Aufgaben und Lösungen

zu dem Buch

Architektur- und Entwurfsmuster der Softwaretechnik

mit lauffähigen Beispielen in Java

Joachim Goll, Manfred Dausmann
Springer Vieweg, 2013

Inhaltsverzeichnis

Lösungen zu den Aufgaben in Kapitel 1

Aufgaben 1.11.1: Objektorientierter Entwurf einzelner Klassen

1.11.1.1 Von wem stammen jeweils die Prinzipien Separation of Concerns und das Single Responsibility-Prinzip?

Lösung: Das Prinzip Separation of Concerns wurde von Edsger W. Dijkstra in der Veröffentlichung „On the role of scientific thought“ postuliert. Das Single Responsibility-Prinzip stammt von Robert C. Martin.

1.11.1.2 Sind Separation of Concerns und das Single Responsibility-Prinzip im Hinblick auf Klassen äquivalent?

Lösung: Das Single Responsibility-Prinzip gilt nur für Klassen, Separation of Concerns gilt prinzipiell. Man kann Separation of Concerns als einen prinzipiellen Prozess der Zerlegung betrachten und das Single Responsibility-Prinzip als ein objektorientiertes Design-Prinzip für Klassen. Im Hinblick auf Klassen sind sie äquivalent.

1.11.1.3 Was bedeutet das Interface Segregation-Prinzip?

Lösung: Das Interface Segregation-Prinzip (ISP) verlangt, dass ein Interface nur den Anforderungen eines bestimmten Clients genügt. Das Interface eines Clients enthält also nur diejenigen Methodenköpfe, die ein Client braucht. Damit schlagen Änderungen an nicht benötigten Schnittstellen nicht auf Client-Programme durch.

Aufgaben 1.11.2: Objektorientierter Entwurf miteinander kooperierender Klassen

1.11.2.1 Erklären Sie strong cohesion und loose coupling.

Lösung: Cohesion ist ein Maß für den Zusammenhalt innerhalb eines Teilsystems. Coupling ist ein Maß für die Stärke der Wechselwirkung zwischen Teilsystemen.

Ziel beim Entwurf ist es, innerhalb eines Teilsystems eine möglichst hohe Bindungsstärke, sprich strong cohesion, zu erreichen und zwischen den Teilsystemen eine schwache Wechselwirkung, d.h. loose coupling.

1.11.2.2 Erklären Sie das liskovsche Substitutionsprinzip.

Lösung: Will man Polymorphie von Objekten für ein Programm haben, so müssen Methoden, die Zeiger oder Referenzen auf Basisklassen verwenden, in der Lage sein, Objekte von abgeleiteten Klassen zu benutzen, ohne es zu bemerken.

1.11.2.3 Erklären Sie das Open-Closed-Prinzip.

Lösung: Dieses Prinzip besagt, dass ein Modul offen im Sinne von Erweiterbarkeit sein soll in demjenigen Sinne, dass er Teil eines neuen Moduls werden kann, und geschlossen gegenüber Veränderungen, um eine Wiederverwendbarkeit zu erreichen.

1.11.2.4 Erklären Sie das Prinzip der Dependency Inversion in einer Hierarchie.

Lösung: Beim Prinzip der Dependency Inversion in einer Hierarchie gilt folgendes:

- Eine Klasse einer höheren Ebene soll nicht von einer Klasse einer tieferen Ebene abhängig sein.
- Eine Klasse der höheren und eine Klasse der tieferen Ebene sollen jeweils von derselben Abstraktion abhängen. Als Abstraktion soll die Klasse der höheren Ebene ein Interface oder eine abstrakte Klasse aggregieren. Diese Abstraktion soll wiederum von der Klasse der tieferen Ebene implementiert werden.
- Eine Abstraktion soll nicht von einer Klasse einer tieferen Ebene abhängen.
- Hingegen soll eine Klasse einer tieferen Ebene von einer Abstraktion abhängen.
- Die Abstraktion wird von der Klasse der höheren Ebene vorgegeben.

1.11.2.5 Erklären Sie die Inversion der Kontrolle des Flusses.

Lösung: Inversion of Control bei der Kontrolle des Flusses bedeutet, dass ein spezielles Modul wie eine Anwendung die Steuerung des Kontrollflusses an ein meist wiederverwendbares Modul abgibt. Damit das wiederverwendbare Modul das spezielle Modul nutzen kann, definiert das wiederverwendbare Modul ein Callback-Interface, das von dem speziellen Modul implementiert werden muss.

Anwendung findet dieses Paradigma oft bei Frameworks, um eine Funktion einer Anwendung aufzurufen.

1.11.2.6 Was ist Dependency Look-up? Was ist Dependency Injection?

Lösung: Dependency Look-Up bedeutet, dass ein Objekt seine Verknüpfung mit einem anderen Objekt zur Laufzeit herstellt, indem es nach diesem anderen Objekt über einen logischen Namen in einem weiteren Objekt – wie beispielsweise einer Registry oder einem Container – sucht.

Dependency Injection bedeutet, dass eine Verknüpfung zwischen Objekten zur Laufzeit von einer eigenen Instanz (Injektor) hergestellt wird und nicht zur Kompilierzeit.

1.11.2.7 Was ist Constructor Injection? Was ist Setter Injection? Was ist Interface Injection?

Lösung: Für Dependency Injection gibt es drei Möglichkeiten:

Bei der Constructor Injection können Abhängigkeiten zu anderen Objekten direkt bei der Erzeugung über einen Konstruktor mit Parametern erzeugt werden.

Bei der Setter Injection werden die Abhängigkeiten zu anderen Objekten erst später vom Injektor über set-Methoden festgelegt.

Bei der Interface Injection wird zusätzlich zu der Abstraktion für die zu injizierenden Objekte noch ein Interface für die Injektion dieser Objekte vorgegeben. Eine Klasse, in die eine Verbindung injiziert werden soll, muss das entsprechende Interface implementieren. Das Interface definiert eine Methode, die von einem Injektor zum Setzen der Abhängigkeit eines Objektes der implementierenden Klasse von einem Objekt der zu injizierenden Klasse aufgerufen werden kann.

Aufgaben 1.11.3: Zusicherungen

1.11.3.1 Was ist eine Zusicherung?

Eine Zusicherung besteht aus Bedingungen über die Zustände von Objekten an einer bestimmten Programmstelle.

1.11.3.2 Betreffen Vor- und Nachbedingungen eine Methode oder eine Klasse?

Lösung: Vor- und Nachbedingungen werden für Methoden einer Klasse erstellt.

1.11.3.3 Wer hat die Pflicht bei einer Vorbedingung, wer hat den Nutzen?

Lösung: Der Aufrufer hat die Pflicht, die Vorbedingung des Aufzurufenden zu erfüllen. Er muss prüfen, ob eine Vorbedingung erfüllt ist. Den Nutzen hat der Aufgerufene.

1.11.3.4 Wer hat die Pflicht bei einer Nachbedingung, wer hat den Nutzen?

Lösung: Bei der Nachbedingung hat der Aufgerufene die Pflicht, den Nutzen hat der Aufrufer. Der Aufgerufene muss also die Nachbedingung erfüllen.

1.11.3.5 Betreffen Invarianten eine Methode oder eine Klasse?

Lösung: Invarianten werden für eine Klasse erstellt.

1.11.3.6 Kann man bei einer Ableitung Vorbedingungen verschärfen?

Lösung: Nein, eine überschreibende Methode einer abgeleiteten Klasse darf eine Vorbedingung der überschriebenen Methode nicht verschärfen, d. h. wenn eine Methode z. B. einen formalen Parameter vom Typ `int` spezifiziert und einen gültigen Wertebereich zwischen 1 und 10 hat, so darf die überschriebene Methode diesen Wertebereich nicht einschränken.

1.11.3.7 Kann man bei einer Ableitung Nachbedingungen aufweichen?

Lösung: Nein, eine überschriebene Methode einer abgleitenden Klasse darf eine Nachbedingung der überschriebenen Methode nicht aufweichen, d. h. wenn eine Methode z.B. einen Rückgabewert vom Typ `int` hat und garantiert, dass sie nur Werte zwischen 1 und 10 liefert, so darf die überschriebene Methode keine Werte außerhalb dieses Bereichs liefern.

Lösungen zu den Aufgaben in Kapitel 2

Aufgaben 2.7.1: Softwarearchitektur allgemein

2.7.1.1 Was ist ein Stakeholder?

Lösung: Jemand, der ein Interesse an einem Projekt hat wie ein Nutzervertreter oder ein EDV-Verantwortlicher, wird auch als Stakeholder bezeichnet.

2.7.1.2. Wie lautet die Definition des Begriffs Architektur?

Lösung: Die Architektur eines Systems umfasst:

- die Zerlegung des Systems in seine physischen Bestandteile (Komponenten) und bei verteilten Systemen die Verteilung dieser Komponenten auf die einzelnen Rechner (Deployment),
- die Beschreibung des dynamischen Zusammenwirkens aller Komponenten und
- die Beschreibung der Strategie für die Architektur, d. h. wie die Architektur funktionieren soll,
- mit dem Ziel, alle nach außen geforderten Leistungen des Systems erzeugen zu können.

2.7.1.3 In welchem Entwicklungsschritt entsteht die Architektur einer Software?

Lösung: Die eigentliche Softwarearchitektur eines Systems wird in dem Entwicklungsschritt Systementwurf konzipiert. Es wird allerdings fast immer schon früher begonnen, an der Architektur zu arbeiten. Das Ergebnis des Systementwurfs oder des Designs soll die Architektur des Systems sein. Eine Architektur wird meist iterativ entwickelt.

2.7.1.4 Was ist das Ergebnis des Systementwurfs?

Lösung: Das Ergebnis des Systementwurfs ist die Architektur.

2.7.1.5 Welche Prototypen gibt es zur Verifikation einer Architektur und wozu werden die einzelnen Typen verwendet?

Lösung: Es gibt verschiedene Prototypen zur Verifikation einer Architektur:

- horizontale Prototypen,
- vertikale Prototypen oder
- Realisierbarkeitsprototypen („Angsthasen“-Prototypen).

Horizontale Prototypen demonstrieren die Tauglichkeit einer Schicht wie z. B. des MMI. Vertikale Prototypen sind ein Durchstich durch alle Schichten eines Systems. Realisierbarkeitsprototypen (sogenannte

"Angsthasen"-Prototypen) untersuchen experimentell, ob bestimmte Systemteile realisierbar sind. Was man nicht theoretisch zeigen kann, muss man eben experimentell beweisen.

2.7.1.6 Beschreiben Sie das Verhältnis zwischen Programm, Anforderungen und Architektur.

Lösung: Eine Architektur ist eine Abstraktion eines Programms und stellt das Bindeglied zwischen Anforderungen und Programm dar.

Aufgaben 2.7.2: Referenzarchitektur, Architektur- und Entwurfsmuster

2.7.2.1 Was ist ein Architekturmuster?

Lösung: Ein Architekturmuster beschreibt die Zerlegung eines Systems in Komponenten nach einer bestimmten Strategie.

2.7.2.2 Was ist ein Entwurfsmuster?

Lösung: Klassen in Rollen, die zusammenarbeiten, um gemeinsam eine bestimmte Aufgabe durchzuführen.

2.7.2.3 Ist ein Architekturmuster gröber als eine Referenzarchitektur?

Lösung: Nein, Referenzarchitekturen sind gröber als Architekturmuster, da sie verschiedene Architekturmuster enthalten können und überdies noch individuell zu entwickelnde Teile beinhalten.

2.7.2.4 Ist ein Architekturmuster feiner als ein Entwurfsmuster?

Lösung: Nein, Architekturmuster sind gröbere Einheiten als Entwurfsmuster. Architekturen können – aber müssen nicht – mehrere verschiedene Entwurfsmuster beinhalten.

Lösungen zu den Aufgaben in Kapitel 3

Aufgaben 3.6.1: Allgemeine Aufgaben

3.6.1.1 Erklären Sie den Zusammenhang zwischen Architekturmustern, Entwurfsmustern und Idiomen.

Lösung: Bei Architekturmustern betrachtet man die Architektur eines gesamten Systems, bei Entwurfsmustern in der Regel ein bestimmtes Problem innerhalb eines Subsystems. Ein Entwurfsmuster beeinflusst die Architektur eines Subsystems, aber nicht die grundsätzliche Architektur des Systems. Ein Idiom ist ein Muster in einer bestimmten Programmiersprache wie z. B. ein ausprogrammiertes Entwurfsmuster.

3.6.1.2 Muss ein Architektur- oder Entwurfsmuster objektorientiert sein?

Lösung: Nein, Muster müssen nicht grundsätzlich objektorientiert sein. Nicht objektorientierte Muster kommen insbesondere ohne Vererbung und Polymorphie aus.

Aufgaben 3.6.2: Entwurfsmuster

3.6.2.1 Was ist das Ziel der Entwurfsmuster?

Lösung: Ziel von Entwurfsmustern ist es, gewonnene Erkenntnisse über die Lösung bestimmter Problemstellungen wieder verwendbar zu machen und durch ihre Anwendung die Flexibilität eines Entwurfs zu erhöhen.

3.6.2.2 Wie entsteht ein Entwurfsmuster?

Lösung: Muster werden nicht erfunden, sondern in Anwendungen als Lösungen für typische Probleme und Sachverhalte "entdeckt", auf Konferenzen vorgetragen und dann von der Fachwelt akzeptiert oder auch nicht. Ein Muster kann für viele verschiedene Anwendungen eingesetzt werden.

3.6.2.3 Was sind Mikroarchitekturen?

Lösung: Entwurfsmuster werden als bewährte Konstruktionsprinzipien im Kleinen eingesetzt. Man spricht auch von Mikroarchitekturen.

3.6.2.4 Kann man sich sicher sein, dass man eine gute Architektur erstellt hat, wenn sie viele Entwurfsmuster enthält?

Lösung: Aus Entwicklersicht bieten Entwurfsmuster eine gewisse Sicherheit. Diese Sicherheit beruht auf der Tatsache, dass der Entwurf, den man verwenden will, sich in der Vergangenheit bereits bewährt hat. Dies führt im Allgemeinen jedoch zu dem Schluss, dass das Anwenden von Entwurfsmustern in jedem Fall die richtige Entscheidung ist. Diese

Annahme ist jedoch falsch. Das reine Anwenden von Entwurfsmustern garantiert einem Entwickler nicht, dass das Muster für sein konkretes Entwurfsproblem sinnvoll ist.

Lösungen zu den Aufgaben in Kapitel 4

Aufgaben 4.23.1: Allgemeine Fragen

4.23.1.1 Welche beiden Entwurfsmuster haben dasselbe Klassendiagramm, das aber jeweils anders interpretiert wird?

Lösung: Zustand und Strategie.

4.23.1.2 Wann zieht man eine abstrakte Basisklasse einem Interface vor?

Lösung: Wenn man eine Aggregation oder eine Default-Implementierung bereitstellen muss.

4.23.1.3 Was ist ein Strukturmuster?

Lösung: Strukturmuster befassen sich mit der Zusammensetzung von Klassen und Objekten.

4.23.1.4 Was ist ein Verhaltensmuster?

Lösung: Verhaltensmuster beschreiben Interaktionen zwischen Objekten in bestimmten Rollen zur gemeinsamen Lösung eines bestimmten Problems.

4.23.1.5 Was ist ein Erzeugungsmuster?

Lösung: Erzeugungsmuster machen ein System unabhängig davon, auf welche Weise seine Objekte erzeugt werden.

Aufgaben 4.23.2: Adapter

4.23.2.1 Welche Ziel hat das Adaptermuster?

Lösung: Das Adapter-Muster passt eine vorhandene "falsche" Schnittstelle an die gewünschte Form an.

4.23.2.2 Was ist der Unterschied zwischen dem Proxy- und dem Adapter-Muster?

Lösung: Der Proxy erweitert ebenso wie der Adapter Objekte um zusätzliche Funktionalität. Ein Proxy verwendet dieselbe Schnittstelle wie ein Objekt einer echten Klasse, das er repräsentiert. Die Schnittstelle darf hier also nicht verändert werden. Für den Adapter gilt diese Einschränkung nicht. Im Gegenteil, der Adapter passt eine vorhandene Schnittstelle an die geänderte Form einer Schnittstelle an.

Aufgaben 4.23.3: Brücke

4.23.3.1 Wie funktioniert das Brücke-Muster?

Lösung: Das Brücke-Muster trennt eine Abstraktion und ihre Implementierung. Dadurch wird erreicht, dass diese beiden Teile getrennt verändert und erweitert werden können.

4.23.3.2 Wo ist die Brücke zu sehen?

Lösung: Die Abstraktion aggregiert das Interface der Implementierung. Die Aggregationsbeziehung verbindet wie eine Brücke die beiden Klassenhierarchien auf der Abstraktions- und der Implementierungsseite.

Aufgaben 4.23.4: Dekorierer

4.23.4.1 Welches Entwurfsmuster hat bis auf eine andere Multiplizität das gleiche Klassendiagramm wie das Dekorierer-Muster?

Lösung: Das Kompositum-Muster.

4.23.4.2 Wozu dient das Dekorierer-Muster?

Lösung: Das Dekorierer-Muster soll es erlauben, zur Laufzeit eine zusätzliche Funktionalität zu einem vorhandenen Objekt einer Klasse oder Subklasse in dynamischer Weise hinzuzufügen.

Aufgaben 4.23.5: Fassade

4.23.5.1 Wozu dient das Fassade-Muster?

Lösung: Das Fassade-Muster soll eine meist vereinfachte abstrakte Schnittstelle zum Zugriff auf die Klassen eines Subsystems bereitstellen.

4.23.5.2 Kann das Fassade-Muster den Zugriff auf die Subsystemklassen verhindern? Begründen Sie Ihre Antwort.

Lösung: Das Fassade-Muster bietet zwar eine einfache Schnittstelle zum Zugriff auf ein Subsystem, verhindert jedoch nicht den direkten Zugriff auf die Subsystemklassen. Ein direkter Zugriff auf eine Subsystemklasse ist trotz der Existenz der Fassade technisch immer noch möglich und sollte durch Konvention untersagt werden. Wenn ein Programmierer sich nicht an die Vereinbarungen hält, kann er die Fassadenschnittstelle umgehen. Solange der Direktzugriff erlaubt ist, kann man nicht die Vorteile von Information Hiding nutzen.

Aufgaben 4.23.6: Kompositum

4.23.6.1 Welche Rollen spielen die Klassen `Knoten`, `Kompositum` und `Blatt` im Rahmen des Kompositum-Patterns.

Lösung: Die abstrakte Klasse `Knoten` legt die Schnittstelle und das Verhalten der abgeleiteten Klassen `Kompositum` und `Blatt` fest. Es wird ein Defaultverhalten für die Kindoperationen implementiert.

Die Klasse `Blatt` repräsentiert ein Abschlusselement in der Baumstruktur, das keine weiteren Knoten aggregiert und selbst immer nur Kind-Knoten sein kann.

Die Klasse `Kompositum` repräsentiert ein Knotenelement in der Baumstruktur, welches weitere Knoten aggregieren kann.

4.23.6.2 Eine Blattklasse erbt von der Basisklasse `Knoten` die Kindfunktionen, hat aber keine Kinder. Wie geht man üblicherweise vor?

Lösung: Die Kindmethoden wie z. B. `gibKindKnoten()` oder `hinzufuegen()` der Klasse `Knoten` werfen üblicherweise Exceptions. Damit führt das Aufrufen einer Kindfunktion bei einem Blatt-Objekt zum Werfen einer Exception. Die Kind-Methoden des Kompositums überschreiben die Kindmethoden der Klasse `Knoten`.

Aufgaben 4.23.7: Proxy

4.23.7.1 Was ist das Ziel des Proxy-Musters?

Lösung: Das Proxy-Muster verbirgt die Existenz eines Objekts hinter einem Stellvertreter mit derselben Schnittstelle. Der Stellvertreter kapselt die Kommunikation zum echten Objekt. Er kann Funktionen an das echte Objekt weiterdelegieren und dabei auch Zusatzfunktionalität hinzufügen.

4.23.7.2 Was ist ein Schutz-Proxy?

Lösung: Mit dem Schutz-Proxy können die Zugriffe auf die Objekte der Anwendung überwacht werden. Der Schutz-Proxy überprüft beim Methodenaufruf, ob das aufrufende Client-Programm tatsächlich über die notwendige Zugriffsberechtigung verfügt. Auf diese Weise kann eine Rechteverwaltung nachträglich in ein bestehendes System eingeführt werden.

Aufgaben 4.23.8: Schablonenmethode

4.23.8.1 Was hat das Muster Schablonenmethode zum Inhalt?

Lösung: Das Muster Schablonenmethode legt bereits in einer Basisklasse die Struktur eines Algorithmus fest. Realisiert werden variante Teile des Algorithmus in Unterklassen.

4.23.8.2 Arbeitet das Muster Schablonenmethode klassenbasiert oder objektbasiert?

Lösung: Da das Muster Schablonenmethode auf statischer Vererbung basiert, ist es ein klassenbasiertes Entwurfsmuster. Einschubmethoden werden in Unterklassen implementiert, wobei jede Unterklasse die Einschubmethoden unterschiedlich implementieren kann.

Aufgaben 4.23.9: Befehl

4.23.9.1 Was ist das Ziel des Befehlsmusters?

Lösung: Das Befehlsmuster soll die Details eines Befehls vor dem Aufrufer des Befehls verbergen. Es soll es erlauben, dass die Erzeugungszeit und die Ausführungszeit des Befehls getrennt werden können.

4.23.9.2 Nennen Sie 2 Anwendungsgebiete für das Befehlsmuster.

Lösung: Logging – vereinfachte Protokollierung
Man kann die Referenzen auf die Befehlsobjekte in einem Stream (Kanal) aufreihen, wodurch man eine Logging-Möglichkeit für Befehle hat.

Undo-/Redo-Funktionalität – Rückgängigmachen von Befehlen
Ein Undo-/Redo-Mechanismus kann realisiert werden, indem ausgeführte Befehle gespeichert werden. Entsprechende `undo()`- und `redo()`-Methoden müssen in jeder einzelnen konkreten Befehlsklasse implementiert werden. Werden die Referenzen auf die ausgeführten Befehle dann in einem Befehlsstack abgelegt, so kann festgestellt werden, welcher Befehl als nächstes rückgängig gemacht oder erneut ausgeführt werden soll.

Aufgaben 4.23.10: Beobachter

4.23.10.1 Erklären Sie das Prinzip einer Callback-Schnittstelle anhand des Beobachter-Musters.

Lösung: Beim Beobachter-Muster werden Nachrichten in beide Richtungen zwischen Beobachter-Objekt und dem beobachteten Objekt ausgetauscht. Das beobachtbare Objekt verpflichtet den Aufrufer, eine spezielle Schnittstelle (Callback-Schnittstelle) zu implementieren, die vom beobachtbaren Objekt vorgegeben wird. Ändert sich der Beobachter, so hat dies keinen Einfluss auf das beobachtbare Objekt. Dagegen wirken sich Änderungen an der Callback-Schnittstelle oder den angebotenen Daten des beobachtbaren Objekts auf den Beobachter aus.

4.23.10.2 Kennt der Beobachtete beim Beobachter-Muster die komplette Implementierung seiner Beobachter?

Lösung: Änderungen am Beobachter sind für den Beobachtbaren irrelevant. Der Beobachtbare ist nur von der Callback-Schnittstelle des Beobachters abhängig, die er aber selbst vorschreibt. Zur Kompilierzeit kennt der Beobachtbare nur die Callback-Schnittstelle.

Aufgaben 4.23.11: Strategie

4.23.11.1 Wozu wird das Strategie-Muster eingesetzt?

Lösung: Das Strategie-Muster soll es erlauben, dass ein ganzer Algorithmus ausgetauscht wird, um die Wiederverwendbarkeit zu steigern. Dazu soll eine Gruppe von verschiedenen Algorithmen mit gleicher Schnittstelle definiert werden können und jeder einzelne separat gekapselt werden, um ihn als Kapsel austauschbar zu machen. Der jeweils zu einer bestimmten Zeit passende Algorithmus ist auszuwählen.

4.23.11.2 Erklären Sie den Unterschied zwischen den Verhaltensmustern Strategie und Schablonenmethode.

Lösung: Die Schablonenmethode erlaubt es, in verschiedenen Unterklassen definierte Teile eines Algorithmus auszutauschen. Das Entwurfsmuster Strategie tauscht den gesamten Algorithmus aus.

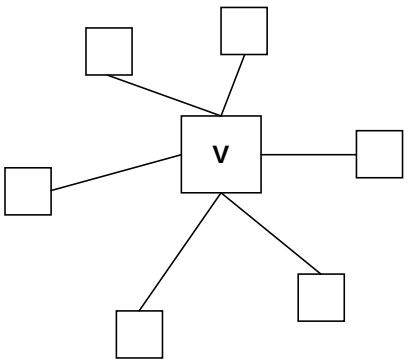
Aufgaben 4.23.12: Vermittler

4.23.12.1 Was ist das Ziel des Vermittler-Musters?

Lösung: Das Vermittler-Muster soll es erlauben, dass Objekte miteinander über einen Vermittler reden, der alle von einer eingehenden Nachricht betroffenen Objekte informiert. Dadurch bleiben die einzelnen Objekte unabhängig voneinander und sind austauschbar. Das Vermittler-Muster stellt die Koordination einer ganzen Gruppe von Objekten durch den zentralen Vermittler in den Vordergrund.

4.23.12.2 Zeichnen Sie die für das Vermittler-Muster zutreffende Topologie.

Lösung:



Aufgaben 4.23.13: Zustand

4.23.13.1 Was ist das Ziel des Zustandsmusters?

Lösung: Im Zustandsmuster werden die einzelnen Zustände durch eigene Klassen gekapselt, die von einer gemeinsamen abstrakten Basisklasse ablei-

ten bzw. eine gemeinsame Schnittstelle als Abstraktion implementieren. Ein zustandsabhängiges Kontextobjekt referenziert den aktuellen Zustand und führt im Rahmen des Grundmusters Zustandsänderungen durch.

4.23.13.2 Hängt der Kontext zur Kompilierzeit (statisch) von den konkreten Zustandsklassen ab?

Lösung: Im Entwurfsmuster Zustand hängt zur Kompilierzeit der Kontext nicht von den konkreten Zustandsklassen direkt ab, sondern nur von einem Interface bzw. von einer abstrakten Klasse, die das Kontextobjekt als Abstraktion vorgibt. Diese abstrakte Zustandsschnittstelle wird von den konkreten Zustandsklassen implementiert.

Aufgaben 4.23.14: Rolle

4.23.14.1 Definieren Sie den Begriff der Rolle.

Lösung: Eine Rolle ist ein wahrnehmbarer Verhaltensaspekt eines Objekts.

4.23.14.2 Was ist das Ziel des Rollenmusters?

Lösung: Das Rollenmuster soll es erlauben, dass ein Objekt dynamisch die Rollen wechseln und mehrere Rollen gleichzeitig annehmen kann, sowie, dass mehrere Objekte dieselbe Rolle haben können.

4.23.14.3 Wie wird beim Rollenmuster erreicht, dass ein Objekt zur Laufzeit verschiedene Rollen annehmen kann?

Lösung: Im Entwurfsmuster Rolle enthält eine Kernklasse die rollenunabhängige, d. h. statisch unveränderliche, Funktionalität eines Objekts. Die Rollen, die ein solches Objekt dynamisch spielen kann, werden in Rollenklassen bzw. Rollenobjekten implementiert. Ein Rollenobjekt aggregiert das Kernobjekt, welches diese Rolle gerade spielt. Es ist auch möglich, dass ein Kernobjekt mehrere Rollen gleichzeitig annehmen kann. Die Objekte der verschiedenen Rollen aggregieren einfach alle dasselbe Kernobjekt.

4.23.14.4 Was ist der Unterschied zwischen dem Rollenmuster und der Spezialisierung durch Ableitung?

Lösung: Im Gegensatz zum Rollenmuster ist die Spezialisierung durch Ableitung statisch und nicht flexibel. Sie ist aber sinnvoll, wenn ein Objekt unveränderliche Ausprägungen besitzt. Ist ein Objekt einer abgeleiteten Klasse erzeugt, so kann es nicht verändert werden. Die Klassen `Mann` oder `Frau` können beispielsweise gefahrlos von Person abgeleitet werden, da das Geschlecht eine statische Eigenschaft darstellt. Wenn ein Objekt zu verschiedenen Zeiten unterschiedliche Rollen übernehmen kann, ist die Spezialisierung in abgeleiteten Objekten jedoch die falsche Vorgehensweise.

Aufgaben 4.23.15: Besucher

4.23.15.1 Was ist das Ziel des Besucher-Musters?

Lösung: Das Besucher-Muster soll es erlauben, eine neue Operation auf einer Datenstruktur aus Objekten durchzuführen. Beim Besucher-Muster müssen die zu besuchenden Objekte auf den Besuch "vorbereitet" sein, dadurch dass sie eine entsprechende Methode dem Besucher zur Verfügung stellen.

4.23.15.2 Beschreiben Sie die Aufgaben eines Besucher-Objekts.

Lösung: Es werden alle Objekte einer Objektstruktur besucht, jeweils lokale Daten ausgelesen und eine objektübergreifende Berechnung gemacht.

Aufgaben 4.23.16: Iterator

4.23.16.1 Was ist das Ziel des Iterator-Musters?

Lösung: Das Iterator-Muster soll es erlauben, eine Datenstruktur in verschiedenen Durchlaufstrategien zu durchlaufen, ohne dass der Client den Aufbau der Datenstruktur kennt.

4.23.16.2 Nennen Sie ein Entwurfsmuster, das dem Iterator-Muster ähnelt.

Lösung: Ein Entwurfsmuster das dem Iterator-Muster ähnelt ist das Besucher-Muster. Dieses Muster ähnelt dem Iterator-Muster insofern, als dass sie sich beide über die Elemente einer Datenstruktur hinweg bewegen. Die Aufgabe eines Iterators beschränkt sich auf einen solchen Durchlauf der Datenstruktur, während ein Besucher zusätzlich während des Durchlaufs eine Funktionalität erbringt.

4.23.16.3 Was ist der grundlegende Gedanke des Iterator-Musters?

Lösung: Ein Iterator trennt den Mechanismus des Traversierens von der zu durchquerenden Datenstruktur. Diese Trennung ist der grundlegende Gedanke des Iterator-Musters. Die Realisierung der Traversierung wird aus der Datenstruktur herausgehalten. Dadurch wird die Datenstruktur einfacher.

Aufgaben 4.23.17: Fabrikmethode

4.23.17.1 Was ist das Ziel des Musters Fabrikmethode?

Lösung: Das Fabrikmethode-Muster soll es erlauben, dass die Erzeugung einer konkreten Instanz in der Methode einer Unterklasse gekapselt wird. Die Unterklassen werden durch Vererbung statisch gewonnen.

4.23.17.2 Wann wird eine Fabrikmethode eingesetzt?

Lösung: Eine Fabrikmethode wird eingesetzt, wenn Objekte einer Klasse erzeugt werden, deren Typ von vornherein nicht bekannt ist. Deshalb wird dieses Erzeugermuster in vielen Frameworks (Klassenbibliotheken) eingesetzt.

4.23.17.3 Welche Nachteile ergeben sich durch die Verwendung des Fabrikmethode-Musters?

Lösung: Nachteile des Musters Fabrikmethode sind:

- Für jede neue Produktklasse muss eine weitere Unterklasse eingeführt werden, um die Fabrikmethode entsprechend neu zu definieren. Damit erhöht sich die Komplexität.
- Aus Performance-Gründen kann es sinnvoller sein, die Konstruktoren der Objekte direkt aufzurufen und nicht die Fabrikmethode einzusetzen.

Aufgaben 4.23.18: Abstrakte Fabrik

4.23.18.1 Was ist das Ziel des Musters Abstrakte Fabrik?

Lösung: Das Muster Abstrakte Fabrik soll es erlauben, dass durch Wahl der entsprechenden konkreten Fabrik die Produktfamilie der zu erzeugenden Produkte zur Laufzeit ausgewählt werden kann.

4.23.18.2 Was ist der Unterschied zwischen den beiden Erzeugungsmustern Fabrikmethode und Abstrakte Fabrik?

Lösung: Die Abstrakte Fabrik hat im Gegensatz zu der Fabrikmethode eine Produktfamilie und nicht ein einzelnes Produkt zum Inhalt. Die Erzeugung eines konkreten Objekts erfolgt in beiden Fällen in Unterklassen.

Aufgaben 4.23.19: Singleton

4.23.19.1 Wozu wird das Singleton-Muster eingesetzt?

Lösung: Das Singleton-Muster soll gewährleisten, dass eine Klasse nur einmal instanziiert werden kann.

4.23.19.2 Skizzieren Sie die Lösung beim Entwurfsmuster Singleton.

Lösung: Die Klasse hat nur einen privaten Konstruktor. Damit muss das einzige Objekt in der Klasse selbst erzeugt werden und nach außen zur Verfügung gestellt werden. Dies wird meist in einer öffentlichen Klassenmethode durchgeführt.

Aufgabe 4.23.20: Objektpool

4.23.20.1 Was ist das Ziel des Musters Objektpool?

Lösung: Das Muster Objektpool soll es ermöglichen, Instanzen zur Verfügung zu stellen, dabei diese wiederzuverwenden und so Ressourcen zu schonen, anstatt die Instanzen immer wieder neu zu erzeugen.

4.23.20.2 Wofür wird im Zusammenhang mit Datenbanken das Objektpool-Muster eingesetzt?

Lösung: Das Entwurfsmuster eines Objektpools wird häufig bei Datenbankzugriffen verwendet, um Verbindungen zur Datenbank zwischenzuspeichern und wiederzuverwenden. Es wird dann in der Regel von einem Connection Pool gesprochen. Dieser Connection Pool ist insbesondere auch bei Web-Technologien sinnvoll, da diese viele kurze Verbindungen nutzen.

Lösungen zu den Aufgaben in Kapitel 5

Aufgaben 5.8.1: Layers

5.8.1.1 Erklären Sie die Grundzüge des Layers-Musters.

Lösung: Das Architekturmuster Layers schneidet die Architektur eines Systems in Schichten, wobei jede Schicht auf die Dienste der darunterliegenden Schicht zugreifen kann (horizontale Schichtung eines Systems). Diese Architektur setzt voraus, dass sich ein sinnvoller Ablauf ergibt, wenn ein Client die oberste Schicht des Systems als Server ruft, diese in der Rolle eines Clients wieder die nächst tiefere Schicht als Server usw. bis hin zur letzten Schicht.

5.8.1.2 Kann eine Schicht n die Dienste einer Schicht n-2 direkt nutzen?

Lösung: In der Grundform des Musters darf eine Schicht nur auf die direkt darunter liegende Schicht zugreifen. Dies wird auch Strict Layering genannt. Der Begriff Layer Bridging wird dagegen verwendet, wenn eine Schicht auf alle unter ihr liegenden Schichten zugreifen darf. Nur wenn also Layer Bridging erlaubt ist, kann die Schicht n die Dienste einer Schicht n-2 nutzen.

Aufgaben 5.8.2: Pipes and Filters

5.8.2.1 Erklären Sie die Grundzüge des Pipes and Filters-Musters.

Lösung: Das Architekturmuster Pipes and Filters strukturiert in seiner Grundform eine Anwendung in eine Kette von sequenziellen Verarbeitungsprozessen (Filtern), die über ihre Ausgabe bzw. Eingabe gekoppelt sind: Die Ausgabe eines Prozesses ist die Eingabe des nächsten Prozesses (datenflussorientierte Architektur eines Systems). Diese Architektur macht Sinn, wenn sich ein System in eine Kette von Prozessen gliedern lässt, wobei jeder Prozess das Ergebnis des vorangehenden Prozesses übernimmt und weiterverarbeitet. Eine Pipe dient hierbei zur asynchronen Entkopplung zweier benachbarter Prozesse.

5.8.2.2 Was ist der Unterschied zwischen einem aktiven und einem passiven Filter?

Lösung: Ein aktiver Filter stellt einen eigenständigen parallelen Prozess dar. Ein aktiver Filter ist der typische Fall. Er wird von einem übergeordneten Programm aufgerufen und läuft immer als parallele Einheit. Es gibt auch passive Filter, die von einem dem passiven Filter benachbarten Filterelement aufgerufen und damit aktiviert werden.

Aufgaben 5.8.3: Plug-in

5.8.3.1 Erklären Sie die Grundzüge des Plug-in-Musters.

Lösung: Das Architekturmuster Plug-in strukturiert eine spezielle Anwendung so, dass diese über Erweiterungspunkte in Form von Schnittstellen verfügt, an denen dann die Plug-ins, die diese Schnittstellen implementieren, eingehängt werden können. Eine Anwendung ist aber bereits ohne diese zusätzlichen Erweiterungen lauffähig. Die Architektur der Anwendungssoftware muss Schnittstellen definieren, an deren Stelle zur Laufzeit Komponenten, die diese Schnittstelle implementieren, treten können.

5.8.3.2 Welche Aufgaben hat ein Plug-in-Manager?

Lösung: Die Plug-ins werden von einem Plug-in-Manager verwaltet. Die Applikation definiert Schnittstellen zur Erweiterung beispielsweise in einer Datei. Unter Verwendung einer Komponente der Laufzeitumgebung instanziiert und verwaltet der Plug-in-Manager die (anhand der Datei gefundenen) Plug-ins. Mittels des Plug-in-Managers bekommt die Applikation Zugriff auf die Instanzen der verfügbaren Plug-ins, ohne diese selbst zu instanziieren.

Aufgaben 5.8.4: Broker

5.8.4.1 Erklären Sie die Grundzüge des Broker-Musters.

Lösung: Das Broker-Muster entkoppelt in verteilten Softwarearchitekturen Client- und Server-Komponenten, indem zwischen diesen Komponenten ein Broker als Zwischenschicht eingeschoben wird und Client- und Server-Komponente untereinander nur über einen Broker kommunizieren. Server melden ihre Dienste am Broker an und warten auf eingehende Anfragen von einem Client. Clients, die einen Dienst benötigen, stellen ihre Anfrage an den Broker. Der Broker leitet eine solche Anfrage dann an den entsprechenden Server weiter, der diesen Dienst anbietet. Die Antwort des Servers geht wiederum über den Broker zurück an den Client. Damit ist der Ort der Leistungserbringung transparent für den Client.

5.8.4.2 Welche Rollen spielen ein Client-side Proxy und ein Server-side Proxy im Rahmen des Broker-Musters?

Lösung: Ein Broker kommuniziert nicht direkt mit Client und Server, sondern nur indirekt über die jeweiligen Proxys. Die Aufgabe der Proxys ist es, einen Methodenaufruf zu serialisieren bzw. zu deserialisieren und auf dem umgekehrten Weg das Ergebnis des Methodenaufrufs zu serialisieren bzw. zu deserialisieren.

Aufgaben 5.8.5: Service-Oriented Architecture

5.8.5.1 Erklären Sie die Grundzüge einer Service-Oriented Architecture.

Lösung: Das Architekturmuster Service-Oriented Architecture gliedert die Architektur eines Systems in Komponenten bzw. Teilkomponenten, die den Anwendungsfällen bzw. Teilen von Anwendungsfällen aus Sicht der Systemanalyse entsprechen und deren Leistungen als Service zur Verfügung stellen. Damit sollen bei einer Abänderung der Geschäftsprozesse nur:

- die entsprechenden Komponenten als Verkörperung der Anwendungsfälle abgeändert werden müssen oder
- ggfs. unter Verwendung bestehender Teilservices neue Komponenten erzeugt werden.

5.8.5.2 Erläutern Sie die Rollen von Serviceanbieter, Servicenutzer und Serviceverzeichnis.

Lösung: Der sogenannte Serviceanbieter (engl. service provider) erbringt eine Dienstleistung, die von einem Servicenutzer (engl. service consumer) innerhalb oder außerhalb des anbietenden Unternehmens genutzt werden kann. Die beiden Rollen Serviceanbieter und Servicenutzer können durch ganze Geschäftsbereiche, Abteilungen, Teams oder einzelne Personen ausgeübt werden. Daneben gibt es eine dritte Rolle: häufig werden Informationen über einen Service in einem Serviceverzeichnis (engl. service registry) gespeichert, damit diese Informationen von Interessenten gesucht, gefunden und eingesetzt werden können.

Aufgaben 5.8.6: Model-View-Controller

5.8.6.1 Erklären Sie die Grundzüge des Model-View-Controller-Musters.

Lösung: Das Architekturmuster Model-View-Controller trennt eine Anwendung in die Komponenten Model (Verarbeitung/Datenhaltung), View (Eingabe) und Controller (Ausgabe). Das Model hängt dabei nicht von der Ein- und Ausgabe ab. Mit dieser Strategie ist es leicht möglich, View und Controller im System auszutauschen (Trennung der Oberflächenkomponenten von der Verarbeitung). Damit kann der Kern einer Anwendung länger leben als die Programmteile für die Ein- und Ausgabe.

5.8.6.2 Welche Abhängigkeiten bestehen zwischen View, Controller und Model?

Lösung: Der Controller erhält die Daten von der View. Er muss das Format der Daten beherrschen. Wird die Struktur der Daten der View geändert, muss der Controller geändert werden. In dieser Sichtweise ist der Controller abhängig von der View.

Wenn die Maske der View durch den Controller geändert werden kann, ist die View abhängig vom Controller. Ändert sich der Wert der Daten des Controllers entsprechend, so muss die View geändert werden.

Die View muss die Daten des Models darstellen. Damit ist sie abhängig vom Model.

Der Controller muss geänderte Daten im Model abspeichern. Damit ist er abhängig vom Model.

5.8.6.3 Welche Schwachstellen hatten Systeme üblicherweise vor MVC?

Früher: Die Benutzerschnittstelle war oft nicht klar definiert.
Eine Benutzerschnittstelle ändert sich erfahrungsgemäß rascher als der Kern der Anwendung. Ohne MVC war es schwer, die Benutzerschnittstelle zu ändern.

5.8.6.4 Nennen Sie die Aufgaben des Model im Pull-Betrieb des Active Model.

Lösung: Die Aufgaben des Model:

- Kapselt und speichert die Daten
- Realisiert die Kernfunktionalität der Anwendung (Geschäftsprozesse)
- Controller macht Änderungsanforderungen im Auftrag des Benutzers
- Beobachter, die an den Daten des Models interessiert sind, können sich am Model anmelden bzw. wieder abmelden, wenn sie kein Interesse mehr haben.
- Benachrichtigt angemeldete Views über Änderungen (Active Model)

5.8.6.5 Nennen Sie die Aufgaben der View im Pull-Betrieb des Active Model.

Lösung: Die Aufgaben der View:

- Das „Fenster“ zum Model
- Präsentiert dem Anwender die Daten
- Meldet sich beim Model an bzw. ab
- Wird vom Model bei Änderungen benachrichtigt
- Holt sich die Daten vom Model ab und aktualisiert die Darstellung
- Leitet eingegebene Daten und Ereignisse an den Controller weiter

5.8.6.6 Nennen Sie die Aufgaben des Controllers im Pull-Betrieb des Active Model.

Lösung: Die Aufgaben des Controllers:

- Interpretiert Eingaben des Benutzers in der View
- Ruft Methoden des Model auf, um Daten zu ändern
- Fordert View zur Änderungen der Darstellung auf
- Ein Controller gehört zu einer View (1-zu-1 Beziehung)
- Benachrichtigung der Views bei Datenänderung (Passive Model)