# Muster in der Software–Technik

## Was macht ein Muster aus?
## Schemata, Klassifizierung und Mustersprachen

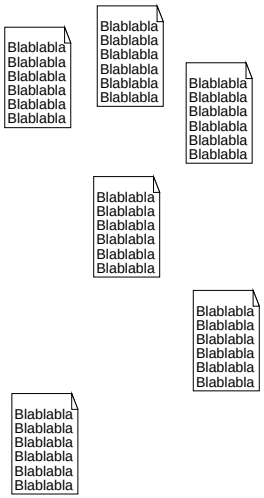
Dipl.-Inofrmatiker Adriano Gesué

**Themen der heutigen Vorlesung:**

- Organisatorisches
- Inhaltliches
- Literatur

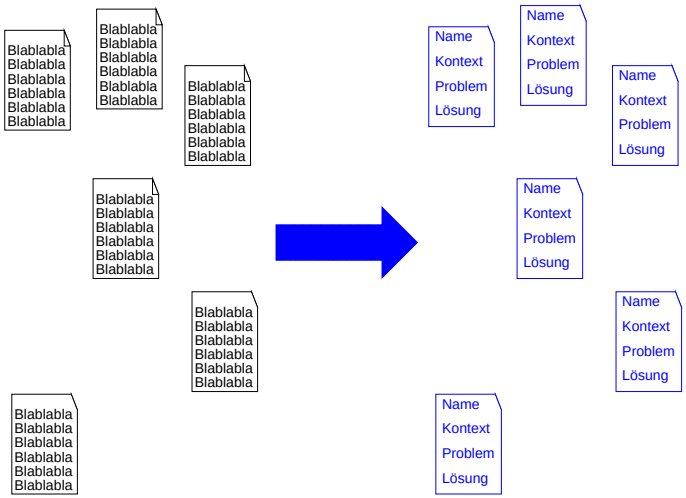
## Inhalte der heutigen Vorlesung:

- Was macht ein Muster aus?



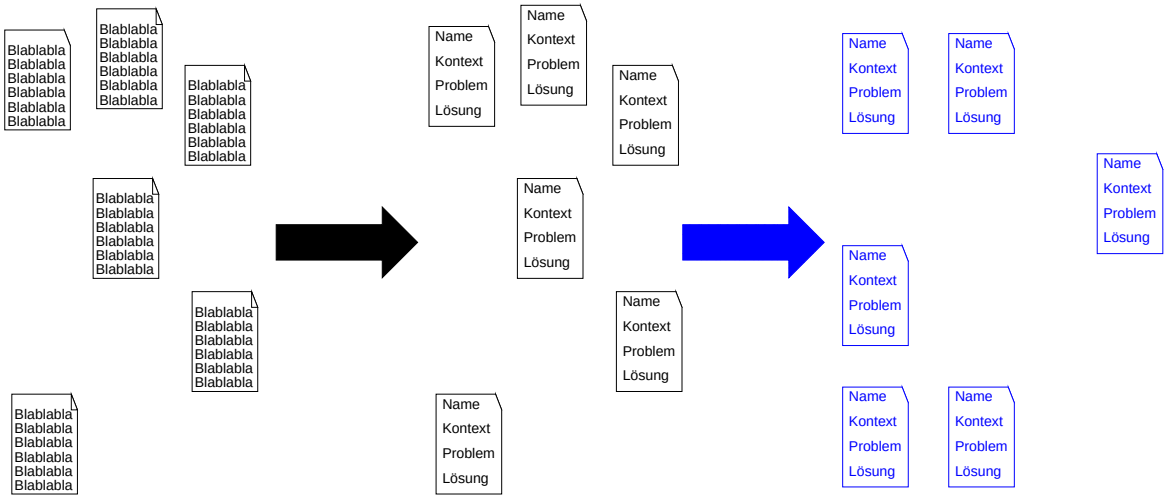
## Inhalte der heutigen Vorlesung:

- Was macht ein Muster aus?
- Strukturierung von Mustern: Schemata



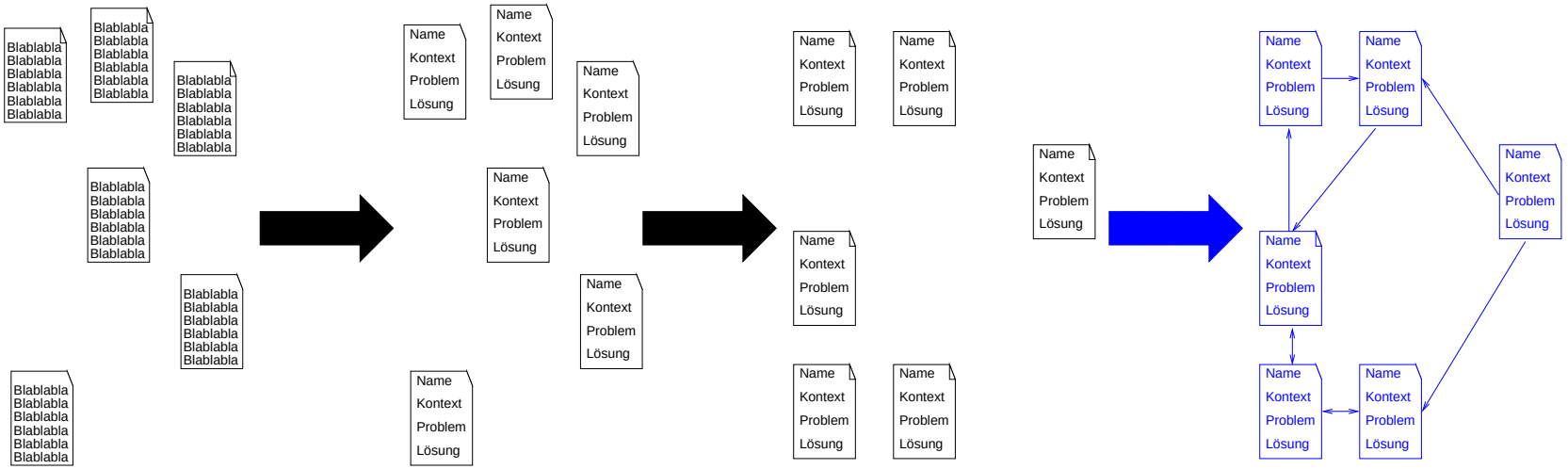
## Inhalte der heutigen Vorlesung:

- Was macht ein Muster aus?
- Strukturierung von Mustern: Schemata
- Klassifizierung von Mustern: Mustersystem und Referenzmodell



## Inhalte der heutigen Vorlesung:

- Was macht ein Muster aus?
- Strukturierung von Mustern: Schemata
- Klassifizierung von Mustern: Mustersystem und Referenzmodell
- Kombination von Mustern: Mustersprache



**Was macht ein Muster aus?**

Muster sind abstrahierte Repräsentationen erfolgreich eingesetzten Problemlösungswissens

- **abstrahiert**: Muster beziehen sich auf wiederkehrende Probleme
- **erfolgreich**: die Kenntnis von Mustern erspart es einem, das Rad immer wieder selbst erfinden zu müssen
- **eingesetzt**: Muster in der Software-Technik sind damit „das Zunftbuch für Software–Entwickler“
- **Problemlösungswissen**: Muster geben einen Vorschlag für effiziente / elegante Lösungen der Problemstellung

## Entartete Formulierung eines Musters

Gelegentlich kommt es vor, dass eine bereits existierende Klasse, wie z.B. aus einer Klassenbibliothek, in einem System verwendet werden soll, aber der Entwurf des Systems es nicht erlaubt, die Klasse direkt zu benutzen: Der Grund dafür kann sein, dass die Schnittstellen der Bibliotheksklasse und des entworfenen Systems nicht zueinander passen. Um beide Schnittstellen miteinander zu verbinden, ist es erforderlich, eine zusätzliche Komponente zu definieren, die die Anpassung beider Schnittstellen vornimmt. Diese Komponente sei Adapter genannt. Zur Realisierung dieser Komponente, die Eigenschaften der Bibliotheksklasse wieder– bzw. weiterverwendet, kann entweder direkte Implementierungsvererbung von der Bibliotheksklasse oder Delegation an eine Instanz der Bibliotheksklasse bei jeweils gleichzeitiger Implementierung der jeweiligen Dienstschnittstelle des entworfenen Systems verwendet werden.

**Strukturierung von Mustern:**

- Das Beispiel weist kaum Struktur auf.
- Zum Verständnis, worum es geht, muss der gesamte Text gelesen werden
- Einzelne Bestandteile sind schwer erkennbar bzw. die Auflösung ist evtl. gar nicht möglich
- Strukturierung bietet Vorteile
  - Schnellere Aufnahme, was das Wesentliche ist, und ob es interessant ist für den Leser
  - einheitliches Format erhöht Vergleichbarkeit von Mustern
  - klare Trennung der Bestandteile erhöht die Lesbarkeit

**Schema zur Beschreibung von Mustern:**

- **Name**: treffende Bezeichnung für die Essenz des Musters
- **Kontext**: In welchem Umfeld ist das Muster überhaupt relevant und einsetzbar?
- **Problem**: Welches Problem kann im gegebenen Kontext auftauchen und bedarf einer Lösung?
- **Lösung**: Wie kann das Problem gelöst werden? Welche Bestandteile sind nötig und welche Interaktionen finden zwischen diesen statt? Häufig begleitet von Klassendiagrammen und/oder Sequenzdiagrammen

**Schema zur Beschreibung von Mustern (Fortsetzung):**

- **Beispiel**: Ein Beispiel, bei dem Kontext und Problem gut erkennbar sind und die Lösung aufgezeigt wird.
- **Implementierungsvorschlag**: Wie kann das Muster implementiert werden (in bestimmten Sprachen)?
- **Varianten**: Gibt es Varianten der Lösung oder der Problemstellung und wie sehen diese aus?
- **Gegenanzeigen**: Wann empfiehlt es sich nicht, das Muster einzusetzen? Welche negativen Folgen könnte der Einsatz mit sich bringen?

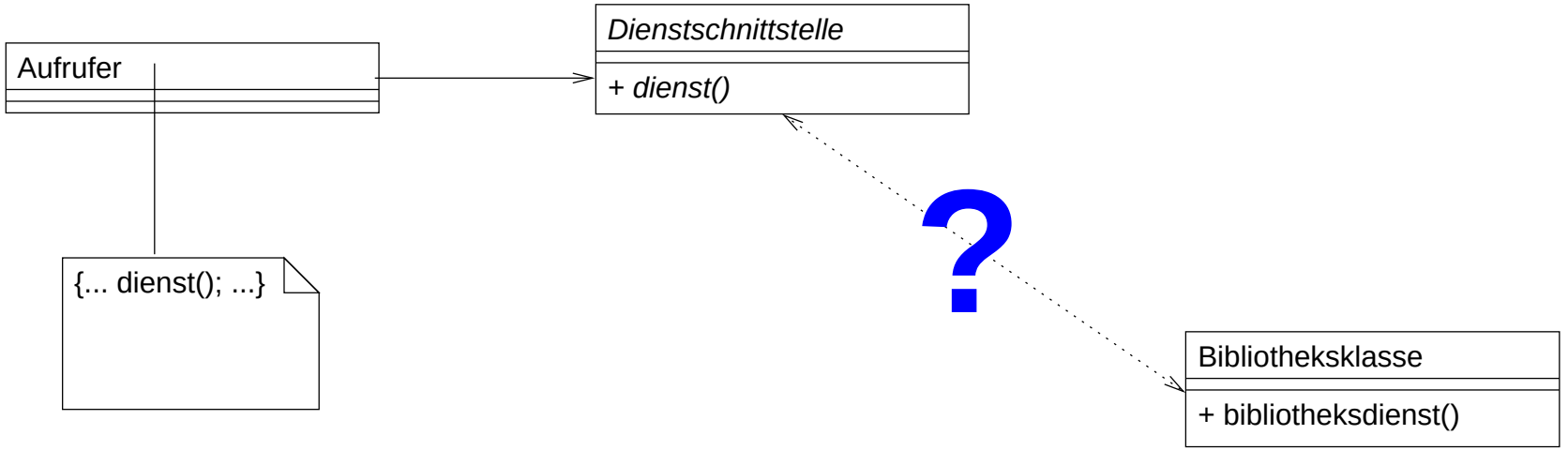
**Schema zur Beschreibung von Mustern:**

- **Name**
- **Kontext**
- **Problem**
- **Lösung**
- **Beispiel**
- **Implementierungsvorschlag**
- **Varianten**
- **Gegenanzeigen**

**Schematisiertes Muster: Adapter**

- **Name:** Adapter (auch Wrapper)
- **Kontext:** Bibliotheksklassen besitzen eine Schnittstelle, die vom Benutzer nicht veränderbar ist, weil der Quellcode im Allgemeinen nicht vorliegt.
- **Problem:** Die Eigenschaften einer Bibliotheksklasse sollen genutzt werden, unser Entwurf verwendet allerdings eine andere Schnittstelle, als die von der Bibliothek bereitgestellte. Die Aufrufe an die Schnittstelle passen nicht zu den von der Bibliotheksklasse bereitgestellten.

# Problem des Adapter–Musters: Klassendiagramm



**Schema zur Beschreibung von Mustern:**

- **Name**
- **Kontext**
- **Problem**
- **Lösung**
- **Beispiel**
- **Implementierungsvorschlag**
- **Varianten**
- **Gegenanzeigen**

**Lösung** für das Adapter–Muster:

Definiere einen **Adapter**, der die Aufrufe, die in unserem System stattfinden, so umsetzt, dass die Eigenschaften der Bibliotheksklasse verwendet werden können.

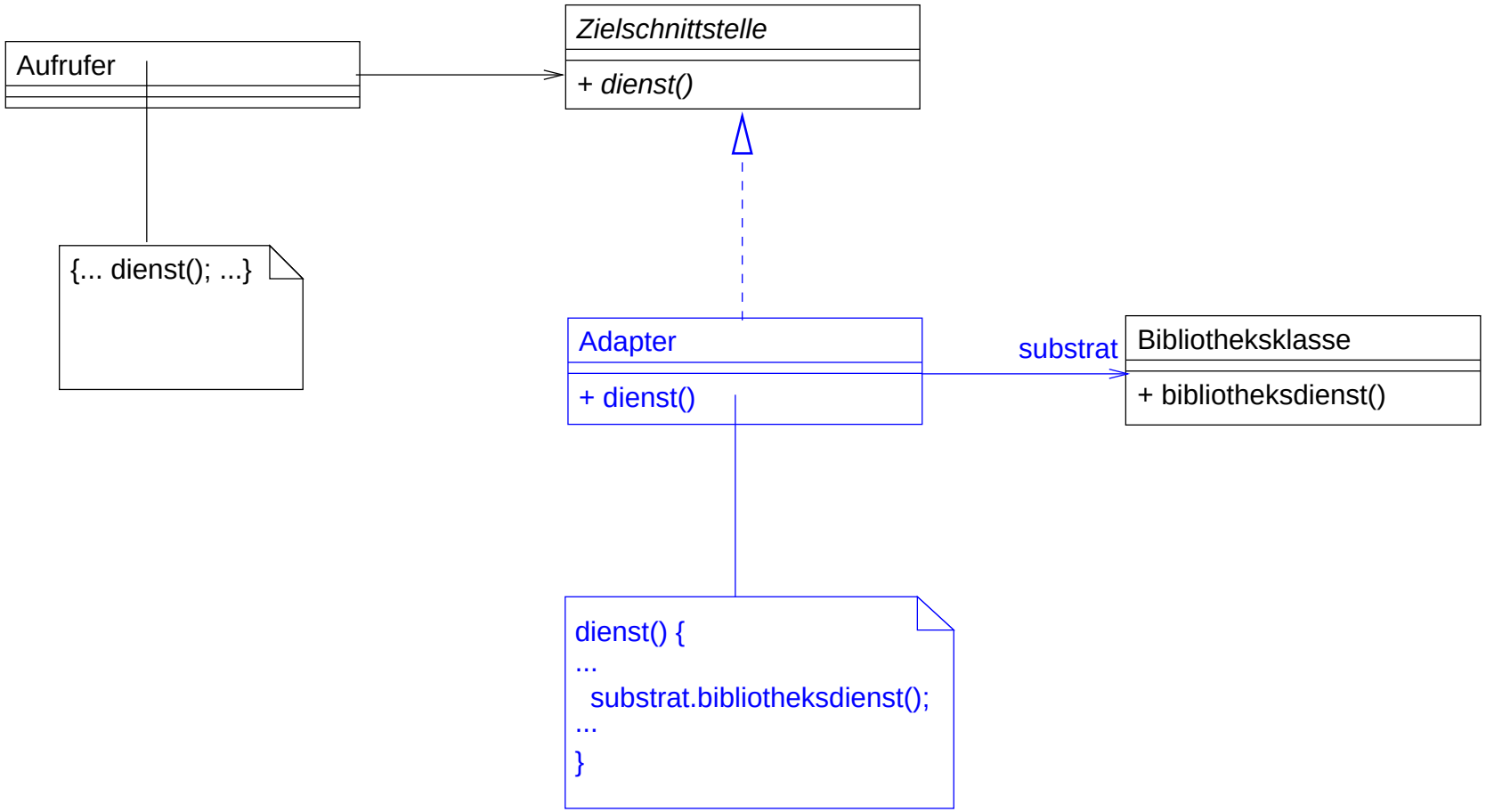
- Der Adapter muss die von System verwendete Dienstschnittstelle implementieren und
- gleichzeitig die Eigenschaften der Bibliotheksklasse wiederverwenden:
  - Delegation an ein Objekt der Bibliotheksklasse: **Objektadapter**
  - Implementierungsvererbung von der Bibliotheksklasse: **Klassenadapter**
- innerhalb der Implementierung des Schnittstellendienstes wird der Bibliotheksdienst verwendet.

**Lösung** für das Adapter–Muster (Fortsetzung):

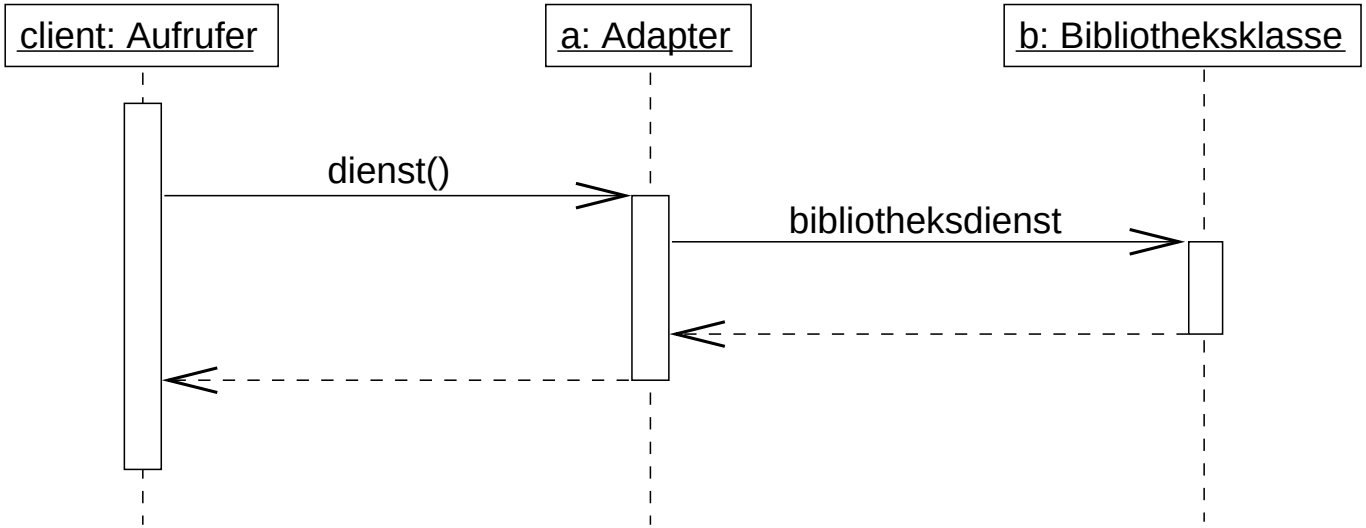
Diese kurze Beschreibung der benötigten Bestandteile des Musters und der Schritte, um zu einer Lösung zu kommen, kann noch graphisch verdeutlicht werden:

- strukturelle Aspekte: beteiligte Klassen und Assoziationen dazwischen
  Klassendiagramm
- dynamische Aspekte: Interaktion zwischen beteiligten Objekten
  Sequenzdiagramm

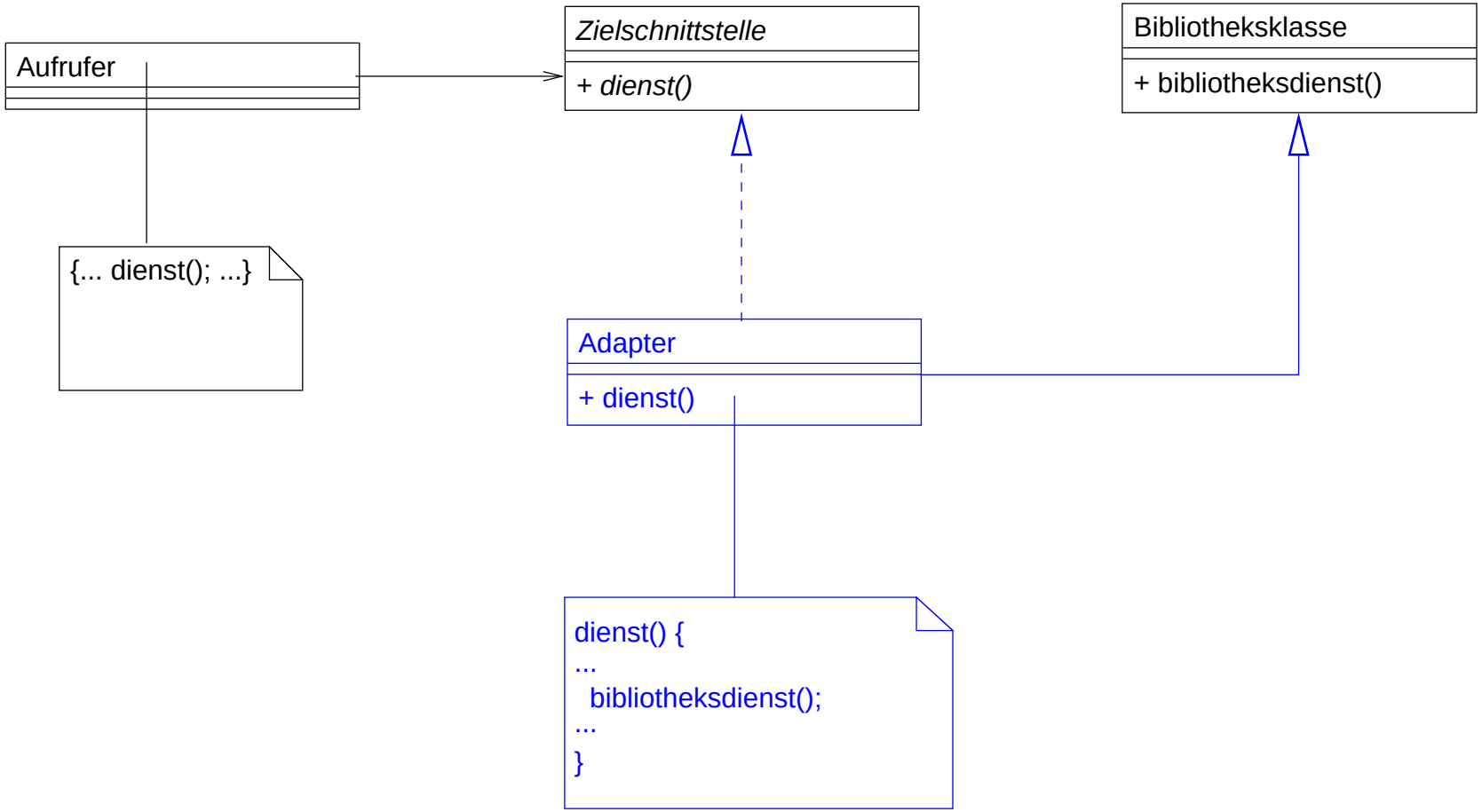
# Lösung des Adapter–Musters: Klassendiagramm für Objektadapter



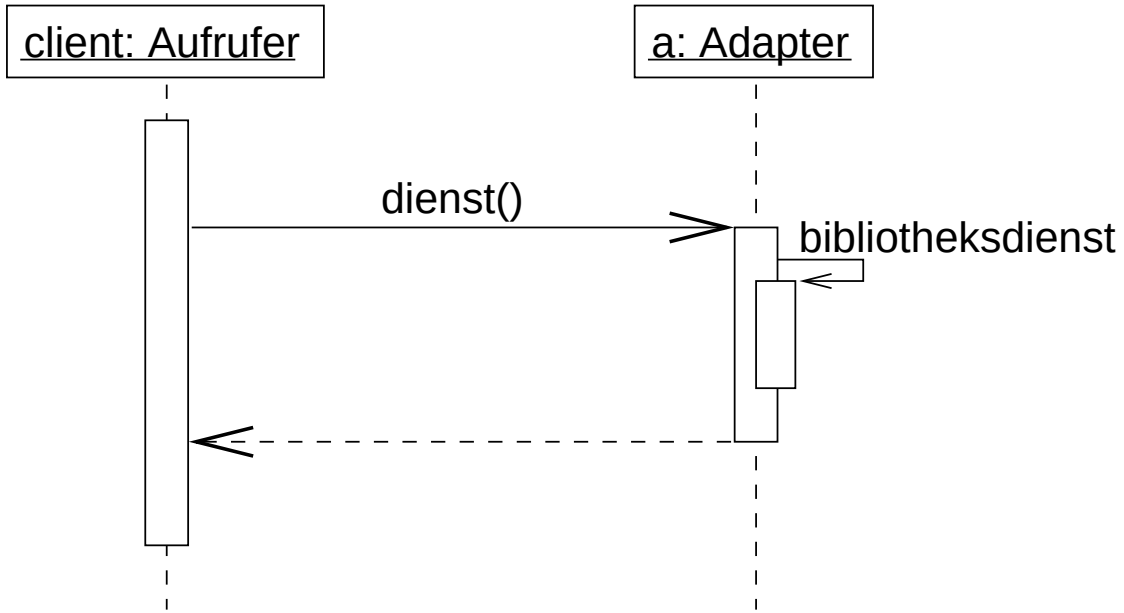
# Lösung des Adapter–Musters: Sequenzdiagramm für Objektadapter



# Lösung des Adapter–Musters: Klassendiagramm für Klassenadapter



# Lösung des Adapter–Musters: Sequenzdiagramm für Klassenadapter



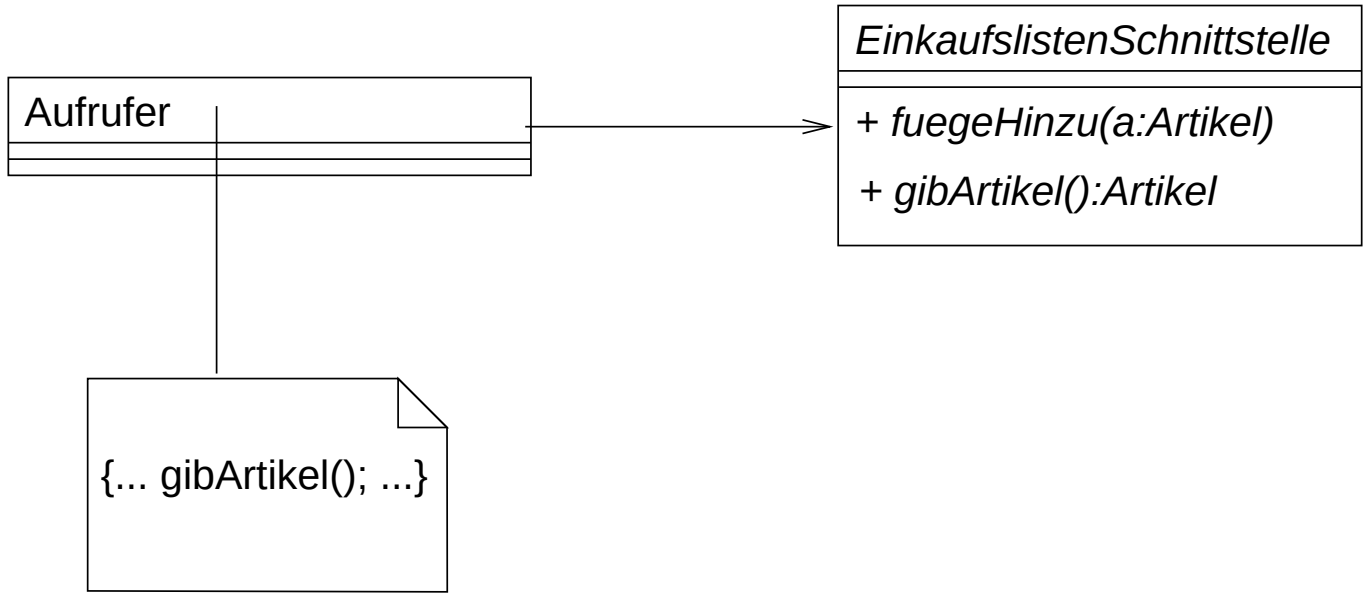
**Schema zur Beschreibung von Mustern:**

- **Name**
- **Kontext**
- **Problem**
- **Lösung**
- **Beispiel**
- **Implementierungsvorschlag**
- **Varianten**
- **Gegenanzeigen**

**Beispiel** für das Adapter–Muster: Einkaufsliste

- Angenommen, wir möchten die Haushaltsführung durch ein Software–System unterstützen: Zu den häufigeren Tätigkeiten dabei gehört der Einkauf der benötigten Dinge. Dazu ist eine Einkaufsliste hilfreich.
- In unserem Unterstützungssystem programmieren wir nun gegen eine Schnittstelle, die typische Operationen für Einkaufslisten enthält.
- Wir können diese Einkaufsliste nun selbst implementieren oder bereits existierende Klassen verwenden, die typische Operationen für Sammlungen realisieren.

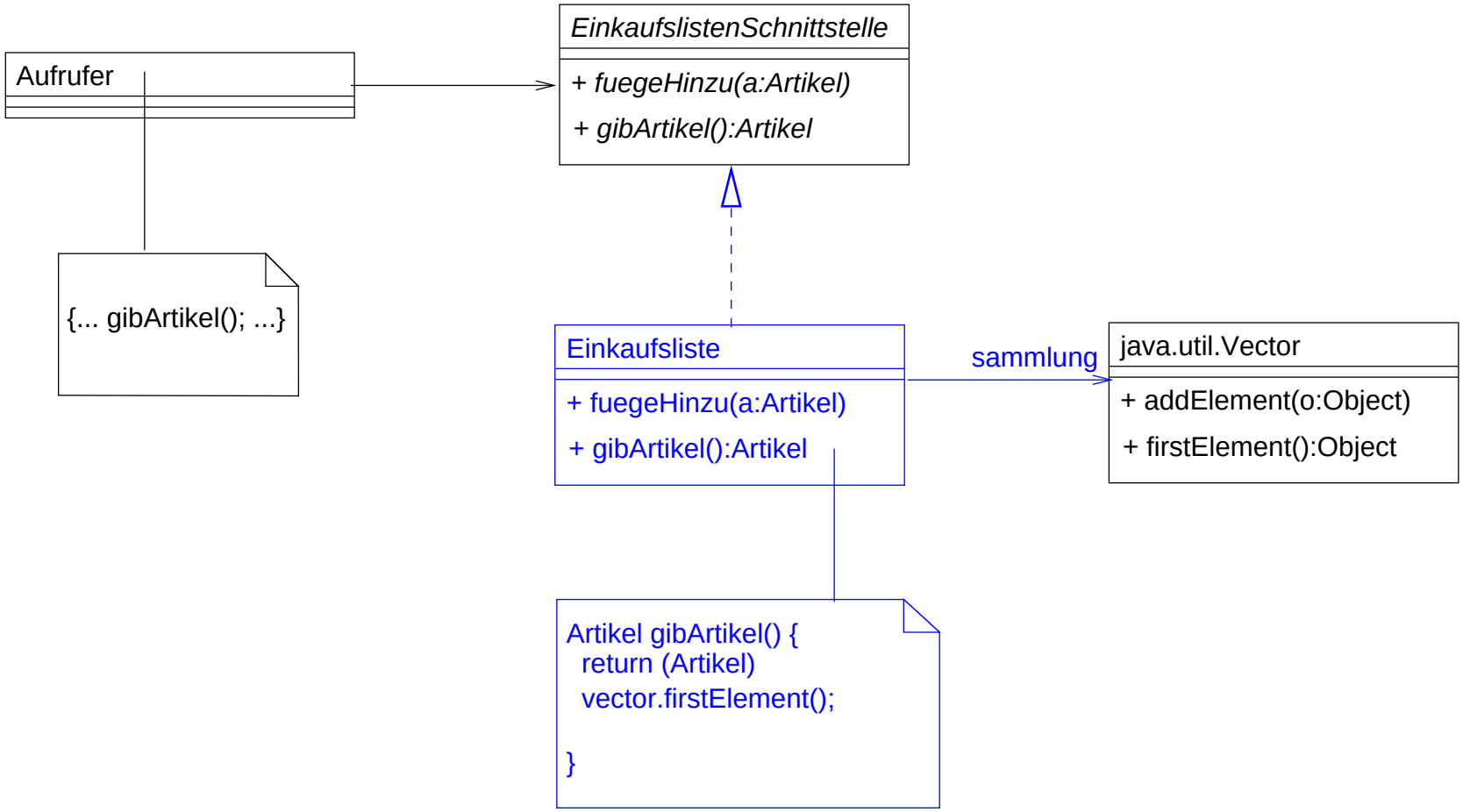
**Beispiel** für das Adapter–Muster: Einkaufsliste



**Beispiel** für das Adapter–Muster: Einkaufsliste und java.util.Vector

- In Java existiert dafür beispielsweise die Klasse `java.util.Vector`.
- Diese Bibliotheksklasse hat eine sehr große Schnittstelle mit vielen Operationen, die wir für Einkaufslisten nicht benötigen.
- Daher bietet sich die Variante des Objektadapters an!
- Der Vector arbeitet allgemein mit `Object`, wir haben aber in der Schnittstelle `Artikel` als Objekte. Neben der Umsetzung der Operationsnamen müssen wir also in den Einkaufslisten–Operationen Typ–Angleichungen vornehmen, um der Schnittstelle gerecht zu werden.

# **Beispiel** für das Adapter–Muster: Einkaufsliste und `java.util.Vector`



**Schema zur Beschreibung von Mustern:**

- **Name**
- **Kontext**
- **Problem**
- **Lösung**
- **Beispiel**
- **Implementierungsvorschlag**
- **Varianten**
- **Gegenanzeigen**

**Implementierungsvorschlag** für das Adapter–Muster: Einkaufsliste

```
public class Einkaufsliste {

  java.util.Vector sammlung = new java.util.Vector();

  void fuegeHinzu(Artikel a) {
    sammlung.addElement(a);
  }

  Artikel gibArtikel() {
   return (Artikel) sammlung.firstElement();
  }
}
```

**Schema zur Beschreibung von Mustern:**

- **Name**
- **Kontext**
- **Problem**
- **Lösung**
- **Beispiel**
- **Implementierungsvorschlag**
- **Varianten**
- **Gegenanzeigen**

**Varianten** für das Adapter–Muster:

- Realisierung
  - Objektadapter
  - Klassenadapter
- Aufwand der Anpassung
  - einfache Umbenennung
  - Typangleichungen
  - komplizierte Vorberechungen, Umcodierung
- Funktionsweise
  - Zwei–Weg–Adapter (Übungsaufgabe 6)

**Schema zur Beschreibung von Mustern:**

- **Name**
- **Kontext**
- **Problem**
- **Lösung**
- **Beispiel**
- **Implementierungsvorschlag**
- **Varianten**
- **Gegenanzeigen**

**Gegenanzeigen** für das Adapter–Muster:

- Allgemein
  - Falls die Bibliotheksklasse nicht über eine wohlgeformte Schnittstelle verfügt, kann eine Neuimplementierung der Wiederverwendung vorzuziehen sein
- Objektadapter
  - Indirektion über Delegation ist wenig effizient
- Klassenadapter
  - Der Klassenadapter ist nicht gut einsetzbar, falls die Bibliotheksklasse noch über Spezialisierungsklassen verfügt
  - die Vereinigung der Schnittstellen führt dazu, dass der Nutzen der Klasse unklar wird (unfokusiert)

## Wo stehen wir bisher?

- Das entwickelte Schema ermöglicht uns
  - Muster zu strukturieren
  - Muster einheitlich zu beschreiben
- Damit können wir beginnen, Muster zu sammeln und zu einem **Musterkatalog** zusammenzustellen
- Der Katalog selbst weist allerdings noch keine Struktur und Organisation auf: Muster sind „Inseln“
  - passendes Muster für Software–Entwickler schwer zu finden
  - Isolation von Mustern behindert Entwurf von größeren und komplexeren Systemen auf Basis von Mustern

**Organisation des Musterkatalogs:**

- Nicht nur Strukturierung eines einzelnen Musters durch Schema
- auch Strukturierung des Katalogs; Ziele sind
  - Ermöglichung eines schnellen Überblicks
  - gezieltes Suchen orientiert an bestimmtem Problem
  - Einschätzung des im Muster enthaltenen Prinzips
  - Bewertung, wie schwierig die Aneignung des Musters ist
- Dabei hilft uns eine **Klassifizierung** der Muster, um sie zu organisieren

**Referenzmodell für Muster:**

Zur genaueren Klassifizierung von Mustern definieren wir ein so genanntes **Referenzmodell**

- Arten von Mustern (schneller Überblick)
- Einsatzgebiet von Mustern (gezielte Suche)
- Inhärentes informatisches Konzept bei Mustern (welches Prinzip?)
- Komplexität (wie schwierig ist Aneigung?)

Damit können wir die Muster eines Katalogs anhand verschiedener Kriterien klassifizieren und eine Organisationsstruktur aufbauen.

**Arten von Mustern:**

- Software–Muster
  - Architekturmuster
  - Entwurfsmuster
  - Idiome
- Prozessmuster
  - Software–Entwicklung
  - Projektmanagement
  - Gesamtsystem– und Betriebsebene

**Einsatzgebiet von Mustern:**

- generell/universell
- speziell
  - nach Systemtypen
    - interaktive Systeme
    - verteilte Systeme
    - ...
  - nach Prozessumgebung
    - Projektgröße
    - organisatorische Besonderheiten

**Inhärentes informatisches Konzept bei Mustern:**

Welches informatische Konzept steht bei dem Problem bzw. der Lösung des Musters im Vordergrund?

- Management
- Konstruktion/Komposition
- Dekomposition
- Interaktionsregelung
- Wiederverwendung

**Komplexität von Mustern:**

- Schwierigkeit des Verständnisses:
  - einfach
  - komplex
- Umfang der Diskussionspunkte
  - ohne Diskussionspunkte
  - viele Diskussionspunkte
- Variationsmöglichkeiten
  - variantenarm
  - variantenreich

## Beispiel für klassifiziertes Muster: Adapter

Klassifizierung des Adapters mit Hilfe des Referenzmodells – Übungsaufgabe 5

- **Art des Musters**: Softwaremuster – Entwurfsmuster
- **Einsatzgebiet**: generell
- **Informatisches Konzept**: Interaktionsregelung, Wiederverwendung
- **Komplexität**: einfach, wenig Diskussionspunkte, variantenreich

## Wo stehen wir bisher?

- Das Referenzmodell ermöglicht uns
  - Muster zu klassifizieren und gruppieren
  - gezielt zu suchen
  - ihre Eignung für bestimmte Probleme zu bewerten
- Dabei erhalten wir ein so genanntes **Mustersystem**.
- Die Muster des Mustersystems sind gruppiert, aber haben noch keine Verbindungen miteinander: Muster sind „anonyme Nachbarn“
  - Isolation von Mustern behindert Entwurf von größeren und komplexeren Systemen auf Basis von Mustern

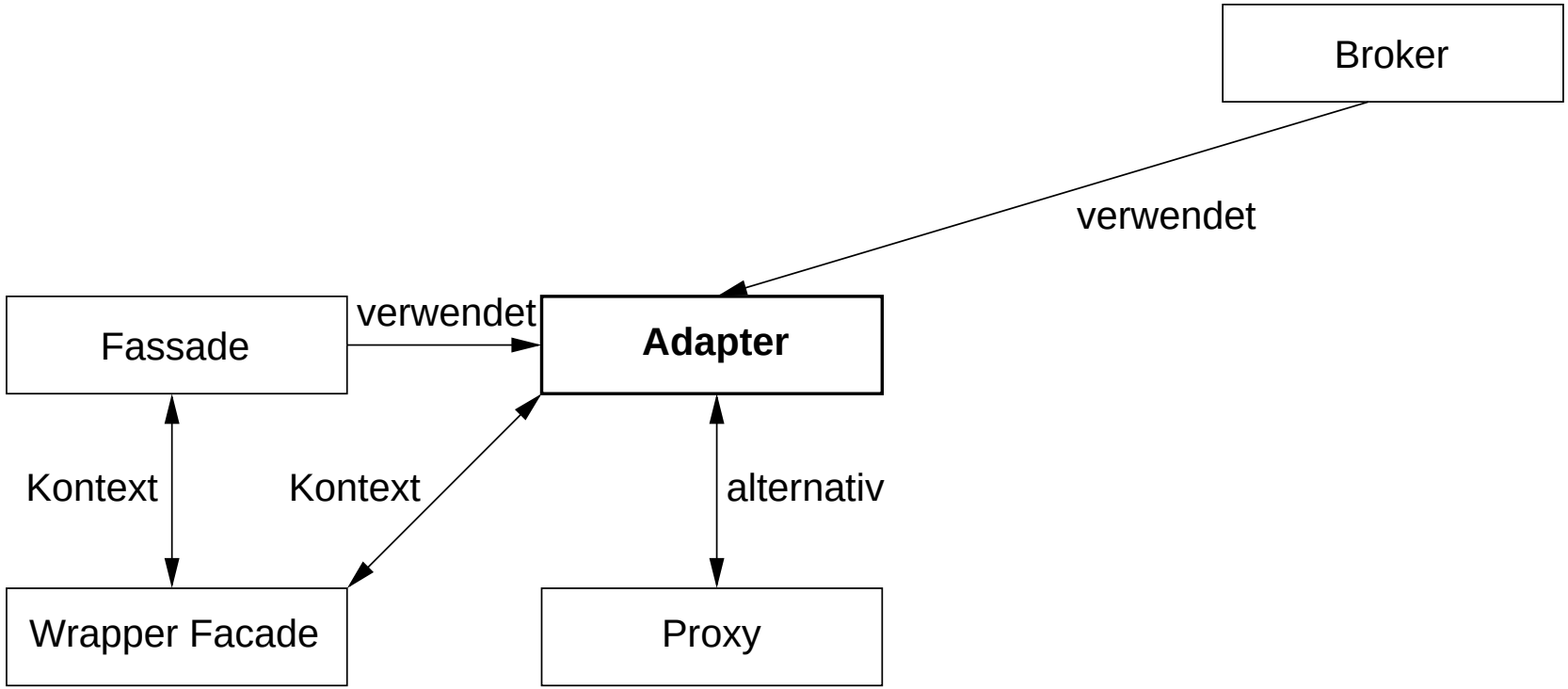
## Kombination von Mustern

Betrachtet man Muster im Zusammenhang untereinander, so stellen sich folgende Fragen:

- Welche Muster werden verwendet, das Muster zu realisieren?
- Welche Muster können realisiert werden mit diesem Muster?
- Welche Muster können ergänzend verwendet werden?
- Welche Alternativen gibt es zu diesem Muster?
- Welche Gemeinsamkeiten/Unterschiede (z.B. im Kontext) bestehen zu anderen Mustern?

Eine Sammlung von Mustern, in der diese Beziehungen ausgedrückt sind, wird als **Mustersprache** bezeichnet.

# Ausschnitt aus einer Mustersprache



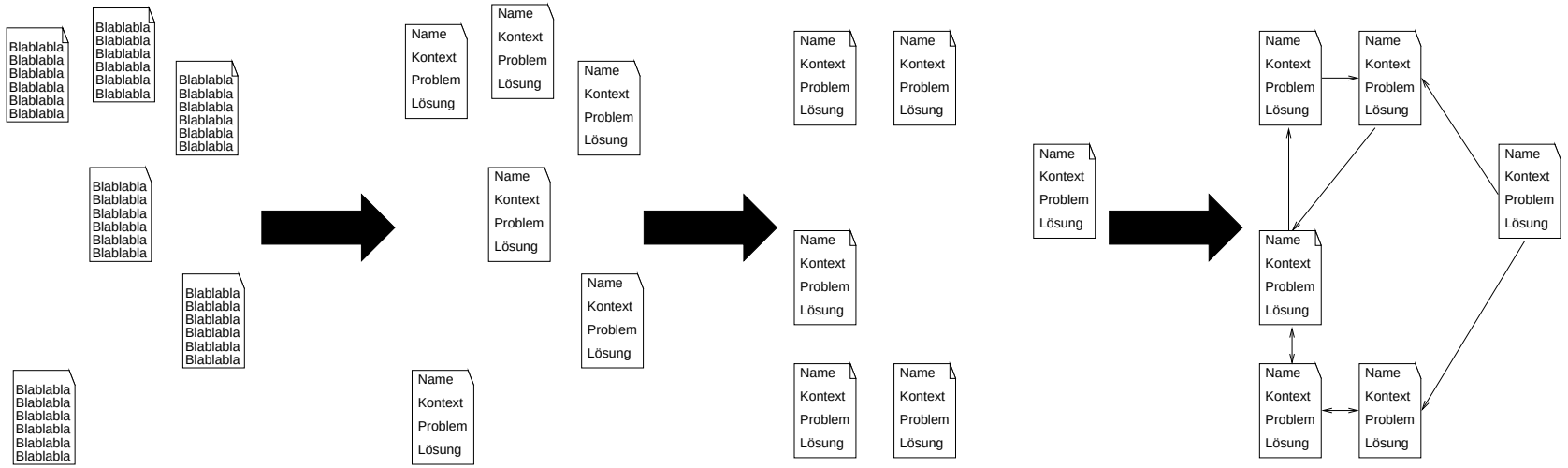
**Zusammenfassung:**

Wir haben nun zur weiteren Verwendung ein erweitertes Schema für Muster. Dies enthält Elemente zur

- **Strukturierung**
  - Name, Kontext, Problem, Lösung, Beispiel, Varianten, Implementierung, Gegenanzeigen
- **Klassifizierung**
  - Art, Einsatzgebiet, Konzept, Komplexität
- **Kombination**
  - Realisierung, Verwendung, Ergänzung, Alternative

## Zusammenfassung (Fortsetzung):

Damit haben wir das nötige Werkzeug, um **Musterkataloge** zu durchforsten, in **Mustersystemen** zu organisieren und **Mustersprachen** zu definieren.




Nächste Woche geht’s los damit …

**Zum Nachschmökern:**

Schemata und Referenzmodelle:

- Brown, W. J., R. Malveau, H. McCormick III, T. Mowbray: *Anti Patterns*, Wiley & Sons. 1998

Mustersysteme:

- Buschmann, F., R. Meunier, H. Rohnert, P. Sommerlad, M. Stal: *A System of Patterns*, Wiley & Sons. 1996

Mustersprache:

- Alexander, C., S. Ishikawa, M. Silverstein: *A Pattern Language – Towns Buildings Construction*, 1977
- Schmidt D., Stal M., Rohnert, H., Buschmann, F.: *Pattern–oriented Software Architecture*, Wiley & Sons, Chichester, 2000