

# Design Patterns im Rahmen der Objekt Orientierten Programmierung

---

Hans Fangohr

2016-02-01

<http://github.com/uni-wuppertal/SE-designpattern>

Einführung

Beispiel

Entwurfsmuster (Design Patterns)

Zusammenfassung

Literatur

# Einführung

---

- Sie können die Idee von Design Pattern erklären,
- Sie sind mit mindestens zwei Design Pattern vertraut, und können diese an einem Beispiel erläutern,
- Sie fangen an, einen Überblick von klassischen Designpattern zu entwickeln, und
- Sie können Literatur benutzen, um selbständig weitere Designpattern zu erlernen.

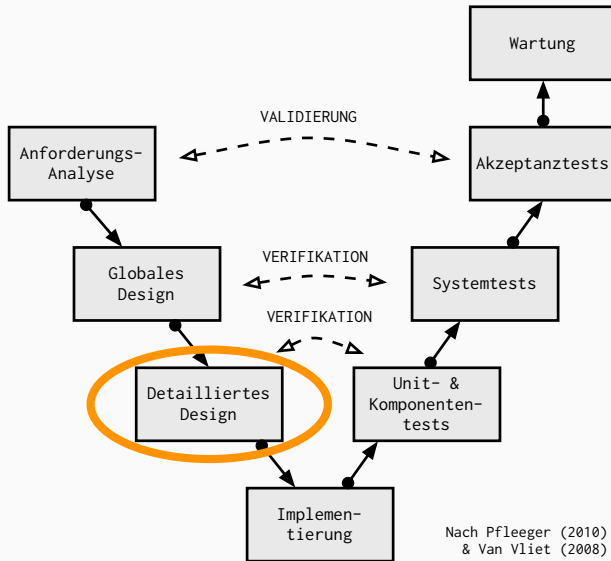
# Was ist ein Entwurfsmuster ("Design Pattern")?

Christopher Alexander (Architekt), 1977:

*Each pattern describes a problem that occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.*

- Alexander bezog sich auf Gebäude und Städte
- Wir sprechen über Objekte und Interfaces anstelle von Wänden und Türen
- Kernidee: Allgemeine Lösung für ein wiederkehrendes Problem in speziellem Kontext

# Wo im Softwarezyklus sind Design Pattern relevant?



# Was sind Design Patterns in der OO Programmierung?

## Designpatterns

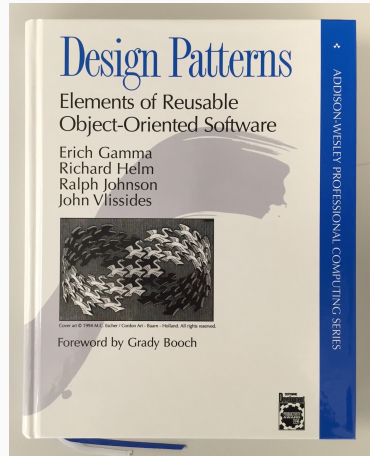
- sind generelle, wiederverwendbare Lösungen für häufig auftretende Entwurfsprobleme im Softwaredesign,
- beschreiben interagierenden Klassen und Objekten,
- stellen getestete Wissen zur Verfügung, und können den Entwurfprozess beschleunigen

## Design Patterns

- sind keine fertig Entwürfe, die nur noch implementiert werden müssen
- haben nichts mit Webdesign zu tun.

## *Design Patterns – Elements of Reusable Object-Oriented Software* (1995)

- Autoren werden oft als "Gang of Four" (GoF) beschrieben.
- Anfang der Design Patterns in Objektorientierter Programmierung





## Beispiel

---

## Beispiel: Sortierproblem 1/2

Anforderungen für FloatData Klasse:

- 5 Gleitkommazahlen speichern, mit Index 0 bis 4
- `swap(i, j)` Methode: zwei Zahlen mit Index `i` und `j` vertauschen
- `greater(i, j)` Methode: True falls Zahl mit Index `i` grösser als Zahl mit Index `j`
- `report()` Methode: Bericht mit Zahlen und Reihenfolge
- `sort()` Methode: sortiert die 5 Elemente

## Beispiel: Sortierproblem 1/2

Anforderungen für FloatData Klasse:

- 5 Gleitkommazahlen speichern, mit Index 0 bis 4
- `swap(i, j)` Methode: zwei Zahlen mit Index `i` und `j` vertauschen
- `greater(i, j)` Methode: True falls Zahl mit Index `i` grösser als Zahl mit Index `j`
- `report()` Methode: Bericht mit Zahlen und Reihenfolge
- `sort()` Methode: sortiert die 5 Elemente

<b>FloatData</b>
<code>-data:float[5]</code>
<code>+void swap(int, int)</code> <code>+bool greater(int, int)</code> <code>+void report(void)</code> <code>+void sort(void)</code>

# Beispiel: Sortierproblem 1/2

Anforderungen für FloatData Klasse:

- 5 Gleitkommazahlen speichern, mit Index 0 bis 4
- `swap(i, j)` Methode: zwei Zahlen mit Index `i` und `j` vertauschen
- `greater(i, j)` Methode: True falls Zahl mit Index `i` grösser als Zahl mit Index `j`
- `report()` Methode: Bericht mit Zahlen und Reihenfolge
- `sort()` Methode: sortiert die 5 Elemente

FloatData
-data:float[5]
+void swap(int, int) +bool greater(int, int) +void report(void) +void sort(void)

Example implementation (C++) in  
[https://github.com/uni-wuppertal/  
SE-designpattern/blob/master/Code/  
sort5-floatonly.cpp](https://github.com/uni-wuppertal/SE-designpattern/blob/master/Code/sort5-floatonly.cpp)

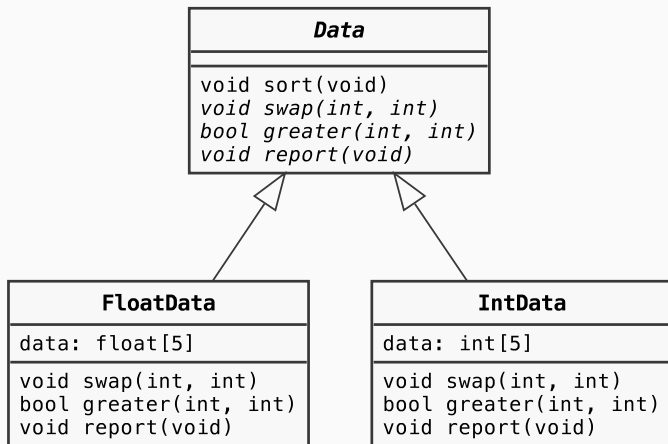
Zusatz: wir brauchen die gleichen Fähigkeiten für Integerdaten

<b>FloatData</b>
data: float[5]
void swap(int, int) bool greater(int, int) void report(void) void sort(void)

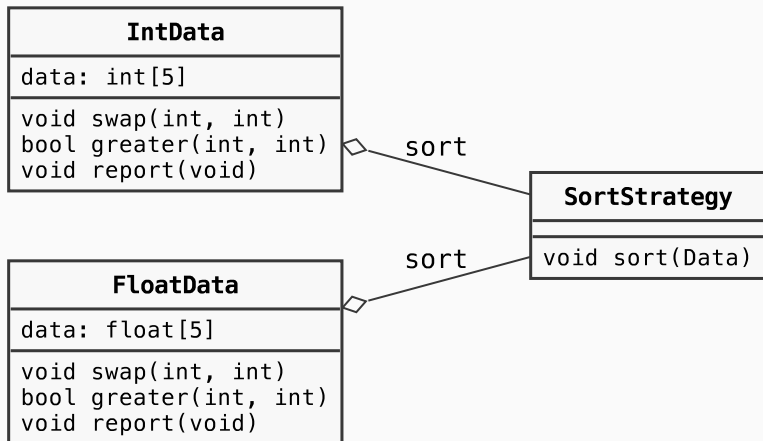
<b>IntData</b>
data: int[5]
void swap(int, int) bool greater(int, int) void report(void) void sort(void)

→ Duplizierung von Code

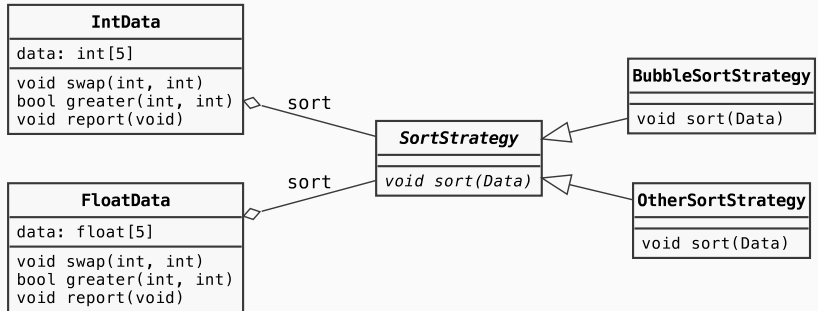
# Entwurfsmuster 1: Template Muster (Template Pattern)



## Entwurfsmuster 2: Strategie Muster (Strategy Pattern)

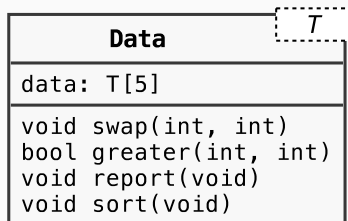


# Strategie Muster erlaubt verschiedene Strategien zu benutzen





## Alternative 3: Template Klasse (C++)



# Zusammenfassung Sortierproblem

## Templatemuster



Generischer code (in Basisklasse (sort)), abgeleitete Klassen für spezifische Details

## Strategiemuster



Benutze eine Klasse für generischen Sortiercode (die Strategie). Datenobjekte werden der Sortierklasse als Argument übergeben. Kann Strategie zur Laufzeit ändern.

## Templateklasse



Benutze C++ Templateklasse für dieses besondere Problem.

# Entwurfsmuster (Design Patterns)

---

# Tabelle Design Patterns

	<i>Erzeugende (Creational)</i>	<i>Strukturelle (Structural)</i>	<i>Verhalten (Behavioral)</i>
<i>Klassen- muster</i>	Factory Method	Adapter (class)	Interpreter Template Method
<i>Objekt- muster</i>	Abstract Factory Builder Prototype Singleton	Adapter (object) Bridge Composite Decorator Facade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

Tabelle 1.1 von GoF Buch

Gute deutsch-englisch Übersetzung:

[https://de.wikipedia.org/wiki/Entwurfsmuster\\_\(Buch\)](https://de.wikipedia.org/wiki/Entwurfsmuster_(Buch))

# Überblick Design Patterns

Eingeteilt in

- Erzeugungsmuster (creational patterns)
  - zB "Singleton": Eine Klasse für die nur eine einzige Instanz erzeugt wird (zB Logging, Datenbank, Hardware)
- Strukturmuster (structural patterns)
  - zB "Fassade": Eine Fassadenklasse agiert als vereinfachtes Interface zu komplexem Set von Klassen und Methoden
- Verhaltensmuster (behavioral patterns)
  - zB: "Mediator": Eine Vermittlerklasse agiert als Mittelsmann zwischen interagierenden Klassen, und erlaubt lose Kopplung zwischen den Klassen

Orthogonale Einteilung in

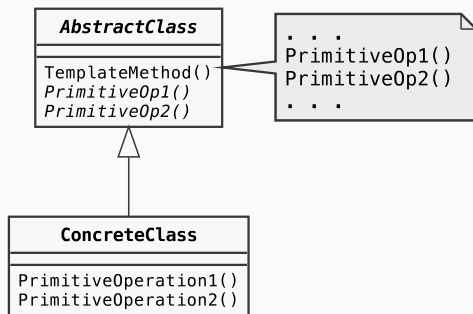
- Klassenmuster (Übersetzungszeit)
- Objektmuster (Laufzeit)

# Kernbestandteile eines Designpatterns

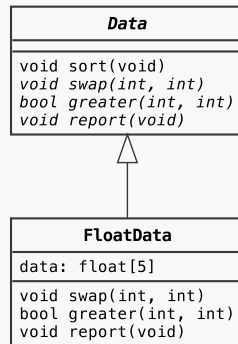
Design Pattern enthalten viel Information. Typische Beschreibung durch:

- Name des Pattern: möglichst beschreibend
- Struktur: UML Klassendiagramm
- Intention: Zusammenfassung
- Anwendbarkeit: beschreibt Situationen, in denen das Pattern verwendet werden kann
- Teilnehmer: beschreibt das Design und Zusammenwirken der Element des Patterns
- Konsequenzen: Vor- und Nachteile dieses Design Patterns
- Implementierung: (ggfs sprachspezifische) Hinweise zur Implementierung
- bekannte Anwendungsbeispiele

## Allgemeine Struktur (GoF)



## Unser Beispiel



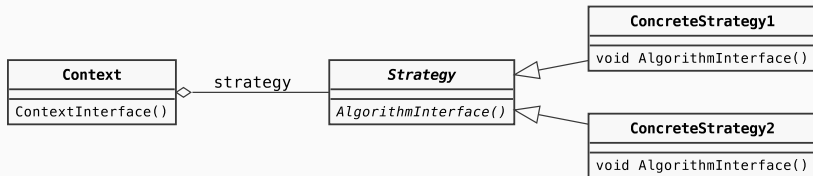
- Intention: Definiert das Skelett eines Algorithmus; lässt einige Schritte von Unterklassen ausführen.
- Anwendbarkeit:
  - implementiere invarianten Teil eines Algorithmus nur einmal, und Unterklassen implementieren abweichendes Verhalten
  - gemeinsamer Code in Subklassen soll reduziert werden ("refactor to generalize")



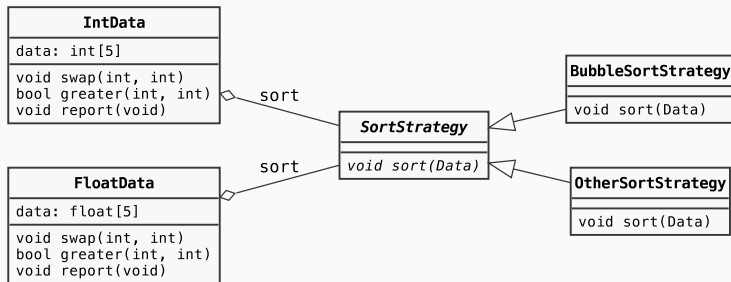
- Teilnehmer:
  - Abstrakte Klasse definiert abstrakte primitive Operationen, die von konkreten Unterklassen implementiert werden
  - Abstrakte Klasse definiert Templatemethode, die die primitiven Operation benutzt.
  - Konkrete Klasse: implementiert die primitiven Operationen die spezifisch für die Unterklasse sind
- Konsequenzen und Implementierung
  - Fundamentale Methode zur Wiederverwendung von Code
  - Unterklassen muss klar sein, welche Methoden überschrieben werden *können* und *müssen*.

# Strategiemuster 1/3

## Allgemeine Struktur Stragiemuster (GoF)



## Unser Beispiel



- Intention: Definiere eine Familie von Algorithmen, verberge jeden in einer Klasse, und mache sie austauschbar: das Strategiemuster erlaubt, den Algorithmus unabhängig vom Klienten, der ihn benutzt, zu ändern.
- Anwendbarkeit:
  - wenn viele Klassen sich nur in ihrem Verhalten unterscheiden. Das Strategiemuster erlaubt eine Klasse mit vielen unterschiedlichen Verhaltensweisen zu konfigurieren
  - wenn unterschiedliche Varianten eines Algorithmus gebraucht werden.
  - wenn eine Klasse vielfache Verhaltensweisen definiert, und diese als wiederholte Fallentscheidungen auftauchen. Zusammenhängende Codefragmente für ein Verhalten formen dann eine Strategieklass.

- Teilnehmer: (i) Strategieklassse (abstrakt) definiert Interface, (ii) konkrete Strategieklassen implementieren Algorithmus, (iii) Klientobjekt wird mit konkretem Strategieobjekt konfiguriert und stellt Interface für die Strategieklassse zur Verfügung.
- Konsequenzen:
  - Stragiemuster ist eine alternative zur Vererbung, und ist flexibler
  - Stragiemuster reduziert Fallunterscheidungen
  - Stragiemuster erlaubt verschiedene Implementierungen für das gleiche Verhalten (zB verschiedene Zeit- und Speicherbedarf)
  - Familien ähnlicher Algorithmen können hierarchisch geordnet werden
  - Klienten müssen Strategie wählen (und Strategien kennen)

# Design Patterns stellen gemeinsames Vokabular zur Verfügung

- Natürliche Sprache gibt uns Namen für häufig benötigte Dinge und Konzepte: Haus, Siedlung, Geld, Steuererklärung, Fallstudie, ...
- Design Patterns geben uns ein Vokabular für gängige Entwurfsmuster in der Softwaretechnologie
- Sehr hilfreich für Kommunikation, Dokumentation und Entwurfsdiskussionen:
  - *Könnten wir ein Strategiemattern verwenden, um die Anzahl der Fallentscheidungen zu reduzieren?*

- Ziel ist robuster, flexibler und wiederverwendbarer Code
- Unser Design muss Anforderungsänderungen in der Zukunft erlauben:
- Nur Code, der sich weiter entwickeln kann, kann wiederverwendet werden
- Monolithischer, unflexibler Code ist ein Risiko: erfordert möglicherweise grundlegende Änderungen / Neuentwicklung in der Zukunft
- Design Pattern unterstützen Flexibilität

*"Design for change"*

# Sind Design Patterns 'language smells' ?

- Design Patterns sind nützliche Muster, die im Allgemeinen nicht direkt von Programmiersprache unterstützt werden:
  - ausgehend von prozeduralen Sprachen würden wir über Muster wie Vererbung, Encapsulation and Polymorphismus reden.
  - Design Patterns in GoF Buch diskutieren Smalltalk/C++-level language (von 1995)
- Einige der Muster sind direkt verfügbar in weniger weit verbreiteten OO Sprachen
- Sprachen ändern sich, und integrieren Design Patterns im Laufe der Zeit
- Neue domänenspezifische Pattern entstehen (Web Design, Concurrency, Security, . . . )

# Zusammenfassung

---



- Einführung Design Patterns
- Überblick
- Template und Strategie Muster
- Design Patterns repräsentieren nützlichen Konzepte und Erfahrungen
- Design Patterns geben uns eine gemeinsam Sprache für diese Konzepte

## In der nächsten Übung

- Sie bekommen Code und den Auftrag, diesen zu verbessern, ohne die Funktionalität zu ändern.
- Wir empfehlen Designpattern, die Sie in Erwägung ziehen und verwenden können.

# Literatur

---

## **Design Patterns – Elements of Reusable OO Software**

Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides:

Addison-Wesley Professional Computing Series, 1994 [Use for Reference]

## **Head First Design Patterns**

Eric Freeman and Elisabeth Freeman

OReilley Publications

## **Videos**

Derek Banas, Design Patterns Video Tutorials

[https://youtu.be/vNHpsC5ng\\_E](https://youtu.be/vNHpsC5ng_E)

## **Vorlesungsmaterialien**

<http://github.com/uni-wuppertal/SE-designpattern>