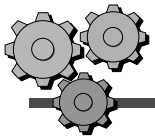


Entwurfsmuster

itelligence
Apcon Workplace Solutions
Member of itelligence

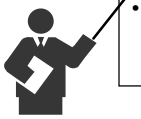
- ➔ Ursprung der Musterdiskussion
- ➔ Beschreibung von Mustern
- ➔ Nutzen von Mustern
- ➔ Entwurfsmuster von Gamma et al.

APCON Workplace Solutions

itelligence
Apcon Workplace Solutions
Member of itelligence

M**uster** (engl. **pattern**)

Ein Muster ist eine Abstraktion von einer konkreten Form, die wiederholt in bestimmten, nicht willkürlichen Kontexten auftritt.



- Der Musterbegriff ist hier sehr allgemein. Er ist weder auf Softwareentwicklung noch auf eine bestimmte Verwendung von Mustern zugeschnitten.
- Die (bekannte) Definition
  "A pattern is a solution to a recurring problem in a context"
  ist auf die Lösung von Entwurfsproblemen ausgerichtet.

Muster in der Literatur

Christopher **Alexander:**

"Each pattern describes a problem which 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, Ishikawa, Silverstein. A pattern language. Oxford University Press, 1977]

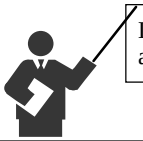
"Each pattern is a three part rule, which expresses a relation between a certain context, a problem, and a solution."
[Alexander. The timeless way of building. Oxford University Press, 1979]

Peter **Coad:**

A pattern is "a fully realized form, original, or model [...] for imitation". In object-orientation it is "a template of objects with stereotypical responsibilities and interactions".

[Coad. Object-oriented patterns. CACM 35, 9, 1992]

[Coad, Noth, Mayfield. Object models:Strategies, Patterns & Applications, Prentice-Hall 1995]

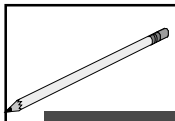


Die meisten Autoren im Bereich Design Patterns schließen sich Alexander an (z.B. Gamma et al., Beck, Johnson, Schmidt, Buschmann, Coplien).

Entwurfsmuster

Ein **Entwurfsmuster** (engl. **design pattern**)

- beschreibt abstrakt eine bewährte Lösung für ein bestimmtes und häufig wiederkehrendes Problem des objektorientierten Softwareentwurfs
- entsteht durch die Analyse und Überarbeitung vorhandener Designlösungen, setzt also Entwurfserfahrung voraus (Entwurfsmuster werden nicht “erfunden”, sondern entdeckt !)
- kann in seiner Struktur verstanden werden als eine Menge von Klassen, die festgelegte Verantwortlichkeiten haben und in einer definierten Vererbungs- bzw. Benutzungsbeziehungen zueinander stehen
- kann in seiner Dynamik verstanden werden als eine Menge von Objekten, die nach einem beschreibbaren Prinzip interagieren bzw. erzeugt werden
- kann immer nur zusammen mit dem Entwurfsproblem beschrieben werden, das es lösen soll.



Beschreibung von Entwurfsmustern

itelligence
Apcon Workplace Solutions
Member of itelligence

Eine **Musterbeschreibung** besteht meist aus den folgenden Teilen:

- dem **Namen** des Entwurfsmusters,
- dem **Problem**, das mit Hilfe des Musters gelöst werden soll,
- der **Kontext**, in dem sich das Problem stellt,
- der **Lösung**, mit der die Organisation von Klassen in einer Klassenhierarchie und die Gestaltung ihrer Interaktion vorgegeben werden,
- die (positiven und negativen) **Konsequenzen** der Musteranwendung.

Von Gamma et al. wird eine feiner ausgearbeitete Notation verwendet:

- Name
- Ziel
- Motivation
- Struktur
- Teilnehmer
- Zusammenarbeit
- Implementation
- Anwendbarkeit
- Konsequenzen

Zum Nutzen von Entwurfsmustern

itelligence
Apcon Workplace Solutions
Member of itelligence

Entwurfsmuster

- helfen, existierende Softwareentwürfe zu analysieren und zu reorganisieren
- erleichtern die Einarbeitung in Software-Architekturen (z.B. Klassenbibliotheken, Rahmenwerke), solange sie auf der Basis von bekannten Entwurfsmustern dokumentiert sind
- sind “Mikroarchitekturen”, die sich von erfahrenen Entwicklern als Bausteine innerhalb größerer Software-Architekturen wiederverwenden lassen (Wiederverwendung von Design-Lösungen statt Wiederverwendung von Code).
- stellen uns die Elemente einer Sprache, in der wir über Software-Architekturen nachdenken und kommunizieren können.
- sollen die softwaretechnische Qualität von Entwürfen erhöhen (z.B. ihre Wiederverwendbarkeit und Erweiterbarkeit).

Unterteilung von Entwurfsmustern

itelligence
Apcon Workplace Solutions
Member of itelligence

Der **allgemeine Teil eines Entwurfsmusters**:

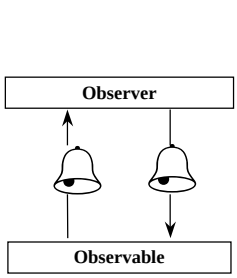
- beschreibt die abstrakten Modellierungselemente und Bezüge für den softwaretechnischen Entwurf,
- beschreibt die zusammenarbeitenden Objekte und Klassen, die maßgeschneidert sind, um ein allgemeines Entwurfsproblem in einem bestimmten Kontext zu lösen,
- muß durch unterschiedliche Konstruktionsansätze konkretisiert werden.

Ein **Konstruktionsansatz** als der konstruktive Teil eines Entwurfsmusters:

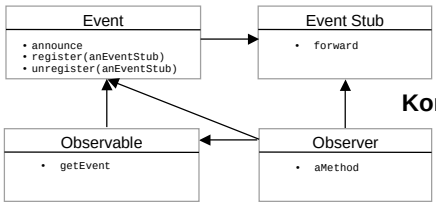
- beschreibt die konkreten Elemente und Bezüge des softwaretechnischen Entwurfs,
- ist i.d.R. eine Implementationsvariante, die einen Aspekt des Musters betont und auf eine bestimmte Programmiersprache zugeschnitten ist.
- kann unmittelbar in eine programmiersprachliche Implementation umgesetzt werden.

Zusammenhang Entwurfsmuster und Konstruktionsansatz

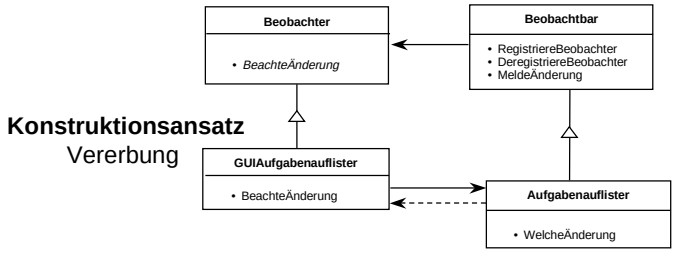
itelligence
Apcon Workplace Solutions
Member of itelligence



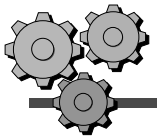
Entwurfsmuster
Beobachter



Konstruktionsansatz
Event-Objekte



Konstruktionsansatz
Vererbung



Schablonenmethode (Template methode)

Apcon Workplace Solutions
Member of itelligence

- ➔ Vorstellung des Beispiels
- ➔ Schablonenmethode
- ➔ Programming to an interface

APCON Workplace Solutions

Motivation: 1. Anforderung

Apcon Workplace Solutions
Member of itelligence

Anforderung

- Es soll eine kleine Applikation erstellt werden, welche eine **ASCII-Datei** auf der Standardausgabe ausgibt.
- Bevor die erste Zeile ausgegeben wird, soll der Text "Jetzt geht's los" ausgegeben werden.
- Nach der Ausgabe der letzten Zeile der Text "Das war's dann".

Lösung: 1. Anforderung


```
class Application {
  public static void main (String [] args) {
    try {
      FileInputStream stream = new FileInputStream ("Application.java");
      int ch = stream.read ();
      System.out.println ("Jetzt geht't los");
      while(ch != -1) {
        System.out.print ((char) ch);
        ch = stream.read ();
      }
      System.out.println ("Das war's dann");
      stream.close ();
    }
    catch (Exception e) { System.out.println (e); }
}
```

Motivation: 2. Anforderung

Anforderung

- Es soll eine kleine Applikation erstellt werden, welche die **Größe** einer Datei bestimmt.
- Alle Zeichen sollen mitgezählt werden.

Lösung: 2. Anforderung

itelligence
Apcon Workplace Solutions
Member of itelligence

```
class Application {
  public static void main (String [] args) {
    try {
      FileInputStream stream = new FileInputStream ("Application.java");
      int ch = stream.read ();
      int count = 0;
      while(ch != -1) {
        count ++;
        ch = stream.read ();
      }
      System.out.println (count);
      stream.close ();
    }
    catch (Exception e) { System.out.println (e); }
  }
}
```

Musterartiges Vorgehen

itelligence
Apcon Workplace Solutions
Member of itelligence

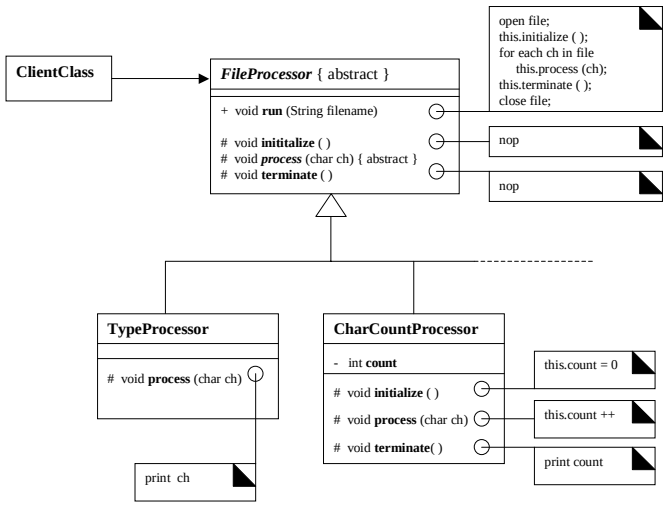
Allgemeinen Form der Applikationen

- Eingabe öffnen
- Initialisierung
- Zeichen lesen
- Zeichen verarbeiten

Es gibt jeweils einen **spezifischen Inhalt** für die allgemeine Form.

Trennen der abstrakten Form vom spezifischen Inhalt

Schablonenmethode: FileProcessor



- Warum sind die Einschubmethoden `protected` und nicht `public` wie die Schablonenmethode?

Lösung: 2. Anforderung

```
public class Application {

  public static void main (String [] args) {
    try {

      String filename = "Application.java";

      FileProcessor fp = new TypeProcessor();
      fp.run (filename);

      fp  = new CharCountProcessor();
      fp.run (filename);

    }
    catch (Exception e) {
      System.out.println (e);
    }
  }
}
```

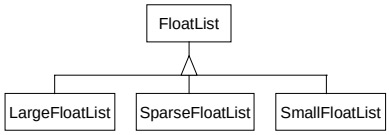
Die konkreten Objekte müssen **erzeugt** werden, obwohl nur auf der allgemeinen Schnittstelle gearbeitet wird.

- Welches Entwurfsmuster kann hier helfen?

"Programming to an interface, not an implementation"

itelligence
Apcon Workplace Solutions
Member of itelligence

```
public abstract class FloatList
{
  public abstract void append(float f);
  public abstract void start();
  public abstract void next();
  public abstract boolean empty();
  public abstract boolean off();
  public abstract float item();
}
```




```
public class LargeFloatList extends FloatList
{
    -- use a representation in a file
    ...
}
```

```
public class SparseFloatList extends FloatList
{
    -- store only values != 0
    ...
}
```

```
public class SmallFloatList extends FloatList
{
    -- store incore

  public void append(float f) {...};
  public void start() {...};
  public void next() {...};
  public boolean empty() {...};
  public boolean off() {...};
  public float item() {...};
}
```

Verschiedene **Implementationen** können durch Vererbung mit der entsprechenden **Spezifikation** verbunden werden.

Benutzung einer abstrakten Klasse

itelligence
Apcon Workplace Solutions
Member of itelligence

```
public abstract class FloatList
{
  public abstract void append(float f);
  public abstract void start();
  public abstract void next();
  public abstract boolean empty();
  public abstract boolean off();
  public abstract float item();
}
```

```
public class Client
{
  public void showList (FloatList aList)
  {
     aList.start();

     while (!aList.off())
     {
       System.out.println(aList.item());
       aList.next();
     }
  }
}
```

```
Client c = new Client();
FloatList aSmall = new SmallFloatList();
c.showList(aSmall);
```

- Es können keine Objekte von der **abstrakten Basisklasse** `FloatList` erzeugt werden.
- Der Klassenname kann aber als **Typname** in Variablendeklarationen verwendet werden.
- Kunden kennen nur die abstrakte Klasse `FloatList`. Sie brauchen weder etwas über die Implementationsklassen zu wissen, noch darüber, welches **konkrete Objekt** sie gerade benutzen.
- Implementationen lassen sich dann jederzeit **austauschen**, ohne daß ein Kunde der Klasse `FloatList` davon betroffen ist.

Schablonenmethode: FloatList

itelligence
Apcon Workplace Solutions
Member of itelligence

```
public abstract class FloatList
{
  public abstract void append(float f);
  public abstract void start();
  public abstract void next();
  public abstract boolean empty();
  public abstract boolean off();
  public abstract float item();

  public boolean contains(float f)
  {
    boolean ergebnis;
    if (empty())
    {
       ergebnis = false;
    }
    else
    {
      start();
      while (!off() && (item() != f))
      {
        next();
      }
      ergebnis = !off();
    }
    return ergebnis;
  }
}
```

`Contains()` kann von der abstrakten Basisklasse `FloatList` unter Rückgriff auf ihre aufgeschobenen Operationen für die gesamte Klassenfamilie der Float-Listen implementiert werden.

```
FloatList aSmall = new SmallFloatList();
aSmall.append(5.0);
aSmall.append(6.0);
aSmall.append(5.8);
...

if (aSmall.contains(6.0))
{
   ...
}
```

Schablonenmethode mit Einschubmethoden

itelligence
Apcon Workplace Solutions
Member of itelligence

Zweck

"Definiere das Skelett eines Algorithmus in einer Operation und delegiere einzelne Schritte an Unterklassen. Die Verwendung einer Schablonenmethode ermöglicht es Unterklassen, bestimmte Schritte eines Algorithmus durch Einschubmethoden zu überschreiben, ohne seine Struktur zu verändern."

Anwendbarkeit

- Die invarianten Teile eines Algorithmus können genau einmal festgelegt werden. Es ist den Unterklassen überlassen, das variierende Verhalten zu implementieren.
- Gemeinsames Verhalten von Unterklassen kann herausfaktoriert werden und in einer allgemeines Klasse plaziert werden, um die Verdopplung von Code zu vermeiden.
- Erweiterungen an Unterklassen können kontrolliert werden, indem eine Schablonenmethode so definiert ist, daß sie Einschubmethoden an bestimmten Stellen aufruft und somit Erweiterungen nur an diesen Stellen zugelassen werden.

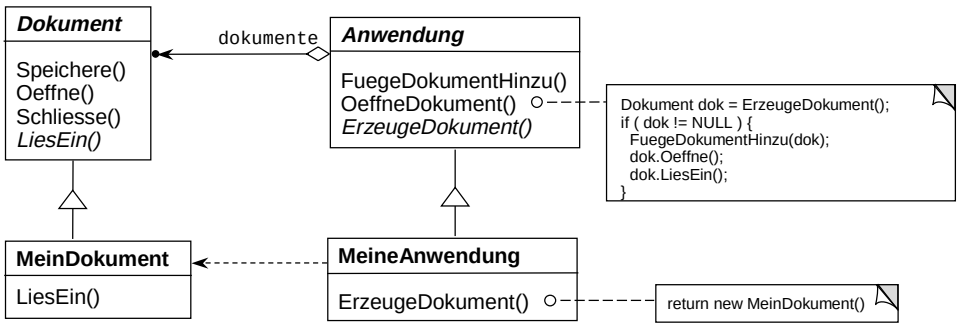
Diskussion des Schablonenmusters

itelligence
Apcon Workplace Solutions
Member of itelligence

- **Grundlegende Technik** zur Wiederverwendung, die besonders für Rahmenwerke wichtig ist.
- Der **Kontrollfluß** wird **invertiert**, so daß die Oberklasse die Operationen einer Unterklasse aufruft und nicht umgekehrt. („Don‘t call us, we call you“ Hollywood-Prinzip)
- Für den Programmierer von Unterklassen muß ersichtlich sein, **welche Operationen** der Oberklasse Einschubmethoden sind.
- Die **Semantik der Einschubmethoden** muß dem Programmierer von Unterklassen zugänglich sein, damit er diese sinnvoll implementieren kann. => Vertragsmodell
- Muß in der Unterklasse eine Operation der Oberklasse **überschrieben** werden, so sollte die Unterklasse die Operation der Oberklasse lediglich **erweitern**, indem sie die Operation der Oberklasse explizit aufruft und dann ihre Erweiterung macht.

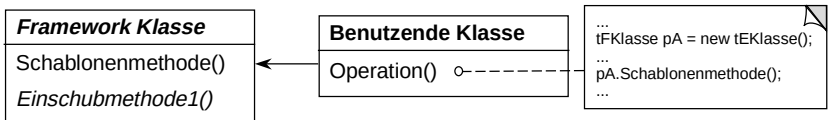
Beispiel für eine Schablonenmethode

itelligence
Apcon Workplace Solutions
Member of itelligence



Benutzen vs. Vererben

Apcon Workplace Solutions
Member of itelligence



- Bei Framework-Klassen kann man oft zwei Schnittstellenteile unterscheiden: das **calling API** und das **subclassing API**.
- Das calling API ist der Teil der Schnittstelle, der von anderen Klassen **benutzt** wird, um Dienste in Anspruch zu nehmen. Es entspricht dem **öffentlichen Teil** der Schnittstelle, den alle Klassen bieten, die keine speziellen Vorkehrungen zu ihrer Spezialisierung treffen.
- Das subclassing API hingegen besteht aus den Operationen, die **zur Spezialisierung** der Klasse vorgesehen sind. Dies sind die abstrakten Operationen, aber auch andere Operationen, die **redefiniert** bzw. **implementiert** werden können / sollen. Zur Laufzeit wird die passende Operation **dynamisch gebunden**.