Softwarequalitätssicherung

Sommersemester 2003

Dr. Thomas Santen
Softwaretechnologie
TU Dresden

KLASSEN-KONTRAKTE

1. Warum Kontrakte?
2. Methoden-Spezifikation mit Vor- und Nachbedingungen
3. Klassen-Invarianten
4. Kontrakte
5. Verhaltenskonformität

WARUM KONTRAKTE?

Präzise Beschreibung der grundlegenden funktionalen Eingenschaften der Instanzen einer Klasse an ihrer *Schnittstelle*:

- Was fordert die Klassen von ihren Klienten?
- Was garantiert die Klasse ihren Klienten?
- Welche Kombinationen von Attribut-Werten (Datenzustand) sind zulässig?

Design by Contract

„Vertrag“: Erfüllt der Klient die Forderungen des Servers, dann bietet der Server die zugesicherte Funktionalität.

- Klient kann sich auf die Zusicherungen des Servers verlassen.
  Interna der Server-Klasse brauchen den Klienten nicht zu interessieren.
- Erfüllt Klient die Forderungen des Servers nicht, dann darf der Server sich beliebig verhalten (einschl. Absturz).

DIE KLASSE TRIANGLE

```
class Triangle extends Polygon {
   public Triangle(LineSegment a, LineSegment b, LineSegment c)
   public void setA(LineSegment a)
   public void setB(LineSegment b)
   public void setC(LineSegment c)

   public LineSegment getA()
   public LineSegment getB()
   public LineSegment getC()

   ⋮
```

Methoden-Spezifikation – Vorbedingung

Methoden bilden die operationale Schnittstelle zwischen Klient und Server.

Ein Kontrakt auf Methodenebene muss deshalb die Bedingung beschreiben, unter der ein Klient eine Methode aufrufen darf (Vorbedingung) und den Effekt der Methode, den der Server dann garantiert (Nachbedingung).

Vorbedingung: Prädikat über den Parametern der Methode und den Attributen der Klasse.

Forderung des Servers an den Klienten –

muss beim Aufruf der Methode gelten.

---

eine Vorbedingung der Methode `setA(LineSegment a)`:

`a.length` > 0

Methoden-Spezifikation – Nachbedingung

Der Effekt einer Methode beschreibt den Ergebniszustand und die Werte der Ausgabeparameter in terminis der Eingabeparameter und des Zustands vor Ausführung der Methode.

Nachbedingung: Relation zwischen den Eingabeparametern, Attributen der Klasse *vor* Ausführung der Methode, den Attributen der Klasse *nach* Ausführung der Methode und Ausgabeparametern.

Nachbedingung der Methode `setA(LineSegment a)`:

die beabsichtigte Änderung tritt ein ...

`getA() = a`

und es ändert sich sonst nichts ...

`getB() = getB@pre() and getC() = getC@pre()`

Die letzte Bedingung wird oft implizit angenommen (*Frame Axiom*).

Klassen-Invariante

Nicht alle Kombinationen von Attribut-Werten beschreiben eine zulässige Instanz einer Klasse.

Klassen-Invariante: Eigenschaft, die den beabsichtigten Zusammenhang der Attribute einer Klasse beschreibt.

Die Klassen-Invariante ist implizt Teil der Vorbedingungen und Nachbedingungen aller Methoden!

Invariante der Klasse `Triangle`:

„Die Linienabschnitte a, b und c bilden ein Dreieck“

`is_triangle(getA(),getB(),getC())`

Übung: Formulieren Sie das Prädikat `is_triangle(a,b,c)`.

Kontrakt – Verbindung zwischen Klient und Server

„Vorleistung“ des Klienten:

- Vorbedingung von Konstruktoren einhalten
- Vorbedingung von Methoden einhalten

„Gegenleistung“ des Servers:

- Konstruktoren etablieren Klasseninvariante
- Methoden erhalten Klasseninvariante
- Methoden etablieren Nachbedingung

Kann `setA` unter der Vorbedingung **`a.length`** > 0 und der Klasseninvariante garantieren, dass nachher die Klasseninvariante gilt?

- Nein! Parameter a muss zu den Seiten b und c passen!
- Vorbedingung von `setA(LineSegment a)`:
  `a.length` > 0 **and** **`is_triangle(a,getB(),getC())`**

VORBEDINGUNGSANALYSE

Die zusätztliche Vorbedingung von `setA` erkennt man durch eine *Vorbedingungsanalyse*:

> Welche Bedingung an die Eingaben muss gelten, damit es Attributwerte nach Ausführung der Methode und Belegungen für die Ausgabeparameter gibt, so dass die Nachbedingung einschließlich der Klasseninvariante erfüllt ist?

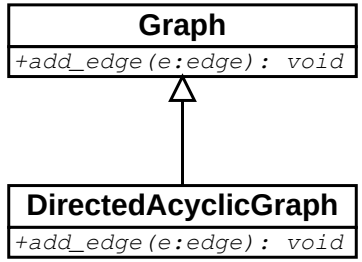
Die Antwort liefert die *schwächste* Vorbedingung der Methode.

VERHALTENSKONFORMITÄT – GEGENBEISPIEL

`Graph`: beliebiger Graphen

`DAG`: gerichteter azyklischer Graph

`add_edge` fügt Kante in den Graph ein



Welches Problem kann auftreten, wenn ein Klient über dynamisches Binden als polymorphen Server statt eines `Graph` einen `DirectedAcyclicGraph` bekommt?

VERHALTENSKONFORMITÄT – GEERBTE KONTRAKTE

Falls eine Vererbungshierarchie Teil einer Schnittstelle sein soll, d.h. Klienten polymorph auf Server zugreifen dürfen, *dann* muss eine Unterklasse alle Kontrakte aller Oberklassen einhalten.

- die Klasseninvariante muss stärker sein als die der Oberklassen
- Vorbedingungen re-definierter Methoden müssen schwächer sein als die der Super-Methoden
- Nachbedingungen überschriebener Methoden müssen stärker sein als die der Super-Methoden

Diese Bedingungen garantieren (fast), dass ein Klient keine „Überraschungen“ erlebt, wenn er einen polymorphen Server benutzt ohne zu wissen, welchen dynamischen Typ das Server-Objekt genau hat.

VORTEILE DES DESIGN BY CONTRACT

- Kontrakte machen vorhandene Einschränkungen explizit
- klare Aufteilung der geforderten Funktionalität an der Schnittstelle zwischen Klient und Server
- Vermeidung überflüssiger Abfragen durch übermäßig defensive Programmierung
- Abstraktion von der Server-Implementierung (Austauschbarkeit)
- (teilweise) Überprüfung zur Laufzeit durch Zusicherungen (*assertions*)