22 Komponenten und Verträge

22.1 Motivation und Definition

Objektorientierter Entwurf: Gliederungseinheit ist die Klasse

- Klasse besteht aus
  - Schnittstelle (Klassendefinition)
  - Implementierung
- Schnittstelle und Implementierung bilden eine Einheit

Wünsche:

- Gliederungseinheiten oberhalb von Klassen
- Trennung von Schnittstelle und Implementierung
- ⇨ Komponenten

Komponente (component) [im engeren Sinn] – Stark gekapselte Menge zusammengehöriger Objekte / Klassen, die eine gemeinsame Aufgabe lösen

- Elemente der Komponente lösen zusammen ein Teilproblem
- Komponente wird typisch als Ganzes verwendet/wiederverwendet
- Bietet die Möglichkeit, nach außen nur eine (gemeinsame) Schnittstelle anzubieten

[Selbständige] **Schnittstelle** (interface) – Komponente, die nur aus einer Leistungsbeschreibung ohne jegliche Implementierung besteht

- keine Klasse
- Leistungen können je nach Umfang der Schnittstelle durch Klassen oder Komponenten erbracht werden
- Leistungsbeschreibung und Leistungserbringung sind völlig entkoppelt
- Jede Implementierung, welche alle in der Schnittstelle beschriebenen Leistungen erbringt, ist zulässig

Beispiel:

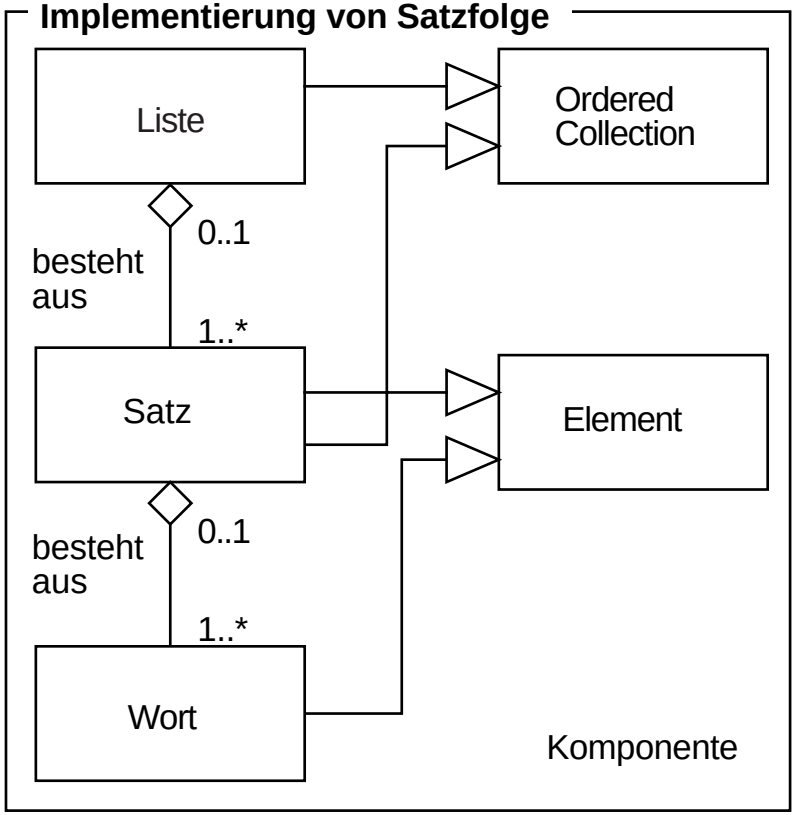
Satzfolge: Schnittstelle

listeErzeugen
satzBilden
wortErzeugen
wortAufPositionEinfügen

satzLöschen
wortAusPositionLöschen

ersterSatz
nächster Satz

erstesWort
nächstesWort
wortSuchen
wortErneutSuchen



22.2 Das Vertragsprinzip für die Definition von Schnittstellen

- Schnittstelle ist **Vertrag** (contract) zwischen Modul und Modulverwender
- Beschreibung des Vertrags mit **Zusicherungen** (assertions)
  - Voraussetzungen (precondition, requirement, Schlüsselworte: pre, require)
  - Ergebniszusicherungen (postconditions, Schlüsselworte: post, ensure)
  - Invarianten (invariants)
  - Verpflichtungen (obligations)
- Vertragserfüllung bedeutet:
  - Verwender muss
    - Voraussetzungen erfüllen
    - Übernommene Verpflichtungen einhalten
  - Jede Implementierung der Schnittstelle muss
    - Zusicherungen erfüllen
    - Invarianten garantieren
    - aber unter der Annahme der Vertragstreue des Verwenders!

Eigenschaften einer vertragsorientierten Schnittstellendefinition

- Präziser und eindeutiger als einfacher Kommentartext
- Voraussetzungen und Resultate klar formulierbar
- Benötigt in der Regel Zustandsvariablen

  Vorsicht: Keine internen Zustandsvariablen verwenden, sondern nur solche aus dem Problembereich

- ☆ In gleicher Weise für die Definition von Modulschnittstellen (Funktionen, Klassen, Komponenten...) und von selbständigen Schnittstellen (ohne Implementierung) verwendbar

22.3 Die Vertragselemente

Voraussetzungen

- ☆ Werden an eine Operation (Methode) gestellt
- ☆ Müssen zum Zeitpunkt des Aufrufs durch den Aufrufer(!) erfüllt sein
- ☆ Werden von der Implementierung der Operation (Methode) nicht geprüft (!)

Ergebniszusicherungen

- ☆ Beschreiben die Effekte einer Operation (Methode) für den Aufrufer
- ☆ Müssen von der Implementierung der Operation (Methode) erfüllt werden
- ☆ Aber unter der Annahme, dass die Voraussetzungen erfüllt sind

Beispiel: Ein einfaches Konto

```
interface EinfachesKonto
{
public EinfachesKonto ();
//    PRE –
//    POST int saldo = 0
public EinfachesKonto Einzahlen (int betrag);
//    PRE betrag ≥ 0
//    POST saldo = saldo@PRE + betrag
public EinfachesKonto Abheben (int betrag);
//    PRE betrag ≥ 0
//    POST saldo = saldo@PRE - betrag
public int Kontostand ();
//    PRE –
//    POST result = saldo and saldo = saldo@PRE
}
```

Hinweis: Die Angabe der Konstruktormethode ist erforderlich, damit der Anfangszustand von saldo spezifiziert werden kann.

Invarianten

- ☆ Beschreiben Eigenschaften der Schnittstelle, die unter allen Operationen invariant sind
- ☆ Spezifizieren Zusammenhänge zwischen Operationen
- ☆ Entlasten die Ergebniszusicherungen der Operationen
- ☆ Beispiel:

  interface EinfachesKonto
  {
  INVARIANT **with** e = Summe aller mit Einzahlen eingezahlten Beträge,
  a = Summe aller mit Abheben abgehobenen Beträge **holds** saldo = e - a
  ...
  }

  - ❍ Garantiert, dass der Saldo nur durch Einzahlen und Abheben verändert wird
  - ❍ Ermöglicht, die Bedingung saldo = saldo@PRE in der Ergebniszusicherung von Kontostand wegzulassen

Verpflichtungen

- ☆ Beschreiben Pflichten, die der Aufrufer beim Aufruf einer Operation übernimmt
- ☆ Beispiel: In einem einfachen Sperrprotokoll übernimmt jeder, der eine Sperre setzt, die Verpflichtung, diese Sperre auch wieder freizugeben.

```
interface EinfacheSperre
    {
    public EinfacheSperre ();
    // PRE –
    // POST boolean gesperrt = false
    public boolean Sperren ();
    // PRE –
    // POST if gesperrt then result = false else gesperrt and result = true endif
    // OBLIGATION sometimes Freigeben()
    public void Freigeben ();
    // PRE –
    // POST gesperrt = false
    }
```

22.4 Voraussetzen vs. Prüfen

Voraussetzen oder Prüfen (mit Fehlermeldung)? ➔ Entwurfsentscheidung!

- ☆ Nicht dem Geschmack der Programmierer überlassen
- ☆ Voraussetzen immer dann, wenn dem Aufrufer die Erfüllung der Voraussetzungen zugemutet werden kann
- ☆ Prüfen immer dann, wenn mit Falscheingaben gerechnet werden muss (zum Beispiel bei Benutzereingaben)
- ☆ Prüfen nur, wenn eine sinnvolle Behandlung von Fehlern möglich ist
- ☆ Beispiel:

  double Kalibrieren (double a, double b);
  // Die Korrekturfaktoren a und b werden vom Benutzer eingegeben
  // PRE a ≠ 0
  // POST result = (this/a)*Fkorr(b)

- ⇨ Gefährlich: Wenn der Wert von a eingegeben wird, kann a ≠ 0 nicht vorausgesetzt werden

22.5 Dokumentation der Zusammenarbeit im Entwurf

Durch Zusammenarbeit werden aus Modulen bzw. Komponenten Systeme

- ➜ Die Festlegung und Dokumentation der Zusammenarbeit ist eine zentrale Entwurfsaufgabe
- ☆ Der Dokumentationsbedarf hängt vom Grad der Unabhängigkeit der Komponenten ab

- ☆ Bei gemeinsam erstellten und vertriebenen Komponenten
  - ❍ Komponenten und Zusammenarbeit werden miteinander verzahnt entworfen
  - ❍ Der Verwendungskontext jeder Komponente ist bekannt
  - ❍ Entwerfende stimmen Schnittstellen und Zusammenarbeitsbedürfnisse aufeinander ab
  - ❍ Dokumentation durch Nennung der verwendeten Komponenten in der Schnittstelle jeder Komponente und Übersicht mit einem Zusammenarbeits-diagramm genügt

- ☆ Bei separat erstellten und vertriebenen Komponenten
  - Jede Komponente steht für sich und kennt bei Erstellung ihren Verwendungskontext nicht
  - Die benötigte Zusammenarbeit mit Dritten muss ebenso ausführlich und präzise dokumentiert sein wie das in der Schnittstelle spezifizierte Leistungsangebot der Komponente: Neben die Leistungsschnittstelle muss eine Bedarfsschnittstelle treten
  - Für jede benötigte Operation sind zu spezifizieren
    - Die Voraussetzungen, welche die benötigte externe Operation höchstens machen darf*
    - Die Ergebniszusicherungen, welche die benötigte Operation mindestens machen muss*
  - Benötigt eine Komponente eine Zusammenarbeit mit wechselseitigem Aufruf von Methoden (zum Beispiel Zusammenarbeit nach dem Abonnementsprinzip) so muss ein Zusammenarbeitsprotokoll spezifiziert werden

*Genau invers zur Leistungsschnittstelle, wo die mindestens zu erfüllenden Voraussetzungen und die höchstens gelieferten Ergebnisse spezifiziert werden

22.6 Spezialisierung von Schnittstellen; Schnittstellenvererbung

Das Prinzip der Vererbung kann in gleicher Weise wie auf Klassen auch auf Schnittstellen angewendet werden:

- ☆ **Steinbruch**: Die Vererbung dient nur dazu, Schreibaufwand zu sparen, indem Teile einer bestehenden Schnittstelle übernommen werden. Im übrigen wird beliebig ergänzt und abgeändert.

- ☆ **Spezialisierung**: Sei S' eine Subschnittstelle von S. Die von S' offerierten Leistungen sind ein Spezialfall der von S offerierten Leistungen.

- ☆ **Substituierbarkeit**: Sei S' eine Subschnittstelle von S. Jede korrekte Implementierung von S' ist gleichzeitig auch eine korrekte Implementierung von S. Dementsprechend ist jede Implementierung von S durch eine beliebige (korrekte) Implementierung von S' ersetzbar.

Beispiel: Steinbruch

Von der Schnittstelle EinfachesKonto wird eine Schnittstelle Spielkasino abgeleitet, weil die Definition der Ein- und Auszahloperationen teilweise wiederverwendet werden kann

- Die Schnittstellen haben keinen systematischen Zusammenhang
- Finger weg von Steinbruchvererbung bei Schnittstellen

Beispiel: Spezialisierung

Von der Schnittstelle EinfachesKonto wird eine Schnittstelle Sparkonto abgeleitet.

Sparkonto hat «Saldo $\geq$ 0» als zusätzliche Invariante, ist also ein echter Spezialfall von EinfachesKonto

Aber: Korrekte Implementierungen von Sparkonto sind keine korrekten Implementierungen von EinfachesKonto (Warum?)

- Die Implementierungen sind nicht substituierbar

Beispiel: echte Subschnittstelle

Von der Schnittstelle EinfachesKonto wird eine Schnittstelle KontoMitVerbuchung abgeleitet, welche bei jeder Mutation des Saldos zusätzlich die Verbuchung dieser Mutation zusichert.

KontoMitVerbuchung ist eine echte Subschnittstelle von EinfachesKonto: Jede korrekte Implementierung von KontoMitVerbuchung ist gleichzeitig auch eine korrekte Implementierung von EinfachesKonto.

- ☆ Nur echte Subschnittstellen garantieren universelle Verwendbarkeit der Implementierungen
- ☆ Aber: Restriktive Bedingungen: Die Subschnittstelle darf gegenüber der Schnittstelle, von der sie abgeleitet ist
  - ✧ keine stärkeren Voraussetzungen machen
  - ✧ keine schwächeren oder irgendwie eingeschränkten Ergebnisse liefern

Benutzung statt Vererbung

Andere Möglichkeit der Schnittstellenhierarchie: Benutzungshierarchie statt Vererbungshierarchie

- Schnittstellen (und deren Implementierungen) benutzen andere Schnittstellen

Beispiel:

- Die Schnittstelle KontoMitVerbuchung
  - legt fest, dass ihre Implementierungen die Schnittstelle EinfachesKonto verwenden
  - verwendet in der Schnittstellendefinition Operationen von EinfachesKonto
- Ebenso benutzt die Schnittstelle Sparkonto die Schnittstelle KontoMitVerbuchung