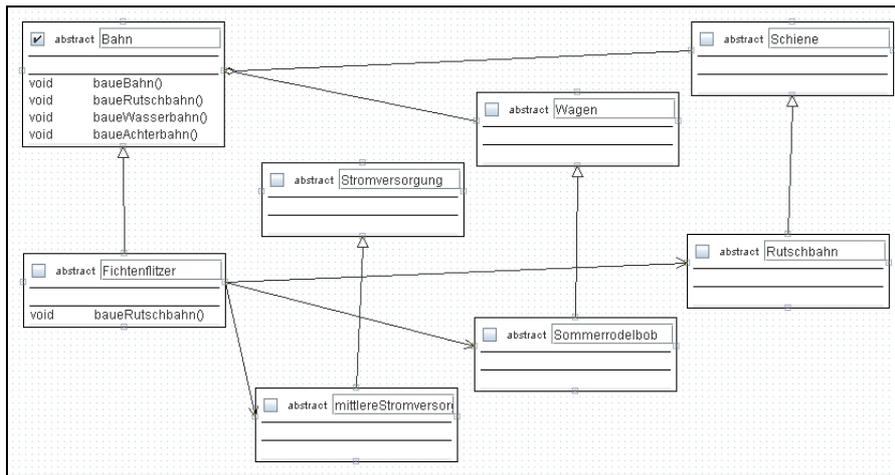
Abbildung [A9.6.1] zeigt die Ausgangssituation des UML-Puzzles zur Schablonenmethode.



[A9.6.1] Ausgangssituation des UML-Puzzles zur Schablonenmethode

Es wird überprüft, ob die Klassen `Bahn`, `Fichtenflitzer`, `MittlereStromversorgung`, `Sommerrodelbob` und `Rutschbahn` vorhanden sind. Falls eine dieser Klassen fehlt oder falls die Klasse `Bahn` nicht als abstrakt gekennzeichnet wurde, wird eine entsprechende Fehlermeldung ausgegeben. Ebenso werden die Vererbungen und die Assoziationen entsprechend der Abbildung [A9.6.2] überprüft. Die Methoden `baueBahn()`, `baueWasserbahn()`, `baueAchterbahn()` (in der Klasse `Bahn`) und `baueRutschbahn()` (in der Klasse `Bahn` und in der Klasse `Fichtenflitzer`) müssen vorhanden sein. Erst wenn alle Kriterien erfüllt wurden, wird die Aufgabe als gelöst gekennzeichnet.

Abbildung [A9.6.2] zeigt die Musterlösung zum UML-Puzzle zur Schablonenmethode.



[A9.6.2] Musterlösung zum UML-Puzzle zur Schablonenmethode

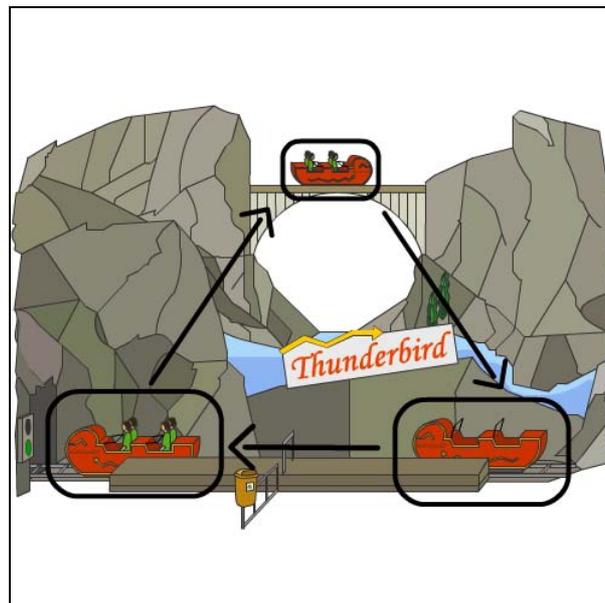
10 Modul Zustand

10.1 Inhalt der Übersichtsseite

Beispiel:

Fahrt einer Achterbahn

Logo:



[A10.1.1] Logo zum Zustand

Beschreibung:

Bei der Fahrt einer Achterbahn werden verschiedene Zustände durchlaufen (z.B. stehend, beschleunigend, bremsend).

Vorteile:

- Gäste können nur dann aus- und einsteigen, wenn sich die Bahn im Zustand „stehend“ befindet.
- Die Bahn kann den Zustand „stehend“ nur dann verlassen, wenn alle Gäste angeschnallt sind.
- Man kann neue Zustände hinzufügen.

Nachteil:

- Wenn es viele Zustände gibt, wird auch der Aufwand sehr groß.

10.2 Glossareintrag

Beschreibung:

Ein Objekt verhält sich unterschiedlich, je nachdem in welchem Zustand es sich befindet. Es gehört zu der Familie der objektbasierten Verhaltensmuster.

Vorteile:

- Klienten müssen nicht wissen, in welchem Zustand sich das Objekt befindet.
- Das Hinzufügen von neuen Zuständen wird vereinfacht.

Nachteil:

- Die Anzahl der Klassen wird sehr groß, wenn es viele Zustände gibt.

Zustand

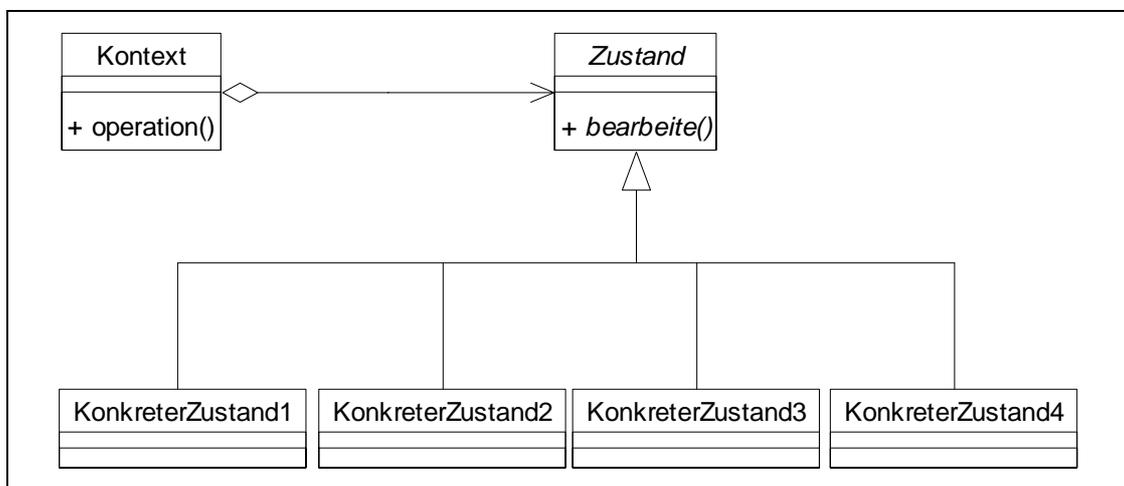
`Zustand` definiert eine Schnittstelle zur Kapselung des mit einem bestimmten Zustand des Kontextobjekts verbundenen Verhaltens, vgl. [A10.2.1].

KonkreterZustand

Jede Unterklasse implementiert ein Verhalten, das mit einem Zustand des Kontextobjekts verbunden ist, vgl. [A10.2.1].

Kontext

Die Klasse `Kontext` definiert eine Schnittstelle, die `Kontext` interessiert. Sie verwaltet ein Exemplar einer Klasse `KonkreterZustand`, die den aktuellen Zustand definiert, vgl. [A10.2.1].



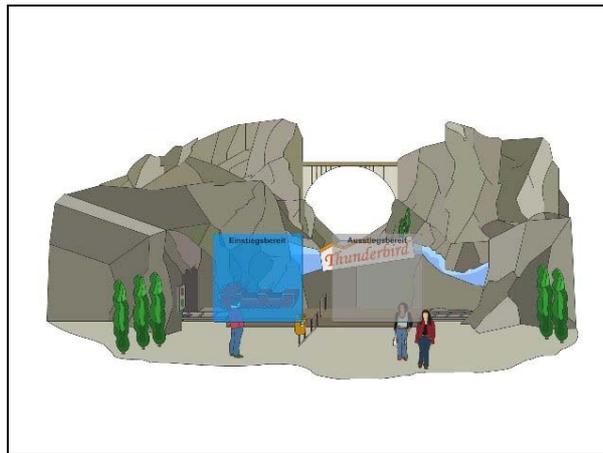
[A10.2.1] UML-Klassendiagramm des Entwurfsmusters Zustand

10.3 Animation

Metadaten

Thema: Die verschiedenen Zustände während einer Fahrt und das Auslagern dieser Zustände in separate Klassen

Dauer: 1:20 Minuten



[A10.3.1] Screenshot der Animation zum Zustand

Sprechertext

„Während einer Fahrt mit der Achterbahn Thunderbird ändert die Bahn mehrmals ihren Zustand. Zum Beispiel wechselt die Bahn vom Zustand Ausstiegsbereit in den Zustand Einstiegsbereit, indem der Wagen ein Stück vorrollt. Um nicht für jeden Zustand ein eigenes Objekt der Achterbahn erstellen zu müssen, wird die Behandlung der Zustände in separate Klassen ausgelagert. Für jeden Zustand der Achterbahn wird eine eigene Klasse gebildet, zum Beispiel Einstiegsbereit, Fahrend oder Bremsend. Um den Zugriff vom Objekt der Achterbahn auf seine Zustandsobjekte zu vereinfachen, wird eine Oberklasse für die erstellten Zustandsklassen gebildet. Diese Oberklasse vererbt ihre Beziehung zur Achterbahnklasse an die einzelnen Zustandsklassen. Das Achterbahnobjekt tauscht nun während einer Fahrt sein zugehöriges Zustandsobjekt bei einem Zustandswechsel aus. Durch die Verwendung des Entwurfsmusters Zustand ist es nun möglich, neue Zustände hinzuzufügen oder die Achterbahn zu tauschen.“

10.4 Übung ohne Einbindung von UML-Diagrammen

Metadaten

Thema:	Steuerung einer Bahn mit Regeln für die Zustandsübergänge
Dauer:	etwa 15 Minuten
Lernziele:	Verbesserung des Verständnisses für Zustände, Zustandsübergänge und bedingte Anweisungen
Vorkenntnisse:	Grundkenntnisse für bedingte Anweisungen Kenntnis der Operatoren <, >, ==, <= und >= Animation Zustand

In diesem Lernmodul soll mit Hilfe von bedingten Anweisungen ein Programm zur Steuerung einer Achterbahn realisiert werden. Per Pseudo-Code werden die Zustandsübergänge beschrieben. In der Aufgabenstellung wird die Funktionsweise der Bahn kurz erklärt und auf die Randbedingungen hingewiesen. Es gibt 3 Zustände, zwischen denen die Bahn wechseln kann, dabei werden die Regeln für die Zustandsübergänge durch die Faktoren Bahnabschnitt, Geschwindigkeit sowie Verwendung der Operatoren <, >, ==, <= und >= festgelegt. Diese sind so auszuwählen, dass die Bahn eine Runde auf der Bahnstrecke fährt und dabei weder stehen bleibt, noch die erlaubten Höchstgeschwindigkeiten überschreitet. Pro Bahnabschnitt kann immer nur ein Zustand angenommen werden. Am Ende soll sich die Bahn wieder am Einstiegspunkt und in dem Ausgangszustand befinden. Es besteht jederzeit die Möglichkeit, das programmierte Verhalten der Bahn zu überprüfen. Bei einer Überprüfung werden neben den Bahneigenschaften Abschnitt, Zustand und Geschwindigkeit auch Textausgaben generiert, um auf Unstimmigkeiten in den Regeln hinzuweisen. Außerdem sind in einer Realsicht die Fahrt der Bahn und der erreichte Endzustand zu sehen. Die Anzahl der verwendeten Regeln kann bis zu maximal sieben Regeln je Zustand gewählt werden. Wenn es nicht ausgefüllte Regeln für die Zustandsübergänge gibt, so werden diese ignoriert. Abbildung [A10.4.1] zeigt eine einfache Musterlösung. In Abbildung [A10.4.2] ist ein Screenshot der Übung zu sehen.

Bahzustand Nr. 1 Beschleunigen			
IF(Abschnitt	< 2	&& Speed < 20)THEN DO Zustand = 3
Bahzustand Nr. 2 Bremsen			
IF(Abschnitt	== 1	&& Speed == 0)THEN DO Zustand = 3
Bahzustand Nr. 2 Bremsen			
IF(Abschnitt	== 1	&& Speed == 0)THEN DO Zustand = 1
IF(Abschnitt	== 1	&& Speed == 10)THEN DO Zustand = 2
IF(Abschnitt	<= 12	&& Speed <= 10)THEN DO Zustand = 3

[A10.4.1] Musterlösung der Übung ohne UML zum Zustand

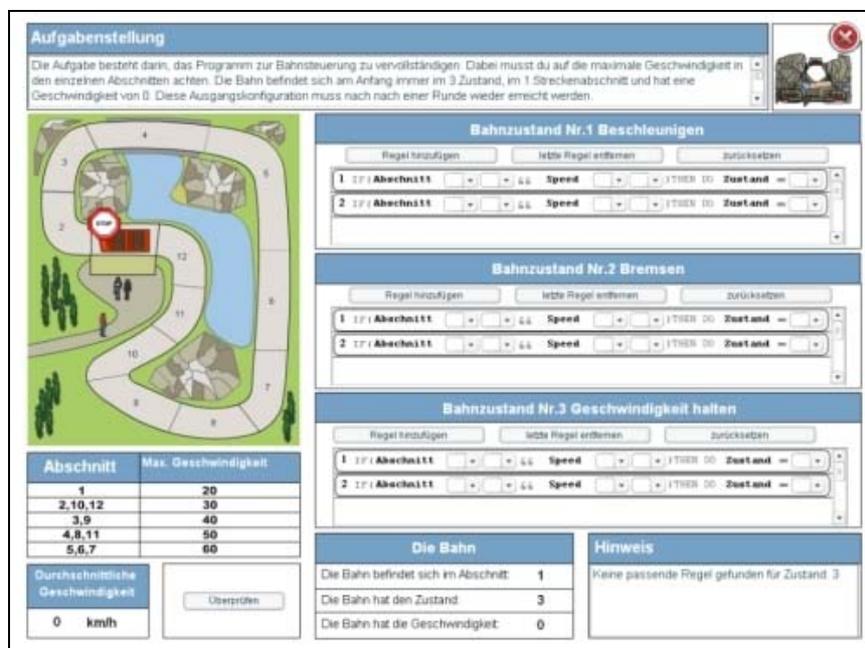


Abbildung [A10.4.2] Screenshot der Übung ohne UML zum Zustand

10.5 Übung mit Einbindung von UML-Diagrammen

Metadaten

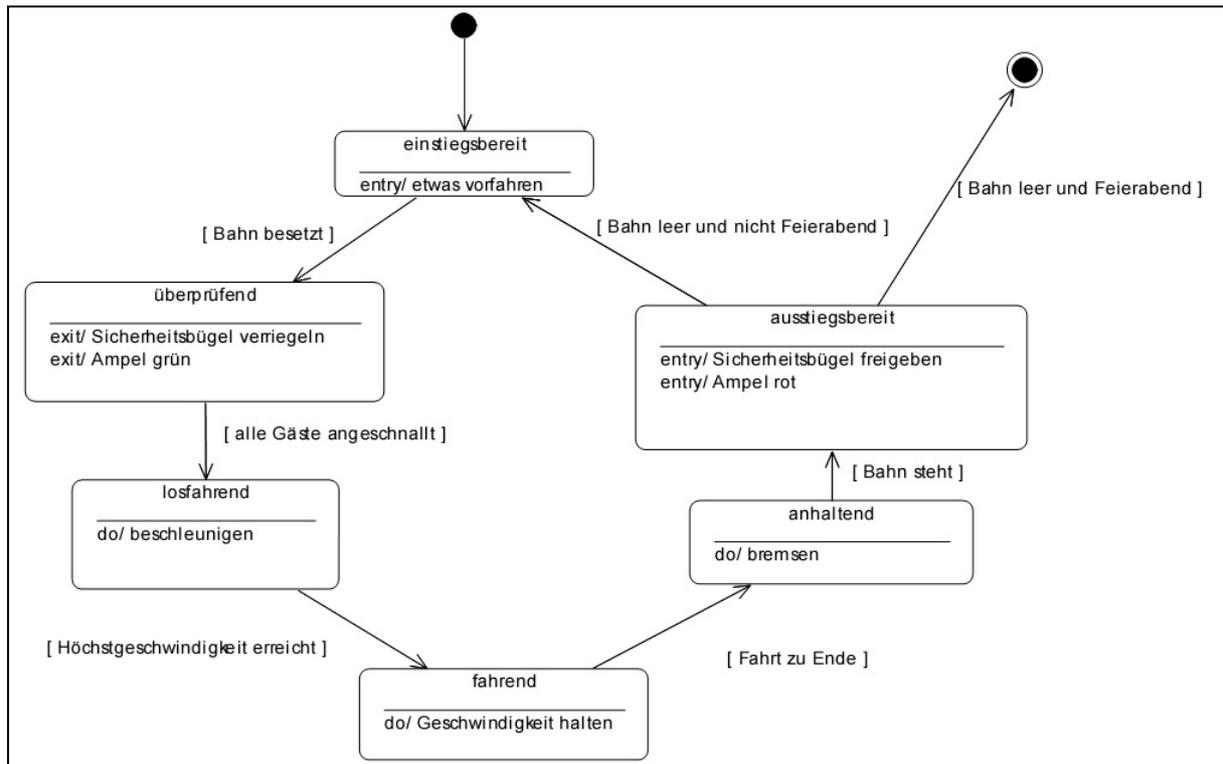
- Thema: Die Bahnfahrt des Thunderbirds als Zustandsdiagramm
1. Vervollständige das Zustandsdiagramm
 2. Steuere die Bahn dem Zustandsdiagramm entsprechend
- Dauer: etwa 15 Minuten
- Lernziele: Verbesserung des Verständnisses für Zustandsdiagramme
Aktionen, Bedingungen und Zustände

Vorkenntnisse: Grundlagen Zustandsdiagramm

Lerngegenstand dieser Übung ist ein Zustandsdiagramm, mit dem die Fahrt mit Thunderbird modelliert werden soll. Der Ablauf einer solchen Fahrt wird in der Aufgabenstellung beschrieben. Im ersten Teil geht es darum, das Grundgerüst eines Zustandsdiagramms so zu vervollständigen, dass es dem vorgegebenen Ablauf entspricht. Die richtigen Namen und Aktionen der Zustände sowie die Bedingungen für die Zustandsübergänge sind aus einer Liste auszuwählen. Bei einer falschen Auswahl gibt es eine Rückmeldung im Hinweisfeld und – sofern sinnvoll möglich – eine Darstellung in der Realsicht, die den Fehler veranschaulicht. [A10.5.1] zeigt einen Screenshot dieser Übung.

Wenn das Zustandsdiagramm korrekt erstellt wurde, gelangt man zum zweiten Teil der Übung. Hierbei besteht die Aufgabe darin, die Bahn dem Diagramm entsprechend zu steuern, also die richtigen Aktionen auszuführen und zu entscheiden, wann die Bahn in welchen Zustand wechselt. [A10.5.2] zeigt das korrekte Zustandsdiagramm dieser Übung.

[A10.5.1] Screenshot zur Übung mit UML zum Zustand

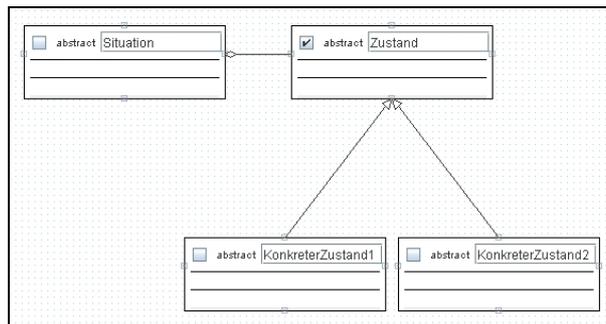


[A10.5.2] Korrektes Zustandsdiagramm der Übung mit UML zum Zustand

10.6 UML-Puzzle Zustand

Aufgabenstellung:

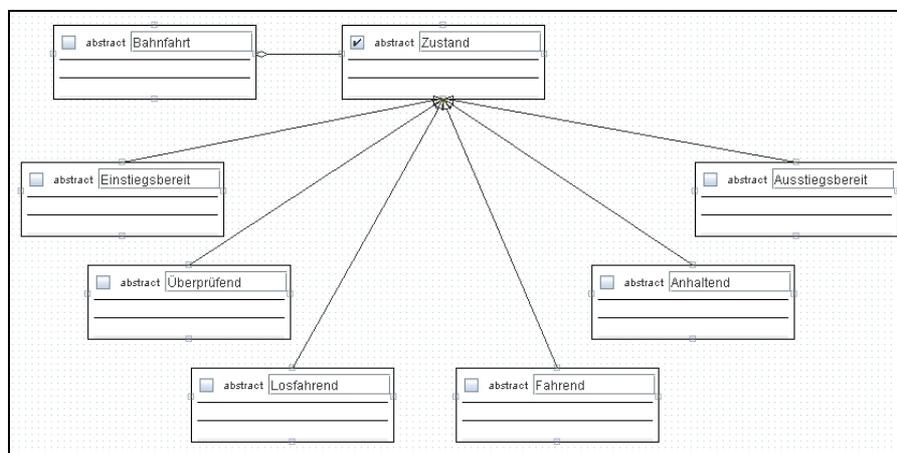
Zu sehen ist ein allgemeines Klassendiagramm zum Zustandsmuster. Dies soll an das konkrete Beispiel einer Bahnfahrt im Pattern Park angepasst werden. Benenne die Situation und die konkreten Zustände neu. Füge zusätzlich neue konkrete Zustände einer Bahnfahrt hinzu. Benutze dabei die Namen der Zustände aus der Übung mit dem Zustandsdiagramm. Abbildung [A10.6.1] zeigt die Ausgangssituation des UML-Puzzles zum Zustand.



[A10.6.1] Ausgangssituation des UML-Puzzles zum Zustand

Beschreibung des Algorithmus zur Überprüfung der Lösung

Es wird überprüft, ob die konkreten Zustands-Klassen umbenannt sind, und die fehlenden Klassen hinzugefügt werden. Fehlt eine der Klassen, bzw. entspricht die Benennung nicht der der vorangegangenen Übung, wird ein entsprechender Fehlertext bei der Überprüfung ausgegeben. Ebenso werden die Vererbungen auf Existenz und korrekte Anordnung überprüft. Erst wenn alle Kriterien erfüllt werden, wird die Aufgabe als gelöst gekennzeichnet. Alles was darüber hinaus noch hinzugefügt wird, wird bei der Überprüfung nicht berücksichtigt. Abbildung [A10.6.2] zeigt die Musterlösung zum UML-Puzzle zum Zustand.



[A10.6.2] Musterlösung zum UML-Puzzle zum Zustand

11 Technische Umsetzung

In der Entwurfsphase wurden verschiedene Konzepte zur technischen Umsetzung der Lernsoftware untersucht und bezüglich ihrer Eignung für das Projekt bewertet.

Es wurden 3 wesentliche Konzepte betrachtet:

1. Implementierung in Java SDK 1.5

Bewertung: Java SDK 1.5 eignet sich hinsichtlich des Funktionsumfangs und der einfach zu realisierenden Datenhaltung. Jedoch können grafisch aufwendig gestaltete Animationssequenzen nur mit sehr großem Aufwand und nicht optimalem Ergebnis erstellt werden. Zusätzlich entstehen durch die Verwendung von Java SDK 1.5 für zukünftige Erweiterungen keine weiteren Kosten, da Java frei verfügbar ist.

2. Implementierung in Adobe Flash 8 / Adobe Actionscript 2.0

Bewertung: Die Implementierung der Lernsoftware in Flash 8 und Actionscript 2.0 liefert das grafisch optimale Ergebnis (Animation, Oberfläche etc.). Durch konsequente Weiterentwicklung des Flash-Konzepts zu einer auf das Web spezialisierten Lernsoftware und dem von Adobe zugrunde gelegtem Sandkastenprinzip, kann Flash 8 keine Wahl für die Umsetzung des Frameworks sein. Flash 8 bzw. das eingebundene Actionscript erlauben keine Speicherung von Daten in einem vordefinierten Ordner des Verzeichnisbaums auf dem Clientrechner. Zusätzlich kann nur eine kleine Datenmenge temporär als sog. Cookies (ähnlich der Programmierung in der Skriptsprache PHP) im Ordner des Flash-Stand-Alone-Players vorgehalten werden.

Da beide Varianten ihre klaren Vorzüge haben, wurde entschieden, grafisch aufwendige Elemente wie Animationen, Übungssequenzen mit Realweltansicht in Flash 8 zu implementieren, das Framework inkl. Datenhaltung und das UML-Puzzle in Java SDK 1.5 umzusetzen. Da es zur Kommunikation zwischen dem in C++ geschriebenen Flash-Stand-Alone-Players und dem Java-Framework nur proprietäre Schnittstellenlösungen gibt, wurde die Ansteuerung der Flash-Filme und -Anwendungen durch die Konvertierung in das durch Flash bereitgestellte Projektor-Format realisiert. Das Framework startet die ausführbaren Dateien für die Betriebssysteme Windows und Linux mit einem neuen Prozess, vgl. [A11.1].

```

/**
 * Method to Start the Flashmovie
 * Method is static
 * @param name ist the name of the flash file
 */
public static void StartFlash(String name) {
    if (isWin()) { //Implementation for Windows Systems

        try {
            Runtime r = Runtime.getRuntime();

            String filetoexe = System.getProperty("user.dir")
                + System.getProperty("file.separator") + name;
            Process pr = r.exec(filetoexe);
        } catch (Exception e) {
            System.out.println(e.toString());
        }
    }
    if (isMac()) { //Implementation for Mac OS Systems
        try {
            String filetoexe = "file://" + System.getProperty("user.dir")
                + System.getProperty("file.separator") + name;
            LinksFrame lf = new LinksFrame();
            lf.startBrowser(filetoexe);
        } catch (Exception e) {
            System.out.println(e.toString());
        }
    }
    if (isLin()) { //Implementation for Linux Systems
        try {
            Runtime myRuntime = Runtime.getRuntime();

            String flashplayer = System.getProperty("user.dir") +
                System.getProperty("file.separator") + "data/flashplayer/flashplayer";

            String film = System.getProperty("user.dir")
                + System.getProperty("file.separator") + name;

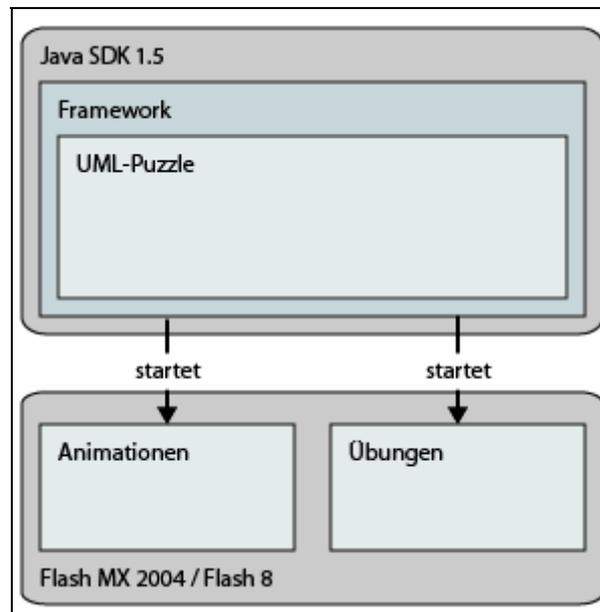
            String playcommando = flashplayer + " " + film;
            Process playProcess = myRuntime.exec(playcommando);
        } catch (Exception e) {
            System.out.println(e.toString());
        }
    }
}
    }
}

```

[A11.1]: Aufruf der Flash Projektor Datei

Für Betriebssysteme Mac OS X wird über den Browser eine HTML-Seite aufgerufen, in der jeweilige Flash-Film im *.swf-Format integriert ist. Für die Unix/Linux-Betriebssystemfamilie wird direkt eine *.swf-Datei gestartet. Für diese beiden Varianten ist der Stand-Alone-Player von Adobe beziehungsweise ein entsprechendes Browser Plug-In erforderlich.

Der in Abbildung A2.11 dargestellte Systemaufbau zeigt die Einbettung des UML-Puzzles in das Framework.



[A11.2]: Systemaufbau der Lernsoftware

Für das UML-Puzzle wurde neben Standardbibliotheken des Java SDK 1.5 die Bibliothek JGraph¹ unter der Version 5.9.2.1 verwendet, welche es erlaubt, UML-Diagramme grafisch ansprechend darzustellen und auf Benutzeraktionen im Fensterbereich zu reagieren. So ist es möglich, UML-Klassen per „Drag-and-Drop“ zu verschieben, ohne dabei Assoziationen zwischen Klassen zu verlieren oder zu überdecken. Die Programmbibliothek JGraph steht unter GNU GPL 2.1. Die Speicherung von UML-Diagrammen erfolgt in einem eigenen XML-Dialekt. Er enthält neben den zu einem gezeichneten Objekt notwendigen Koordinatendaten, die Eingaben des Benutzers, wie etwa Klassenname, Attribute etc. Als Schnittstelle zwischen XML und dem UML-Puzzle wurde die frei verfügbare Bibliothek JDOM² verwendet. Diese kommt auch bei der Speicherung von Benutzereinstellungen innerhalb des Frameworks zum Einsatz. JDOM steht unter Apache-style open source license.

Für die Realisierung der interaktiven Parkkarte wurden auf bereits bestehende Implementierungen von Yura Mamyry³ zurückgegriffen, die eine Image-Map in Java unter dem Arbeitstitel Yura Button Panel 1.0 umsetzen. Diese Programmbibliothek wurde im Einverständnis mit dem Autor für die Bedürfnisse des Pattern Park erweitert und weiterentwickelt. Die Programmbibliothek steht unter GNU GPL.

¹ JGraph ist eine frei verfügbare Java-Bibliothek der Firma JGraph Ltd. (<http://www.jgraph.com>) .

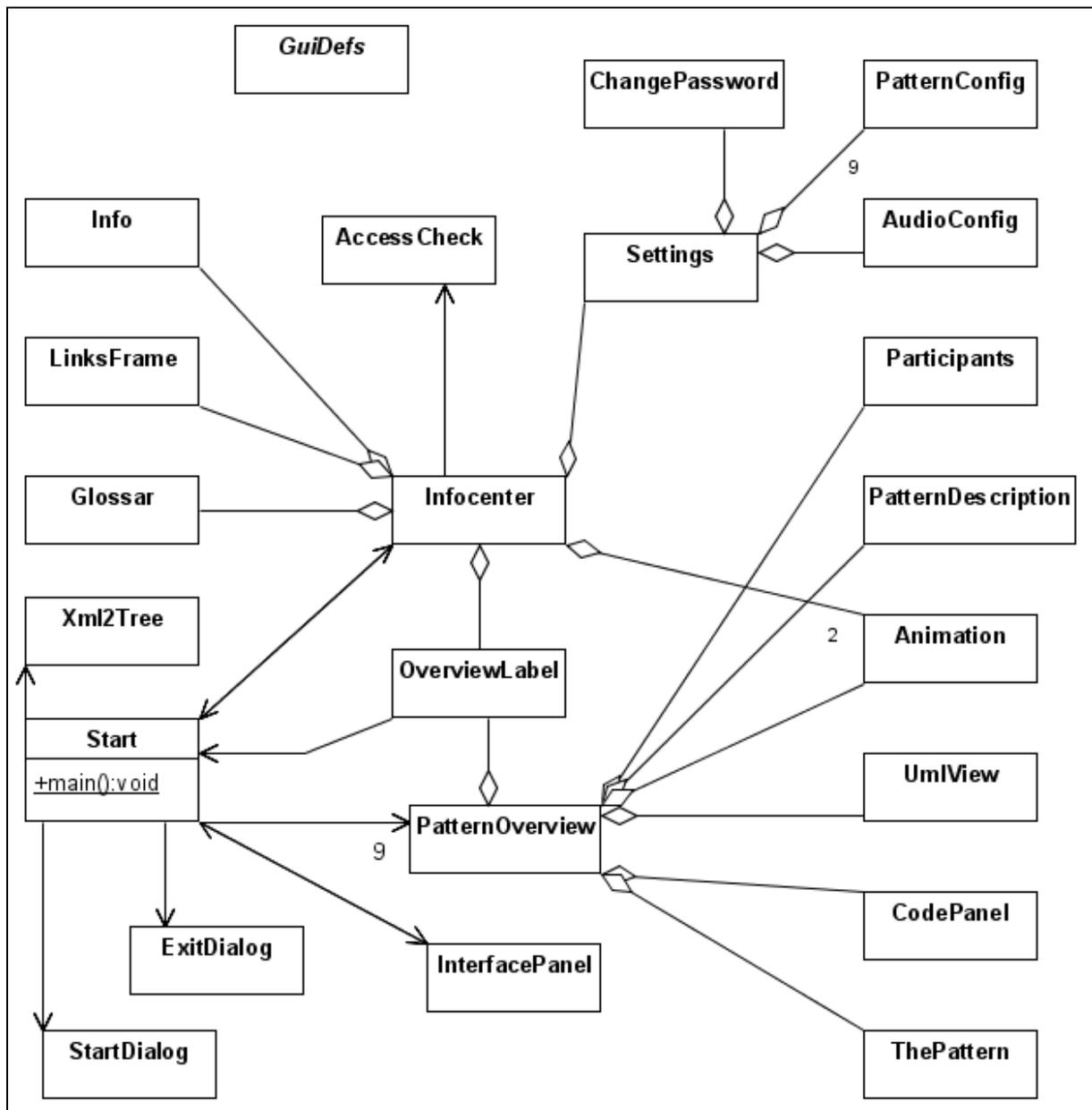
² <http://www.jdom.org>

³ yura@yura.net

Abbildung [A11.3] zeigt die Klassenstruktur des Frameworks. Die `main()`-Methode befindet sich in der Klasse `Start`. Beim Starten des Programms werden die XML-Dokumente `halt.xml` und `config.xml` von der Klasse `Xml2Tree` geladen und als JDOM-document Objekte an `Start` übergeben. Die Form dieser beiden XML-Dokumente wird in den entsprechenden DTD-Dateien beschrieben. Die Parkkarte als Benutzungsoberfläche wird durch die Klasse `InterfacePanel` realisiert. Nach dem Start des Programms wird das Dialogfenster der Klasse `StartDialog` eingeblendet. Vor dem Beenden des Programms erscheint der Dialog der Klasse `ExitDialog`.

Von der Parkkarte aus sind die Übersichtsseiten zu den acht Entwurfsmustern und der Musterkombinationsaufgabe erreichbar. Diese werden in der Klasse `PatternOverview` beschrieben und bestehen aus den JPanel-Klassen `PatternDescription` (Übersicht), `Animation` (Animationsfilm), `Participants` (Übung ohne UML), `UmlView` (Übungen mit UML), `ThePattern` (Das Muster) und `CodePanel` (Code) sowie einer Klasse `OverviewLabel`, die die obere Menüleiste und die Schaltfläche enthält, über den man zur Parkkarte zurückgelangt. Von dort aus kann man auch zum Infocenter gelangen, das durch die gleichnamige Klasse beschrieben wird und ähnlich wie die Übersichtsseiten zu den Entwurfsmustern aus den JPanel-Klassen `Animation` (Animationsfilme „Einführung in den Pattern Park“ und „Einführung in Entwurfsmuster“), `Glossar`, `LinksFrame`, `Info`, `Settings` und `AccessCheck` besteht. Diese letzte Klasse enthält das Dialogfenster mit der Passwortabfrage, das eingeblendet wird, wenn man auf die Einstellungen zugreifen will. Diese Einstellungsseite wiederum besteht aus neun Instanzen der Klasse `PatternConfig` (für jedes Entwurfsmuster eine), sowie einer Instanz der Klasse `AudioConfig` und einer Instanz der JDialog-Klasse `ChangePassword`.

In der abstrakten Klasse `GUIDefs` werden Pfade zu den verwendeten Grafiken und die Größen der grafischen Elemente in Klassenattributen gespeichert.



[A11.3]: Klassenstruktur des Frameworks

12. Zusammenfassung und Ausblick

Die im Rahmen der Projektgruppe erstellte Lernsoftware Pattern Park eignet sich für den Einsatz im Informatikunterricht. Dazu sind Blended-Learning-Sequenzen denkbar, bei der die einzelnen Lernmodule des Pattern Park von Schülern bearbeitet werden. Einzelne Module, wie etwa die Übung ohne UML zum Entwurfsmuster Kompositum können auch im Informatikeingangunterricht unterstützend verwendet werden, um im konkreten Beispiel die Baumstruktur als Datenstruktur zu veranschaulichen. Durch den frei wählbaren Lernweg innerhalb des Pattern Park können Lernszenarien auf unterrichtsspezifische Anforderungen angepasst werden.

Während der Implementierung der Lernsoftware wurde großer Wert auf eine anschauliche Darstellung der in den Entwurfsmustern innewohnenden informatischen Konzepte/ fundamentalen Ideen gelegt. Resultierend hieraus wurde eine Lösung entwickelt, welche sowohl aus einer Java-Applikation, als auch aus Flash-Elementen besteht. Im Verlauf der Implementierung erwies sich diese Umsetzung als Zielführendste, zeigte zugleich aber auch Grenzen der Interoperabilität zwischen Java und Flash auf, da eine vollständige Einbettung von Flash-Elementen in die Java-Applikation ohne Erwerb einer proprietären Programmbibliothek nicht möglich ist. In beiden Programmteilen wurde darauf geachtet, dass eine zukünftige Erweiterung der Lernsoftware möglich ist. Im folgenden möchten wir näher auf die beispielhaften Erweiterungsmöglichkeiten eingehen.

Im Java-Framework könnte zukünftig eine Benutzerverwaltung eingebunden werden, um die Lernsoftware im Schulalltag zu personalisieren. Gerade bei der Installation auf einem Mehrbenutzerarbeitsplatz ist diese Option wünschenswert, um Lernwege der Lernenden individuell nachvollziehen zu können und Konfigurationen der Lernsoftware Pattern Park für bestimmte Schülergruppen zu laden.

Die Lernsoftware kann zudem, durch die konsequente Auslagerung von verwendeten Textbausteinen in XML-Dateien, problemlos in beliebige Sprachen übersetzt werden. Hierzu müssen lediglich die entsprechenden XML-Dateien⁴ für die einzelnen Übungen ersetzt werden. Zu beachten ist hierbei, dass der Fließtext nicht wesentlich mehr Buchstaben als die bisherige Sprachversion besitzt, da in den jeweiligen Animationen und Übungen der zur Verfügung stehende Platzhalter für Textfelder und Beschriftungen begrenzt ist. In diesem Zu-

⁴ data/data_o_uml/*.xml, data/data_m_uml/*.xml, data/data_animation/*.xml, data/config/inhalt.xml, umIPuzzle/umlData/textGerman.xml

sammenhang wäre es auch sinnvoll, die verwendeten Audiospuren für die Animationen neu zu erstellen, oder auf Audiospuren generell zu verzichten.

Ist es erforderlich, dass neue Entwurfsmuster der Lernsoftware hinzugefügt werden, muss die interaktive Parkkarte, welche in Flash gezeichnet wurde, entsprechend erweitert werden und im Folgenden die Imagemap neu generiert werden. Es sei hier darauf hingewiesen, dass die Parkkarte auf Grund der dargestellten Motive und aus gestalterischen Gesichtspunkten nicht beliebig Platz für die Aufnahme weiterer Entwurfsmuster bietet. Übersichtsseiten und nachfolgende Übungssequenzen für neue Entwurfsmuster lassen sich jedoch einfach in die vorgegebene Inhaltsstruktur der Lernsoftware einbinden.

Das Übungselement UML-Puzzle kann aus unserer Sicht wie folgt erweitert werden:

- Bei Attributen und Operationen könnte ein DropDown-Menü eingefügt werden, das nur sinnvolle Eingaben wie int, String, <Klassenname>, void, etc. enthalten könnte.
- Wenn ein Attribut als Referenz auf eine andere Klasse hinzugefügt wurde, könnte diese beim nächsten Neustart als entsprechende Assoziation angezeigt werden und nicht mehr als Attribut.
- Sichtbarkeiten für Klassen könnten hinzugefügt werden, sowie für deren Attribute und Operationen (z.B. als Extraspalte mit DropDown-Menü). Für die jetzigen Anforderungen an das UML-Puzzle waren Sichtbarkeiten nicht notwendig und wurden deshalb auch bisher nicht realisiert.
- Aus dem bearbeiteten UML-Diagramm könnte automatisch Quellcode generiert werden, zum Beispiel in Java.
- Neue Menüpunkte „Speichern unter“ und „Neu“ könnten eine neue, leere Aufgabe erzeugen bzw. speichern.
- Das Hilfe-Textfeld könnte noch mehr genutzt werden, um dem Benutzer eine noch deutlichere Unterstützung zu geben.

Literaturverzeichnis

- [ABCM98] Astrachan, Owen. Berry, Geoffrey. Cox, Landon. Mitchener, Garrett (1998) Design patterns: an essential component of CS curricula. Proceedings of the 29th SIGCSE technical symposium on Computer science education, 2000. (<http://www.cs.duke.edu/~ola/patterns/talks/sigcse98.pdf>)
- [Brin03] Brinda, Torsten (2003) Didaktisches System für objektorientiertes Modellieren im Informatikunterricht der Sekundarstufe II. Dissertation. Universität Siegen (<http://www.die.informatik.uni-siegen.de/forschung/brinda.pdf>)
- [Bu96] Buschmann, Frank. Meunier, Regine. et al. (1996) A System of Patterns. John Wiley & Sons. Chichester
- [CalHir] Callaghan, Michael. Hirschmüller, Heiko. 3-D Visualisation of Design Patterns and Java Programs in Computer Science Education. De Montfort University
- [Christ] Christensen, Henrik Bærbak. Frameworks. Putting Design Patterns into Perspective. University of Aarhus.
- [DGR97] Duell, Michael. Goodsen, John. Rising, Linda (1997) Non-software examples of software design patterns. Conference on Object Oriented Programming Systems Languages and Applications. Object Magazine, Vol. 7, No. 5, July 1997, pp. 52-57. (siehe auch <http://www2.ing.puc.cl/~jnavon/IIC2142/patexamples.htm>)
- [Duden06] Claus, Volker. Schwill, Andreas (2006) Duden Informatik. Dudenverlag. 4.Auflage. Mannheim
- [FFS04] Freeman, Eric. Freeman, Elisabeth. Sierra, Kathy (2004) Head First Design Patterns. Your brain on design patterns. O'Reilly Media. Peking (Probekapitel online: <http://www.oreilly.com/catalog/hfdesignpat/chapter/ch03.pdf>)
- [GHJV95] Gamma, Erich et al. (1995) Design Patterns. Elements of Reusable Object-Oriented Software. Addison-Wesley. Massachusetts (deutsche Version: Gamma, Erich et al. (1996) Entwurfsmuster. Elemente wieder verwertbarer objektorientierter Software. 1.Auflage. Addison-Wesley. München)
- [Grand] Grand, Mark. Overview of Design Patterns. (http://www.mindspring.com/~mgrand/pattern_synopses.htm)
- [Harr02] Harrer, Andreas (2002) Muster in der Software-Technik. Vorlesungs-Skript. Technische Universität München (<http://atbruegge27.informatik.tu-muenchen.de/teaching/ss02/muster/skriptDruckversion.pdf>)
- [HaSc02] Harrer, Andreas und Schneider, Markus (2002) Didaktische Betrachtungen zur Unterrichtung von Software-Mustern im Hochschulbereich In: Schubert et al. (2002) Forschungsbeiträge zur Didaktik der Informatik. Theorie, Praxis, Evaluation. Tagungsband. Witten-Bommerholz (http://www.die.informatik.uni-siegen.de/DIE_BIB/proceedings/gi-Ini-22.pdf)

- [Hoff03] Hoffmann, Andreas (2003) Theoretisch begründete Vorgehensweise bei der Entwicklung von Software zur Exploration im Bildungskontext. Diplomarbeit. Universität Siegen
(<http://www.die.informatik.uni-siegen.de/forschung/Hoffmann/Diplomarbeit-Hoffmann.pdf>)
- [Hust] Huston, Vince. Design Patterns.
(<http://home.earthlink.net/~huston2/dp/patterns.html>)
- [Kelt02] Kelter, Udo (2002) Softwaretechnik I. Vorlesungsskript. Universität Siegen
(<http://pi.informatik.uni-siegen.de/kelter/lehre/02w/stl02w/skripten.html>)
- [Kuch04] Kuchen, Herbert (2004) Entwurfsmuster. Auszug aus Vorlesungsskript Software Engineering II. Universität Münster
(<http://danae.uni-muenster.de/lehre/kuchen/SS04/SE2k2b.pdf>)
- [LaBo01] Lang, Joseph E. und Bogovich, Brian R. (2001) Object-Oriented Programming and Design Pattern. SIGCSE Bulletin Vol. 33 No. 4, ACM Press. University of Dayton.
- [LEO] Endbericht der Projektgruppe LEO (2001), Universität Dortmund
(<http://www.die.informatik.uni-siegen.de/pgleo/download/endbericht.zip>)
- [Netz02] Netzer, Cajus (2002) Entwurfsmuster im Informatikunterricht an allgemein bildenden Schulen. Projektarbeit. Universität Dortmund.
- [Niere] Niere, Jörg. Einführung Programmierpraktikum. Softwareentwicklung im 21 Jahrhundert. Vorlesungsfolien. Universität Siegen Fachgruppe Praktische Informatik. Auf Seite 9 Informationen zum Singleton-Muster zu finden.
- [Nik98] Nikander, Pekka (1998) Gang-of-Four Design patterns as UML models.
(<http://www.tml.tkk.fi/~pnr/GoF-models/html/>)
- [Przy02] Przygienda, Andreas (2002) Eine Architektur für Explorationsbausteine für objektorientiertes Modellieren im Informatikunterricht. Diplomarbeit. Universität Zürich
(http://www.ifi.unizh.ch/ifiadmin/staff/rofrei/DA/DA_Arbeiten_2002/Przygienda_Andreas.pdf)
- [QuiCir98] Quibeldey-Cirkel, Klaus (1998) Lehr-Erfahrung vermittelt durch Lehr-Muster. Ein Beitrag zur Didaktik der Informatik. Informatik und Ausbildung. Seite 33-42
(<http://www.informatik.uni-stuttgart.de/fakultaet/ausbildung98/Tagungsband/Quibeldey-Cirkel/beitrag.doc>)
- [QuiCir99] Quibeldey-Cirkel, Klaus (1999) Entwurfsmuster. Design Patterns in der objektorientierten Softwaretechnik. Springer. Berlin.
- [Roess00] Rössling, Guido (2000) ANIMAL User Guide
(<http://www.animal.ahrgr.de/usingAnimalVisually.pdf>)
- [Schoe02] Schönwälder, Jürgen (2002) Informatik C, Vorlesungsskript Wintersemester 2001/2002. FB Mathematik, Informatik. Universität Osnabrück
<http://www.vorlesungen.uos.de/informatik/c01/infc.ps.gz>
- [ScSc04] Schubert, Sigrid und Schwill, Andreas (2004) Didaktik der Informatik. 1.Auflage. Spektrum Akademischer Verlag. Heidelberg

- [Schu03] Schulte, Carsten (2003) Lehr-Lernprozesse im Informatik-Anfangsunterricht. Theoriegeleitete Entwicklung und Evaluation eines Unterrichtskonzepts zur Objektorientierung in der Sekundarstufe II. Dissertation. Universität Paderborn (<http://ubdata.uni-paderborn.de/ediss/17/2003/schulte/disserta.pdf>)
- [Schwill93] Schwill, Andreas (1993) Fundamentale Ideen der Informatik. (<http://www.informatikdidaktik.de/Forschung/Schriften/ZDM.pdf>)
- [ShTr03] Shalloway, Alan. Trott, James R. (2003) Entwurfsmuster verstehen. Eine neue Perspektive auf objektorientierte Software-Entwicklung. mitp-Verlag. Bonn
- [Stech06a] Stechert, Peer: Informatics System Comprehension - A learner-centred cognitive approach to networked thinking. In: Education and Information Technologies 11 (2006) 3, Springer Netherlands, Oktober 2006.
- [Stech06b] Stechert, Peer: Informatics System Comprehension - A learner-centred cognitive approach to networked thinking. In IFIP TC3 / WG 3.1, WG 3.3 & WG 3.5 Joint Conference "Imagining the future for ICT and Education". 26th-30th June 2006, Alesund, Norway, 2006.
- [Stech06c] Stechert, Peer: Unterrichtsmodellentwicklung zur Förderung des Informatiksystemverständnisses mit Entwurfsmustern. In: Lecture Notes in Informatics: Schwill, A.; Schulte, C.; Thomas, M. (Hrsg.), 3. Workshop der GI-Fachgruppe Didaktik der Informatik, 19.-20.06.2006 an der Universität Potsdam, 2006. (http://www.die.informatik.uni-siegen.de/gruppe/stechert/publikationen/stechert_potsdam_2006.pdf)
- [Stech07] Stechert, Peer: Understanding of Informatics Systems - A theoretical framework implying levels of competence. In: A. Berglund & M. Wiggberg (Eds.) Proceedings of the 6th Baltic Sea Conference on Computing Education Research, Koli Calling. Finland 2006. Uppsala University, Uppsala, Sweden. 2007
- [StSch07] Sigrid Schubert, Peer Stechert: A Strategy to Structure the Learning Process Towards Understanding of Informatic Systems. Accepted for publication at Joint IFIP-Conference Informatics, Mathematics and ICT (IMICT2007): A golden triangle. Boston. USA. 27th–29th June 2007.
- [StuFlo04] Stuurman, Sylvia. Florijn, Gert (2004) Experiences with teaching design patterns. The 9th Annual Conference on Innovation and Technology in Computer Science Education (ITICSE), 2004, Leeds, UK. (<http://www.ou.nl/open/stm/publikaties/p055-stuurman.pdf>)
- [Ulle05] Ullenboom, Christian (2005) Vermittlung von Entwurfsmustern in der informatischen Ausbildung am Beispiel eines Media Players. Diplomarbeit. Universität Paderborn
- [Unger00] Unger-Lamprecht, Barbara (2000) Experimentelle Bewertung der Auswirkungen von Entwurfsmustern. Dissertation. Universität Karlsruhe (http://www.ktw.at/KTWDE/Download/Unger_Lamprecht.pdf)
- [Zünd03] Zündorff, Albert (2003) Design Pattern. Vorlesungsskript. Universität Kassel (<http://www.se.eecs.uni-kassel.de/se/fileadmin/se/courses/DesignPattern/DPAZPart1.Version2.pdf>)