

3. Entwurfsmuster zur Entkopplung von Modulen

OOP-3.1

Entwurfsmuster (Design Patterns):

Software-Entwicklungsaufgaben, die in vielen Ausprägungen häufig auftreten.

Objektorientierte Schemata, die als Lösungen **wiederverwendet** werden können.

Software-Qualitäten: erprobte Strukturen, wartbar, flexibel, adaptierbar.

Präsentation der Entwurfsmuster:

- Name:** Aufgabe treffend beschreiben, gut kommunizierbar
- Aufgabe:** typische Situation, gut wiedererkennbar, Struktur, Umgebung, Anwendungsbedingungen
- Lösung:** objektorientierte Strukturen, abstrakte Beschreibung variierbar, instanzierbar, Anwendungsbeispiele
- Folgerungen:** Nutzen, Kosten, Varianten; Implementierbeispiele

[E. Gamma, R. Helm, R. Johnson, J. Vlissides: Design Patterns, Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995]

© 2013 bei Prof. Dr. Uwe Kastens

Vorlesung Objektorientierte Programmierung WS 2013/2014 / Folie 301

Ziele:

Idee der Entwurfsmuster verstehen

in der Vorlesung:

Ziele

- Ähnliche Aufgaben wiedererkennen,
- bewährte Strukturen wiederverwenden,
- Aufwand für gute Lösungen reduzieren

nachlesen:

Budd: Understanding OOP with Java, Abschnitt 15

Übersicht zu Entwurfsmustern

OOP-3.2

Purpose	Design Pattern	Aspect(s) That Can Vary
Creational	Abstract Factory (87)	● families of product objects
	Builder (97)	how a composite object gets created
	Factory Method (107)	● subclass of object that is instantiated
	Prototype (117)	class of object that is instantiated
	Singleton (127)	the sole instance of a class
Structural	Adapter (139)	interface to an object
	Bridge (151)	● implementation of an object
	Composite (163)	structure and composition of an object
	Decorator (175)	responsibilities of an object without subclassing
	Facade (185)	interface to a subsystem
	Flyweight (195)	storage costs of objects
	Proxy (207)	how an object is accessed; its location
	Behavioral	Chain of Responsibility (223)
Command (233)		when and how a request is fulfilled
Interpreter (243)		grammar and interpretation of a language
Iterator (257)		how an aggregate's elements are accessed, traversed
Mediator (273)		how and which objects interact with each other
Memento (283)		what private information is stored outside an object, and when
Observer (293)		● number of objects that depend on another object; how the dependent objects stay up to date
State (305)		states of an object
Strategy (315)		● an algorithm
Template Method (325)		steps of an algorithm
Visitor (331)	operations that can be applied to object(s) without changing their class(es)	

[Gamma, u.a.: Design Patterns, S. 30]

behandelt

© 2005 bei Prof. Dr. Uwe Kastens

Vorlesung Objektorientierte Programmierung WS 2013/2014 / Folie 302

Ziele:

Eindruck von den Themen der Muster

in der Vorlesung:

Erläuterung der Klassifikation und einiger Beispiele

nachlesen:

Budd: Understanding OOP with Java, Abschnitt 15

Adaptierbare Strukturen

Spezielles Ziel in dieser Vorlesung:

Software-Strukturen für zukünftige Änderungen vorbereiten; „Designing for Change“

Einige Gründe für **schlechte Adaptierbarkeit** und Entwurfsmuster, die sie beheben (Auswahl aus [Gamma u.a.]):

1. **Objekte einer fest benannten Klasse erzeugen.**
Abstract Factory, Factory Method
2. **Abhängigkeit von spezieller Software-Plattform**
Abstract Factory, Bridge
3. **Abhängigkeit von speziellen Implementierungen**
Abstract Factory, Bridge
4. **Abhängigkeit von speziellen Algorithmen**
Strategy
5. **Zu enge Kopplung**
Abstract Factory, Bridge, Observer
6. **Funktionalität erweitern durch Vererbung**
Bridge, Observer, Strategy

Ziele:

Adaptierbarkeit durch Entkopplung

in der Vorlesung:

- Erläuterungen zu den Problemen,
- Hinweis auf die Muster die wir ansehen werden;
- Im Original sind die Listen der Gründe und der Muster zu jedem Grund länger; hier sind sie auf den Aspekt Adaptierbarkeit gekürzt.

nachlesen:

Budd: Understanding OOP with Java, Abschnitt 15

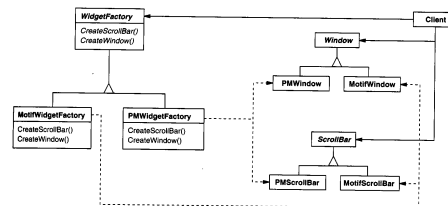
Abstract Factory (Beispiel)

Ziel: Objekte zu einer Gruppe zusammengehöriger Klassen erzeugen, ohne die Klassen konkret zu benennen

Beispiel:

Verschiedene GUI-Werkzeuge bieten unterschiedliche Ausprägungen von Sätzen von GUI-Komponenten an

Die Anwendung soll nicht durch **Klassennamen** auf eine bestimmte Ausprägung festgelegt werden



Eigenschaften des Beispiels:

- verschiedene Sätze von GUI-Klassen verfügbar machen
- Klassennamen nicht „fest verdrahten“, da schwer änderbar
- 2 Schnittstellen bereitstellen:
 - abstrakte GUI-Factory mit Ausprägungen für verschiedene GUI-Werkzeuge zur Generierung aller Produkte
 - Schnittstelle für jedes Produkt mit Ausprägungen für die GUI-Werkzeuge

Ziele:

Aufgabe und Muster verstehen

in der Vorlesung:

- Aufgabe und Struktur am Beispiel erläutern,
- Entkopplung zeigen,

nachlesen:

Budd: Understanding OOP with Java, Abschnitt 15.6

Abstract Factory (Muster)

OOP-3.4a

Ziel: Objekte zu einer Gruppe zusammengehöriger Klassen erzeugen, ohne die Klassen konkret zu benennen

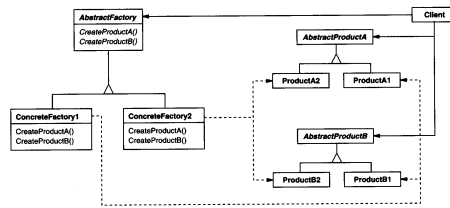
Muster:

AbstractFactory:
Schnittstelle zur Objekterzeugung

ConcreteFactory:
eine Ausprägung der Objekterzeugung

AbstractProduct:
Schnittstelle jeweils einer Produktklasse

ConcreteProduct:
Ausprägung jeweils einer Produktklasse



E. Gamma, R. Helm, R. Johnson, J. Vlissides: Design Patterns, Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995

Einsatzziele:

- Ein **System** soll mit einer von mehreren Produktgruppen **konfiguriert werden**.
- **Unabhängigkeit** von der Wahl der Produktgruppe.
- **Nur die Schnittstellen** nicht die Implementierungen der Produktgruppen **freigeben**.

© 2015 bei Prof. Dr. Uwe Kasrens

Vorlesung Objektorientierte Programmierung WS 2013/2014 / Folie 304a

Ziele:

Aufgabe und Muster verstehen

in der Vorlesung:

- Muster aus Beispiel verallgemeinern,
- Anwendbarkeitsbedingungen nennen.

nachlesen:

Budd: Understanding OOP with Java, Abschnitt 15.6

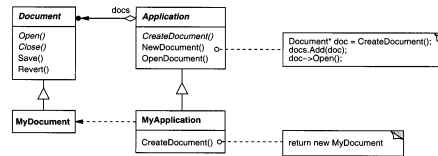
Factory Method (Aufgabe)

OOP-3.5

Ziel: Objekterzeugung zu einer Klasse von der Anwendung entkoppeln;
Anwendung realisiert ein **komplexes Konzept (Spezialisierung)**;
Objekterzeugung ist darin eine **Erweiterungsstelle**

Beispiel:

Ein Framework zur Präsentation von Dokumenten.
Die spezielle Ausprägung der Dokumente wird erst mit der speziellen Anwendung festgelegt.



E. Gamma, R. Helm, R. Johnson, J. Vlissides: Design Patterns, Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995

Eigenschaften des Beispiels:

- Anwendung (**Application**) benutzt ein abstraktes Produkt (**Document**), spezialisierte Anwendung (**MyApplication**) erzeugt das konkrete Produkt (**MyDocument**)
- **Application**: allgemeine Dokumentverwaltung, Dokument erzeugen, öffnen, etc. bestimmt **wann Objekte erzeugt** werden - **nicht welcher Klasse** sie angehören
- **konkrete Implementierung** der abstrakten Factory Method in **MyApplication** **bestimmt die Klasse**, z. B. Graphik-Objekte im Zeichenprogramm

© 2013 bei Prof. Dr. Uwe Kasrens

Vorlesung Objektorientierte Programmierung WS 2013/2014 / Folie 305

Ziele:

Aufgabe verstehen

in der Vorlesung:

- Aufgabe und Struktur am Beispiel erläutern,
- Entkopplung zeigen,

nachlesen:

Budd: Understanding OOP with Java, Abschnitt 15.7

Factory Method (Muster)

OOP-3.5a

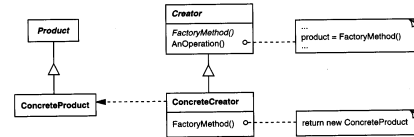
Muster:

Product:
Schnittstelle der zu erzeugenden Objekte

ConcreteProduct:
spezielle Ausprägung der Produkte

Creator:
Kontext, in dem die Objekte erzeugt werden;
hat Schnittstelle oder Default-Implementierung der FactoryMethod

ConcreteCreator:
implementiert oder überschreibt die FactoryMethod,
erzeugt ein ConcreteProduct-Objekt



E. Gamma, R. Helm, R. Johnson, J. Vlissides: Design Patterns, Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995

Einsatzziele:

- **Entscheidung** über Objekterzeugung (**welche Klasse?** ConcreteProduct) **verschieben** und in einer Klasse (ConcreteCreator) **kapseln**.
- Art von Delegationsobjekten (ConcreteProduct) in einer Klasse (ConcreteCreator) kapseln.

© 2013 bei Prof. Dr. Uwe Kastens

Bridge (Aufgabe)

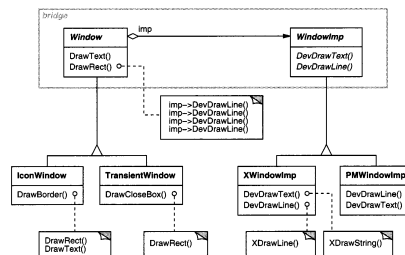
OOP-3.6

Ziel: Eine **Spezifikation wird von ihren Implementierungen entkoppelt**.
Verfeinerte Spezifikationen und **weitere Implementierungen** unabhängig zufügen.
2 Aspekte unabhängig variieren benötigt 2 Hierarchien, durch Delegation verbunden

Beispiel:

In einem GUI-Paket zu einer Window-Schnittstelle 2 Aspekte:

- Implementierungen für verschiedene Plattformen (XWindowImp, PMWindowImp)
- spezialisierte Schnittstellen (IconWindow, TransientWindow)



E. Gamma, R. Helm, R. Johnson, J. Vlissides: Design Patterns, Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995

VP Spezialisierung (von Schnittstellen) **VP Spezifikation** (von Implementierungen)

© 2013 bei Prof. Dr. Uwe Kastens

Vorlesung Objektorientierte Programmierung WS 2013/2014 / Folie 305a

Ziele:

Muster verstehen

in der Vorlesung:

- Muster aus dem Beispiel verallgemeinern,
- Anwendbarkeitsbedingungen nennen.

nachlesen:

Budd: Understanding OOP with Java, Abschnitt 15.7

Vorlesung Objektorientierte Programmierung WS 2013/2014 / Folie 306

Ziele:

Aufgabe verstehen

in der Vorlesung:

- Aufgabe und Struktur am Beispiel erläutern,
- Entkopplung zeigen,

nachlesen:

Budd: Understanding OOP with Java, Abschnitt 15.11

Bridge (Muster)

OOP-3.6a

Muster:

Abstraction:

definiert Schnittstelle,
hat Referenz auf ein Implementor-Objekt,
implementiert Methoden damit

RefinedAbstraction:

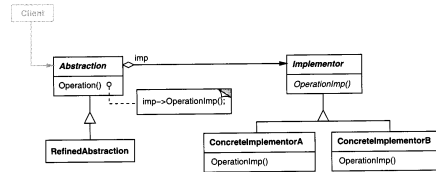
Implementor:

Schnittstelle für Implementierung,
ist i. a. elementarer als die der **Abstraction**

ConcreteImplementor:

Einsatzkriterien:

- **Feste Bindung** zwischen Abstraktion und Implementierung **vermeiden**
- benutzte **Implementierung zur Laufzeit wechseln**
- **2 Dimensionen der Variation** nicht in 1 Hierarchie!



E. Gamma, R. Helm, R. Johnson, J. Vlissides: Design Patterns, Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995

Vorlesung Objektorientierte Programmierung WS 2013/2014 / Folie 306a

Ziele:

Muster verstehen

in der Vorlesung:

- Muster aus dem Beispiel verallgemeinern,
- Anwendbarkeitsbedingungen nennen.

nachlesen:

Budd: Understanding OOP with Java, Abschnitt 15.11

Observer (Aufgabe)

OOP-3.7

Ziel: Observer-Objekte werden über Zustandsänderungen eines Subjekts informiert. Dynamisch veränderliche Zuordnung von Observern zum Subjekt.

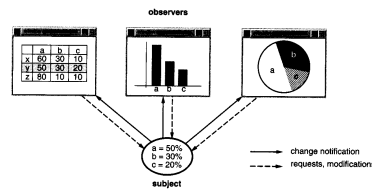
Beispiel:

In GUI-Paketen: Trennung verschiedener Varianten der Präsentation von der Anwendung, die die präsentierten Daten liefert.

MVC: Model / View / Controller
[Krasner, Pope, 1988, Smalltalk]

Eigenschaften des Beispiels:

- **Entkoppeln der Anwendung** (Model, Concrete Subject) von den **Präsentationen** (View, Observer)
- Software-**Aufgaben trennen**
- Präsentations-Software **separat variieren** (statisch)
- Präsentatoren **zur Laufzeit zufügen** und entfernen (dynamisch)
- Auch **andere** als GUI-Anwendungen:
z. B. eine Uhr und Prozesse, die die Zeit beobachten



E. Gamma, R. Helm, R. Johnson, J. Vlissides: Design Patterns, Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995

Vorlesung Objektorientierte Programmierung WS 2013/2014 / Folie 307

Ziele:

Aufgabe verstehen

in der Vorlesung:

- Aufgabe und Struktur am Beispiel erläutern,
- Entkopplung zeigen,

nachlesen:

Budd: Understanding OOP with Java, Abschnitt 15.4

Observer (Muster)

OOP-3.7a

Muster:

Subject:

Abstraktion; verwaltet Referenzen auf alle Observer
Operationen zu Benachrichtigen

Observer:

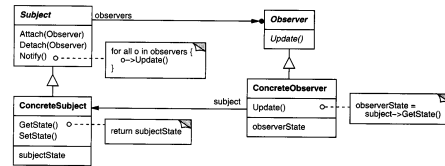
Update-Schnittstelle

ConcreteSubject:

Anwendung mit dem beobachteten Zustand

ConcreteObserver:

hat eine Referenz auf sein **concreteSubject**-Objekt,
hat einen Zustand, der mit dem **subject** konsistent gehalten wird,
implementiert die **update**-Schnittstelle



E. Gamma, R. Helm, R. Johnson, J. Vlissides: Design Patterns. Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995

Einsatzkriterien:

- **abhängige Aspekte** entkoppeln (Model -> View)
- Verbindung zu Beobachtern **dynamisch änderbar**
- **keine Annahmen** beim Benachrichtigen

© 2015 bei Prof. Dr. Uwe Kastens

Vorlesung Objektorientierte Programmierung WS 2013/2014 / Folie 307a

Ziele:

Muster verstehen

in der Vorlesung:

- Muster daraus verallgemeinern,
- Anwendbarkeitsbedingungen nennen.

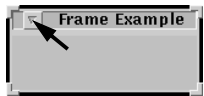
nachlesen:

Budd: Understanding OOP with Java, Abschnitt 15.4

Entwurfsmuster Observer im AWT-Paket von Java

OOP-3.7b

An AWT-Komponenten werden Ereignisse ausgelöst, z. B. ein WindowEvent an einem Frame-Objekt:



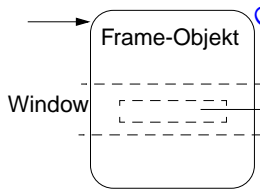
Subject



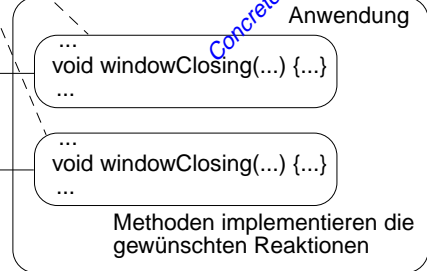
Observer

Beobachter für bestimmte Ereignistypen: Objekte von Klassen, die das zugehörige Interface implementieren

ConcreteObserver



ConcreteSubject
Beobachterobjekte dynamisch zuordnen



Anwendung

Methoden implementieren die gewünschten Reaktionen

Ereignis auslösen: zugehörige Methode in jedem Beobachter-Objekt aufrufen

Entwurfsmuster „Observer“: Unabhängigkeit zwischen den Beobachtern und dem Gegenstand wegen Interface und dynamischem Zufügen von Beobachtern.

Vorlesung Objektorientierte Programmierung WS 2013/2014 / Folie 307b

Ziele:

Anwendung des Musters Observer in bekanntem Kontext

in der Vorlesung:

Erläuterungen dazu

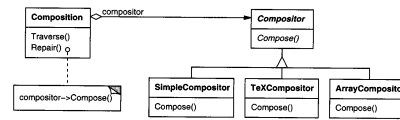
Strategy (Aufgabe)

OOP-3.8

Ziel: Für eine Gruppe unterschiedlicher Algorithmen für den gleichen Zweck Entwicklung entkoppeln und dynamisch austauschbar machen

Beispiel:

Algorithmen zum Zeilenumbruch:
Composition zeigt veränderbaren Text an.
Den Zeilenumbruch delegiert sie an ein **Compositor**-Objekt.
Eines mit den gewünschten Fähigkeiten wird installiert und ggf. ausgetauscht.



E. Gamma, R. Helm, R. Johnson, J. Vlissides: Design Patterns, Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995

weitere Beispiele:

- **LayoutManager** im AWT-Paket von Java
- Algorithmen zur Speicherzuteilung nach unterschiedlichen Verfahren
- Algorithmen zur Registerzuteilung nach unterschiedlichen Verfahren

Vorlesung Objektorientierte Programmierung WS 2013/2014 / Folie 308

Ziele:

Aufgabe verstehen

in der Vorlesung:

- Aufgabe und Struktur am Beispiel erläutern,
- Entkopplung zeigen,

nachlesen:

Budd: Understanding OOP with Java, Abschnitt 15.3

Strategy (Muster)

OOP-3.8a

Muster:

Strategy:

gemeinsame **Algorithmen-Schnittstelle (Spezifikation)**

ConcreteStrategy:

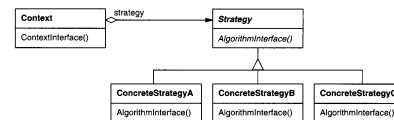
Varianten von **Algorithmen-Implementierungen**

Context:

wird mit einem **ConcreteStrategy**-Objekt konfiguriert und hat Referenz darauf (**Delegation**, wie bei Bridge); kann eine Schnittstelle anbieten, über die Algorithmen ihre Daten beziehen

Einsatzkriterien:

- Algorithmen separieren macht auch den Kontext einfacher, **Aufgabenzerlegung, Wiederverwendung**
- **nachträglich separieren ist aufwändig**
- Daten **kapseln**



E. Gamma, R. Helm, R. Johnson, J. Vlissides: Design Patterns, Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995

VP Spezifikation

Vorlesung Objektorientierte Programmierung WS 2013/2014 / Folie 308a

Ziele:

Muster verstehen

in der Vorlesung:

- Muster aus Beispiel verallgemeinern,
- Anwendbarkeitsbedingungen nennen.

nachlesen:

Budd: Understanding OOP with Java, Abschnitt 15.3