5AHIFS

Design Patterns

Berisa Valon & Savan Maljuric



Inhalt

[Was sind Design Patterns? 2](#_Toc264707)

[Anforderungen 2](#_Toc264708)

[Wozu Design Patterns? 2](#_Toc264709)

[Nachteile 3](#_Toc264710)

[Pattern Schablone 3](#_Toc264711)

[3 Typen von Design Patterns 5](#_Toc264712)

[1. Strukturelle Pattern (Structural Patterns) 5](#_Toc264713)

[2. Verhaltensorientierte Pattern (Behavioral Patterns) 5](#_Toc264714)

[3. Konzeptionelle Pattern (Creational Patterns) 6](#_Toc264715)

[Beispiele 6](#_Toc264716)

[1. Architectural Patterns 6](#_Toc264717)

[MVC 6](#_Toc264718)

[2. Structual Patterns 7](#_Toc264719)

[Data Access Object (DAO) Pattern 7](#_Toc264720)

[Adapter Pattern 9](#_Toc264721)

[3. Creational Pattern 11](#_Toc264722)

[Singleton Pattern 12](#_Toc264723)

[4. Behavorial Pattern 13](#_Toc264724)

[Observer Pattern 14](#_Toc264725)

# Was sind Design Patterns?

Design Patterns (Entwurfsmuster) sind bewährte Lösungswege für wiederkehrende Designprobleme in der Softwareentwicklung. Sie beschreiben die essenziellen Entwurfsentscheidungen (Klassen- und Objektarrangements). Durch den Einsatz von Design Pattern wird ein Entwurf flexibel, wiederverwendbar, erweiterbar, einfacher zu verwenden und änderungsstabil.

## Anforderungen

Ein gutes Design Pattern (Entwurfsmuster) sollte

* ein oder mehrere bekannte, wiederkehrende Probleme lösen
* ein erprobtes Konzept bieten
* auf realen Konstrukten basieren
* über das völlig Offensichtliche hinausgehen
* Beziehungen aufzeigen, die tiefergehende Strukturen und Mechanismen eines Systems umfassen
* Unterstützung für (Gesamt-)Design Entscheidungen bieten

## Wozu Design Patterns?

Der primäre Nutzen eines Design Patterns

* liegt in der Beschreibung einer Lösung für eine bestimmte Klasse von Entwurfsproblemen
* ergibt sich aus dem Namen des Musters. Dies vereinfacht die Diskussion unter den Entwicklern

Das Benutzen (und Dokumentieren) von Design Patterns erleichtert das Lesen/Verstehen des Programmcodes.

## Nachteile

Das Kennen vieler Design Patterns kann dazu verleiten

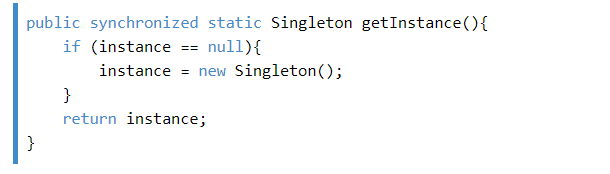
* Pattern als Wunderwaffe für ein gutes Design anzusehen
* viele bekannte Pattern zu verwenden und dabei einfachere/elegantere Lösungen zu übersehen
* den Code unnötig aufzublähen
* ein zu allgemeines (generisches) Problem zu lösen

## Pattern Schablone

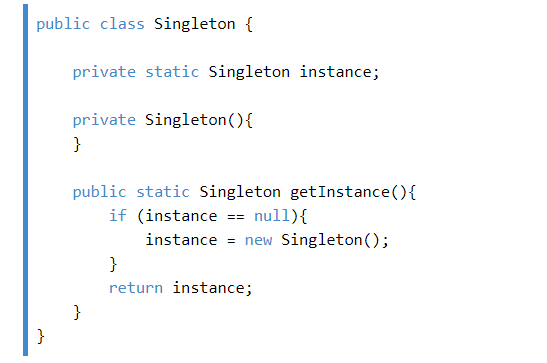
1. Pattern Name
2. Kategorisierung, Zweck
   1. Wozu dient das Pattern?
3. Szenario/Problem/Kontext
   1. Ein Beispiel Problem
4. Lösung/Struktur/Implementation
5. Vorteile/Nachteile
6. Verwendung
   1. Anwendungsgebiete
7. Varianten
8. Verweis auf andere Muster

Beispiel:

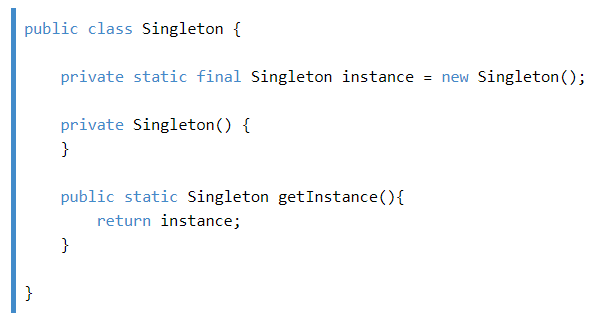
1. Singleton Pattern
2. Konzeptionelles Modell
   1. Einheitlich Objekt, welches nur einmal initialisiert wird und überall verwendet kann.
3. Problem
   1. Objekt nicht immer wieder erzeugen
4. Lösung
   1. Erstelle ein Objekt, wenn es nicht vorhanden ist.
   2. Falls vorhanden soll eine Referenz vom Objekt zurückgegeben werden.
5. Vorteile und Nachteile
   1. Vorteile: weniger Speicher, einheitliches Objekt, kann von überall zugegriffen werden
   2. Nachteile: Abhängigkeiten zur Singleton-Klasse werden verschleiert, Es besteht die große Gefahr, durch exzessive Verwendung von Singletons quasi ein Äquivalent zu [globalen Variablen](https://de.wikipedia.org/wiki/Globale_Variable) zu implementieren und damit dann prozedural anstatt objektorientiert zu programmieren.
6. Varianten
   1. Synchronozierung



* 1. Lazy Loading



* 1. Eager Loading



1. -

# 3 Typen von Design Patterns

Design Patterns für die Programmierung von Software lassen sich grundsätzlich in 3 Pattern-Klassen einteilen:

## 1. Strukturelle Pattern (Structural Patterns)

* die Bindung zwischen 2 oder mehr Klassen zu reduzieren
* eine abstrakte Klasse einzuführen, welche die zukünftige API-Erweiterung ermöglicht (vgl. Schnittstellenprogrammierung)
* komplexe Strukturen zu kapseln

## 2. Verhaltensorientierte Pattern (Behavioral Patterns)

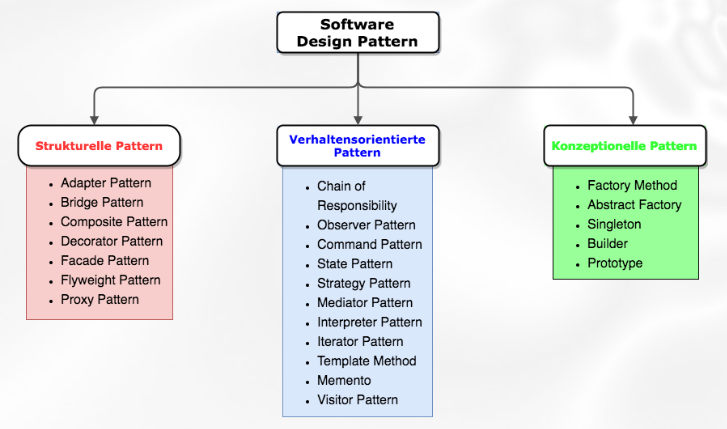
Erlauben die softwaregesteuerte Wahl zwischen einem Algorithmus und der Zuweisung von Verantwortlichkeiten von Objekten („Welches Objekt tut was?“)

Sie vereinfachen komplexe Steuerungsmechanismen und Kontrollflüsse die während der Laufzeit schwer zu analysieren sind.

## 3. Konzeptionelle Pattern (Creational Patterns)

Erlauben eine vereinfachte Darstellung von komplexen, instanziierten Prozessen

machen Softwaresysteme unabhängig davon wie Objekte entworfen, erstellt und repräsentiert werden



# Beispiele

## Architectural Patterns

### MVC

Das MVC Pattern hilft bei der Implementierung von Benutzeroberflächen in einer Software. Es teilt die gegebene Applikation in drei Teile:

• Model

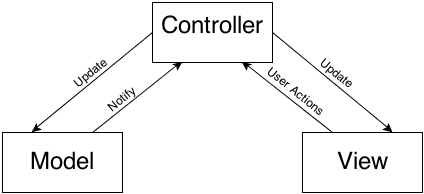
• View

• Controller

Für Ausgaben jeder Art ist die View-Komponente zuständig. Es kann auch vorkommen, dass es für dieselbe Information mehrere verschiedene Views gibt, die sich je nach Zielgruppe unterscheiden.

Zu guter Letzt, der Controller. Er nimmt Anfragen der anderen Komponenten entgegen und wandelt sie zu konkreten Befehlen für das Gegenüber um.

Ursprünglich wurde das MVC Pattern für Desktop GUIs verwendet, aber im Laufe der Zeit wurde es auch für die Verwendung in Web Applikationen und sogar für mobile Endgeräte bekannt.

Einer der größten Vorteile dieses Musters ist die Wiederverwendbarkeit, denn sauber getrennt können die Komponenten in zukünftigen Applikationen wieder problemlos eingebaut werden.

## Strukturelle Patterns

### Data Access Object (DAO) Pattern

Das Data Access Object (DAO) -Muster ist ein strukturelles Muster, das es uns ermöglicht, die Anwendungs- / Business-Schicht von der Persistenzschicht (normalerweise einer relationalen Datenbank, es könnte jedoch auch ein beliebiger anderer Persistenzmechanismus sein) mithilfe einer abstrakten API zu isolieren.

Es ist ein Objekt / (Interface) eine Schnittstelle, über die auf Daten Datenbank zugegriffen wird.

Warum benützt man DAO?

Es abstrahiert den Abruf von Daten aus einer Datenressource wie einer Datenbank. Das Konzept besteht darin, die Clientschnittstelle einer Datenressource von ihrem Datenzugriffsmechanismus zu trennen.

Beim direkten Zugriff auf Daten besteht das Problem, dass sich die Datenquelle sich ändern kann. Beispielsweise greift Ihre Anwendung auf eine Oracle-Datenbank zu. Auf einmal entscheidet man sich eine andere Datenbank zu verwenden (z.B. Microsoft SQL Server). Wenn Ihre Anwendung gespeicherte Prozeduren und datenbankspezifischen Code verwendet (z. B. das Generieren einer Zahlenfolge), wie gehen Sie damit in Ihrer Anwendung vor? Sie haben zwei Möglichkeiten:

• Schreiben Sie Ihre Anwendung neu, um SQL Server anstelle von Oracle zu verwenden (oder fügen Sie bedingten Code hinzu, um die Unterschiede zu behandeln) oder

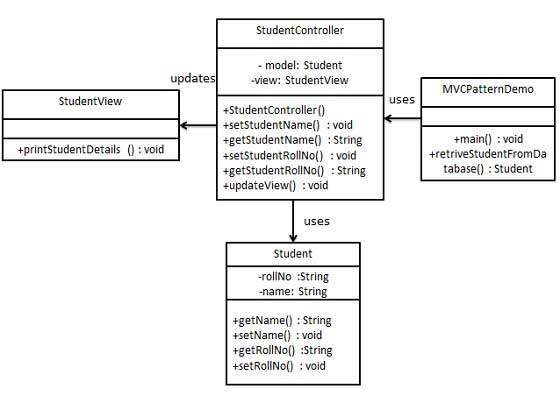
• Erstellen Sie eine Schicht zwischen Ihrer Anwendungslogik und dem Datenzugriff

Das DAO-Pattern besteht aus Folgendem:

• Data Access Object Interface - This interface defines the standard operations to be performed on a model object(s).

• Data Access Object concrete class - This class implements above interface. This class is responsible to get data from a data source which can be database / xml or any other storage mechanism.

• Model Object or Value Object - This object is simple POJO containing get/set methods to store data retrieved using DAO class.

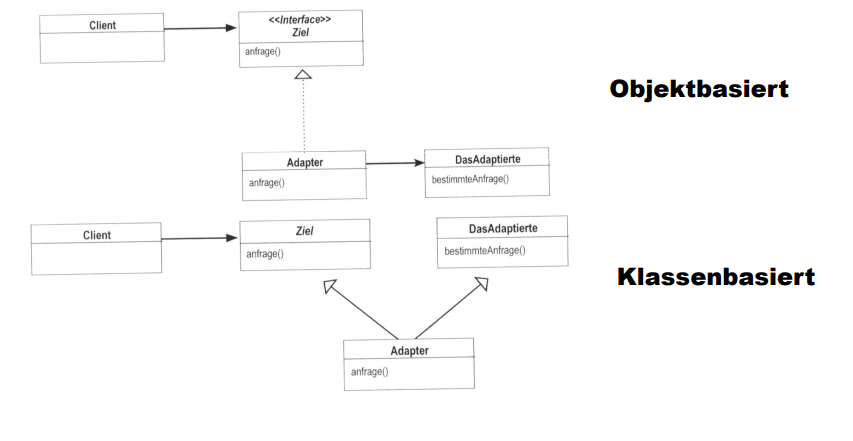


### Adapter Pattern

Das Adaptermuster fungiert als Brücke zwischen zwei inkompatiblen Schnittstellen. Diese Art von Entwurfsmustern fällt unter Strukturmuster, da dieses Muster die Fähigkeit von zwei unabhängigen Schnittstellen kombiniert.

Dieses Muster umfasst eine einzige Klasse, die dafür verantwortlich ist, Funktionalitäten unabhängiger oder inkompatibler Schnittstellen zusammenzuführen. Ein Beispiel aus der Praxis könnte ein Kartenleser sein, der als Adapter zwischen Speicherkarte und Laptop fungiert. Sie stecken die Speicherkarte in den Kartenleser und den Kartenleser in den Laptop ein, sodass die Speicherkarte über den Laptop gelesen werden kann.

Struktur:



Teilnehmer:

–Ziel: vom Client verwendete Schnittstelle.

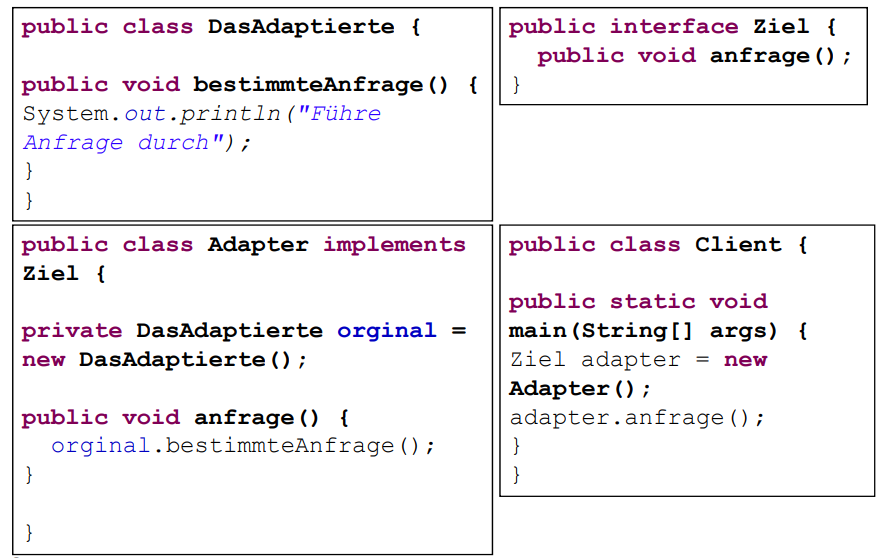
–Client: verwendet Objekte mit Zielschnittstelle.

–Das Adaptierte: definiert anzupassende Schnittstelle.

–Adapter: passt Schnittstelle der unpassenden Klasse an Zielschnittstelle an

Interaktionen: Clients rufen Operationen auf Adapterobjekten auf. Der Adapter delegiert den Aufruf an die Adaptierte Klasse und implementiert ggf. fehlende Methoden komplett neu.

Implementierung:

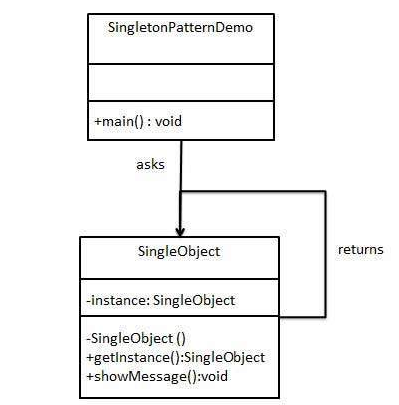


## Konzeptionelle Pattern

### Singleton Pattern

Singleton-Pattern ist eines der einfachsten Designmuster. Diese Art von Design-Mustern ist ein kreatives Muster, da dieses Muster eine der besten Möglichkeiten zum Erstellen eines Objekts bietet.

Dieses Muster umfasst eine einzelne Klasse, die dafür verantwortlich ist, ein Objekt zu erstellen, wobei sichergestellt wird, dass nur ein einzelnes Objekt erstellt wird. Diese Klasse bietet eine Möglichkeit, auf ihr einziges Objekt zuzugreifen, auf das direkt zugegriffen werden kann, ohne dass das Objekt der Klasse instanziiert werden muss.

Struktur:

Implementierung:

Wir haben eine Klasse „Database“ die als einzige Klasse Datenbank Aufrufe macht. Wir wollen sicherstellen, dass es nur ein Datenbankobjekt gibt, welches die Aufrufe ausführt.



Diese Klasse kann jetzt von anderen Klassen wie z.B. Controllern aufgerufen werden, um das einzig existierende Database Objekt zu benutzen.

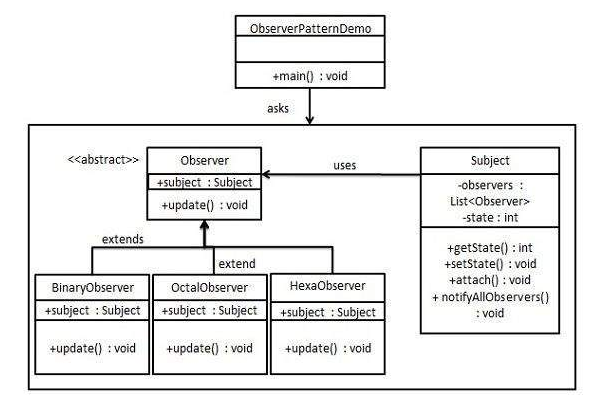


## Verhaltensorientierte Pattern

### Observer Pattern

Das Observer-Muster wird verwendet, wenn zwischen Objekten eine Eins-zu-Viele-Beziehung besteht, z. B. wenn ein Objekt geändert wird und seine abhängigen Objekte automatisch benachrichtigt werden. Das Beobachtermuster fällt in die Kategorie der Verhaltensmuster.

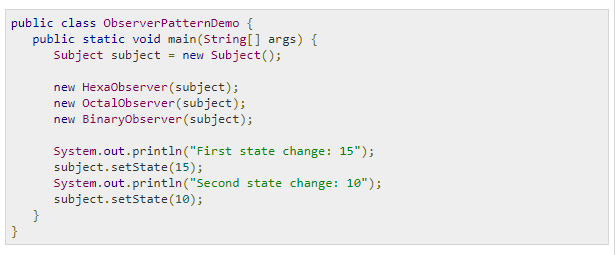
Struktur:



Implementierung:







Quellen:

<https://www.tutorialspoint.com/design_pattern/>

<https://www.journaldev.com/1827/java-design-patterns-example-tutorial>

<https://www.philipphauer.de/study/se/design-pattern/singleton.php#variationen>