Entwurf

Prof. Sven Apel

Universität des Saarlandes

Basierend auf Material von Andreas Zeller und Norbert Siegmund

Teil II

Vorgehen beim Entwurf

Entwurf nach Zuständigkeit

Ein verbreitetes Grundprinzip ist der *Entwurf nach Zuständigkeit*: Jedes Objekt ist für bestimmte Aufgaben zuständig und

- *besitzt* entweder die *Fähigkeiten*, die Aufgabe zu lösen,
- oder es *kooperiert* dazu mit anderen Objekten.

Ziel ist damit *das Auffinden von Objekten* anhand ihrer *Aufgaben in der Kooperation*.

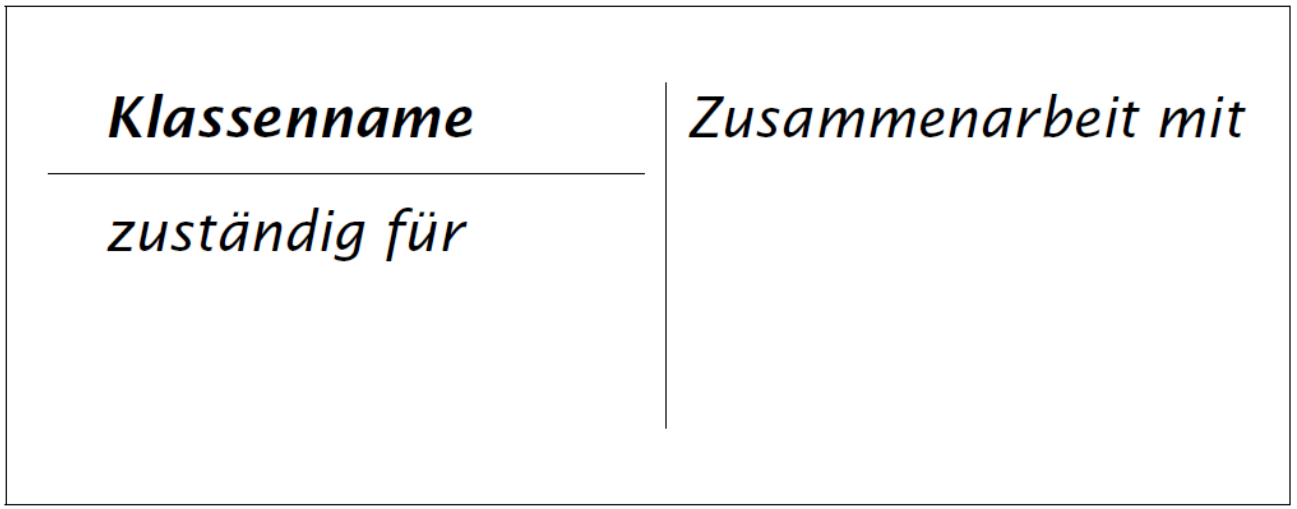
Entwurf nach Zuständigkeit

Wir gehen zunächst von einer *informalen Beschreibung* aus und betrachten *zentrale Begriffe* der Aufgabenstellung:

- *Hauptworte* in der Beschreibung werden zu *Klassen* und *konkreten Objekten*
- *Verben* in der Beschreibung werden zu *Diensten* –
  - entweder zu Diensten, die ein Objekt anbietet,
  - oder zu Aufrufen von Diensten kooperierender Objekte.
- Diese Dienste kennzeichnen die *Zuständigkeit* und *Kooperationen* der einzelnen Klassen.

Entwurf nach Zuständigkeit

Die so gefundenen Klassen werden dann anhand ihrer *Zuständigkeiten* auf sogenannte *CRC-Karten* (Klasse – Zuständigkeit – Kooperation) notiert:



Die CRC-Karte gibt die *Rolle* wieder, die Objekte im Gesamtsystem übernehmen sollen.

Beispiel

Studentin Dagmar bestellt mit einem Brief bei der Firma Rechnerwelt einen Rechner. Dieser wird ihr nach mehreren Tagen als Paket durch die Firma Gelbe Post zugestellt.

Eine erste Näherung

Student	*Zusammenarbeit mit*
zuständig für	
bestellen	Rechnerhändler
Paket annehmen	Zusteller

Rechnerhändler	*Zusammenarbeit mit*
zuständig für	
Bestellung annehmen	Student
Paket absenden	Zusteller

Zusteller	*Zusammenarbeit mit*
zuständig für	
Paket annehmen	Rechnerhändler
Paket abgeben	Student

"One purpose of CRC cards is to *fail early*, to *fail often*, and to *fail inexpensively*. It is a lot cheaper to tear up a bunch of cards than it would be to reorganize a large amount of source code."

C. Horstmann

Verfeinerung des Entwurfs

Diese erste Näherung ist jedoch noch nicht vollständig:

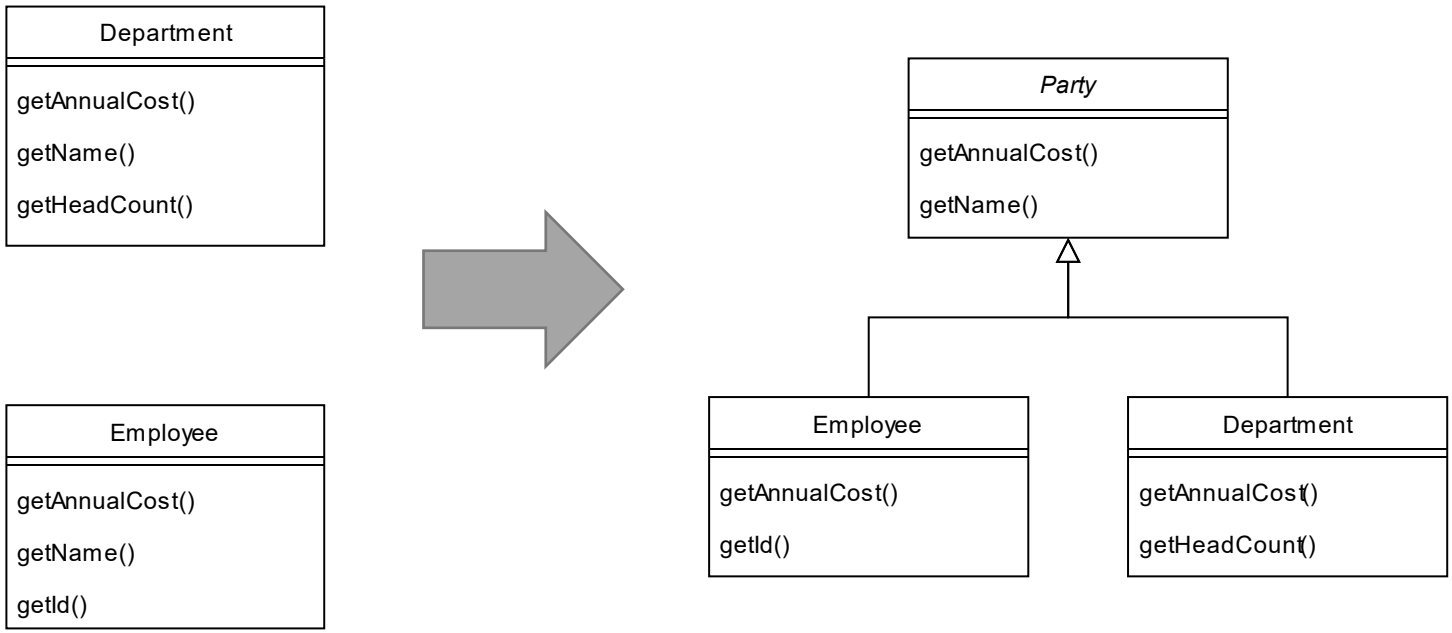
- Dagmar tritt in ihrer Rolle als Kunde auf; es kommt nicht darauf an, dass sie auch studiert (es sei denn, sie bekäme Studentenrabatt). Besser wäre also der Klassenname „Kunde“ statt „Student“.
- Brief und Paket fehlen – es handelt sich um reine Datenobjekte, die weder Zuständigkeit noch Kooperationspartner haben.
- Wir haben offengelassen, wie der Bestellbrief zum Rechnerhändler gelangt; ggf. ist hierfür ebenfalls ein Zusteller zuständig.
- *Datenfluss*, *Zustandsübergänge* und *Klassenhierarchien* sind nicht berücksichtigt.

Überarbeitung des Entwurfs

Ein erster Grobentwurf kann meistens deutlich verbessert werden:

- Herausfaktorisieren gleicher Merkmale.
  - Können *gemeinsame Merkmale* (Attribute, Methoden) verschiedener Klassen in *Zusammenhang* gebracht werden?
- Diese gemeinsamen Merkmale können
  - (1) in eine *dritte* Klasse verlagert werden, die von den bestehenden Klassen aggregiert wird. Die bestehenden Klassen müssen die ausgelagerten Dienste aber weiterhin anbieten
  - (2) in eine *gemeinsame Oberklasse* verlagert werden. Dies ist bei gemeinsamen *ist-ein-Beziehungen* sinnvoll.

Beispiel: Herausfaktorisieren



Überarbeitung des Entwurfs

Verallgemeinerung von Verhaltensweisen.

Können Methoden mit einheitlicher Schnittstelle bereits auf einer abstrakten Ebene angegeben werden?

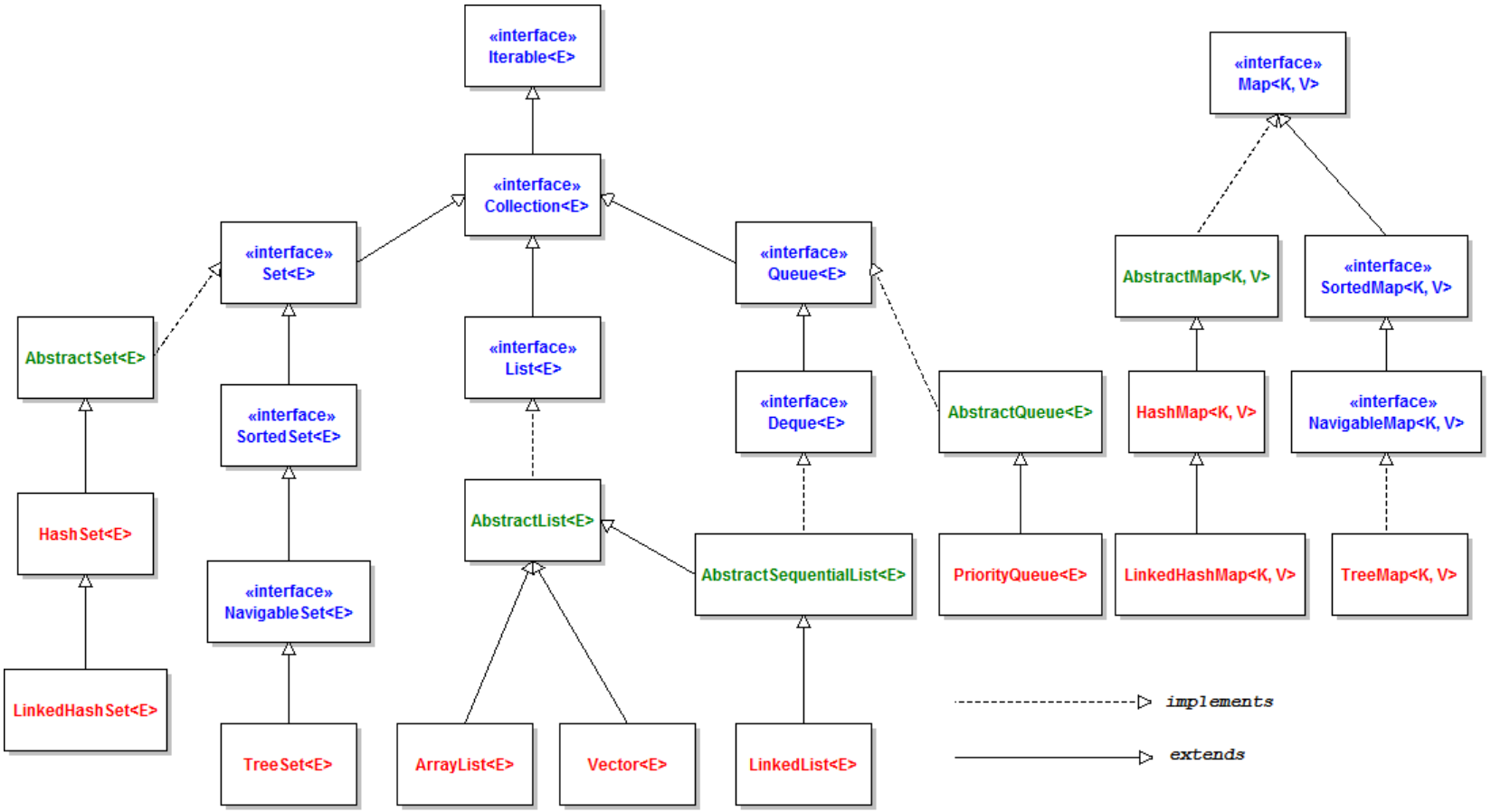
Abstrakte Klassen können etwa allgemeine Methoden bereitstellen, deren Details (abgeleitete Kernmethoden) in den konkreten Unterklassen realisiert werden.

Ersetzen einer umfangreichen Klasse durch ein Teilsystem.

Können Klassen mit vielen Merkmalen weiter unterteilt werden?

Evtl. Einführung eines Teilsystems aus mehreren Objekten und zugehörigen Klassen

Beispiel: Verallgemeinerung



Überarbeitung des Entwurfs

Minimierung von Objektbeziehungen.

Kann man durch Umgruppierung der Klassen oder durch neue Schnittstellen die Zahl der „Benutzt“-Beziehungen verringern?

Auch hier unterhält nur noch ein neu einzuführendes Teilsystem Außenbeziehungen.

Wiederverwendung, Bibliotheken.

Kann man bestehende Klassen wiederverwenden?

Ggf. können geeignete *Anpassungsklassen* (Adapter) eingeführt werden

Beispiel: Adapter

Problem: Wir würden gern bestehende Funktionalität verwenden aber die Schnittstelle passt nicht!

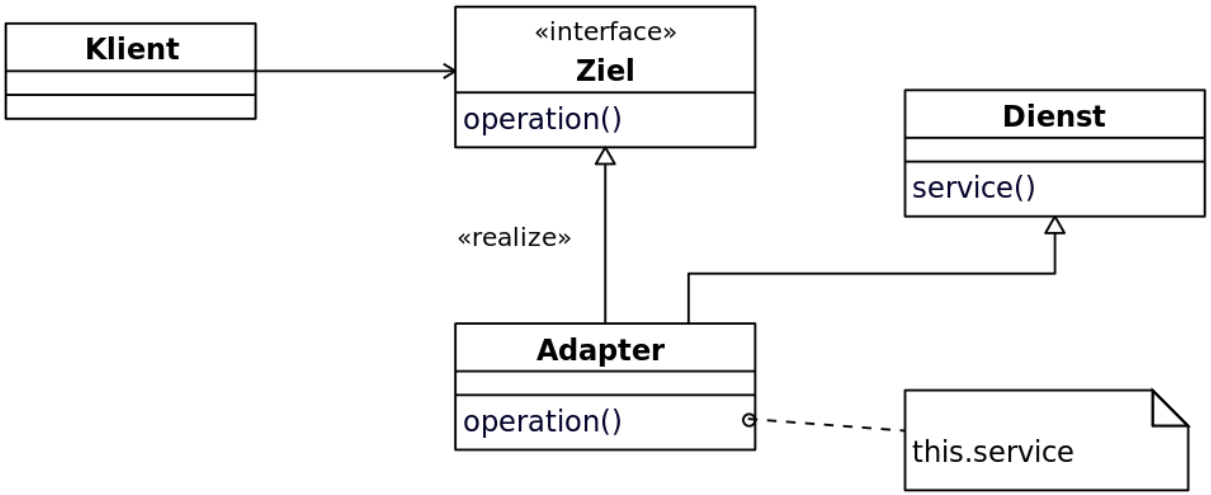
Ziel

```
interface Stack<T> {
  public void push (T t);
  public T pop ();
  public T top ();
}
```

Dienst

```
class DList<T> {
  public void insert (DNode pos, T t);
  public void remove (DNode pos, T t);
  public void insertHead (T t);
  public void insertTail (T t);
  public T removeHead ();
  public T removeTail ();
  public T getHead ();
  public T getTail ();
}
```

Lösung: Adapter



Beispiel: Adapter

Ziel

```
interface Stack<T> {
  public void push (T t);
  public T pop ();
  public T top ();
}
```

Dienst

```
class DList<T> {
  public void insert (DNode pos, T t);
  public void remove (DNode pos, T t);
  public void insertHead (T t);
  public void insertTail (T t);
  public T removeHead ();
  public T removeTail ();
  public T getHead ();
  public T getTail ();
}
```

Adapter

```
class DListImpStack<T> extends DList<T> implements Stack<T> {
  public void push (T t) {
    insertTail (t);
  }
  public T pop () {
    return removeTail ();
  }
  public T top () {
    return getTail ();
  }
}
```

Überarbeitung des Entwurfs

Generizität

Können generische Klassen und Methoden eingesetzt werden?

Oder auch: können Klassen und Methoden des Entwurfs generisch gestaltet werden?

Entwurfsmuster

Kann man Standard-Entwurfsbausteine verwenden?

Beispiel: Generizität

```
class ThreadPool {
   Thread[] pool …
    …
}

class ConnectionPool {
    Connection[] pool …
    …
}

… new ThreadPool();
… new ConnectionPool();
```

```
class Pool<T> {
    T[] pool …
    …
}

… new Pool<Thread>();
… new Pool<Connection>();
```

Modellierung: Digital vs. Analog