Tagesprogramm

Zusicherungen

Zusicherungen und Ersetzbarkeit

Ersetzbarkeit und Verhalten

U ist Untertyp von T wenn

Instanz von U überall verwendbar wo Instanz von T erwartet

Struktur der Typen für Ersetzbarkeit nicht ausreichend

Beispiel:

```
void draw()  // zeichne Bild
void draw()  // ziehe Revolver
```

Ersetzbarkeit muss Verhalten berücksichtigen → **Design by Contract**

Client-Server-Beziehungen

Server bietet Dienste an, Client nutzt Dienste
wobei Dienst = Ausführung einer Methode

Objekt ist gleichzeitig Client und Server

Vertrag (Contract) zwischen Client und Server:

- **Client** erfüllt **Vorbedingungen** eines Dienstes
- **Server** erfüllt **Nachbedingungen** eines Dienstes
- **Server** erfüllt **Invarianten** des Objekts
- **Client + Server** erfüllen **History-Constraints** des Objekts

Vorbedingung (Precondition)

Verantwortlich: Client

Wann: vor Methodenaufruf

Was: hauptsächlich Eigenschaften von Argumenten

Beispiel: Argument ist Array aufsteigend sortierter Zahlen

manchmal auch (sichtbarer) Zustand des Servers

Beispiel: `abheben` nicht aufrufen wenn Konto überzogen

Nachbedingung (Postcondition)

Verantwortlich: Server

Wann: vor Rückkehr aus Methodenaufruf

Was: Eigenschaften von Methodenergebnissen sowie
Änderungen und Eigenschaften des Objektzustands

Beispiel: „Methode fügt Element (falls noch nicht vorhanden) in Menge ein. Ergebnis ist 'true' falls Element bereits vorher in Menge war."

Nachbedingung klingt oft wie Methodenbeschreibung

Invariante

Verantwortlich: Server

Wann: vor und nach Ausführung von Methoden

Was: unveränderliche Eigenschaften von Objekten und Variablen

Beispiel: Guthaben am Sparbuch ist immer positive Zahl

Gültigkeit der Invariante kann von Bedingungen abhängen

Beispiel: „'zuverlaessig == false' wenn Konto überzogen“

impliziert Nachbedingung

Ausnahme: wenn Variable von außen geschrieben werden kann

dann Client **und** Server für Invariante darauf verantwortlich

Server-kontrollierter History-Constraint

Verantwortlich: Server

Wann: nach Ausführung von Methoden

im Bezug auf Zustand vor Ausführung der Methoden

Was: Einschränkungen auf Veränderungen von Variablen

Beispiel: „Zählerwert erhöht sich mit jedem Methodenaufruf“

Prüfung vor Methodenausführung von Natur aus unmöglich

impliziert Nachbedingung

Ausnahme: wenn Variable von außen geschrieben werden kann

dann Client **und** Server gemeinsam verantwortlich

Client-kontrollierter History-Constraint

Verantwortlich: Client

Wann: vor Methodenaufrufen

Was: Einschränkungen auf der Reihenfolge von Aufrufen

Beispiele: zuerst `initialize`, dann andere Methoden aufrufbar;
nach jedem Aufruf von A ein Aufruf von B nötig

Server kennt Aufrufreihenfolge oft nicht, Clients schon

ein Client oder **alle** Clients bestimmen Reihenfolge

Zusammenhang mit Synchronisation (Koordination)

Beispiel zu klassischen Zusicherungen

```
public class Konto {
    public int guthaben;
    public int ueberziehungsrahmen;
    // guthaben >= -ueberziehungsrahmen

    // einzahlen addiert summe zu guthaben; summe >= 0
    public void einzahlen(int summe)
        { guthaben = guthaben + summe; }

    // abheben zieht summe von guthaben ab;
    // summe >= 0; guthaben+ueberziehungsrahmen >= summe
    public void abheben(int summe)
        { guthaben = guthaben - summe; }
}
```

Beispiel zu History-Constraints

```
public class Transaction {
    private int started = 0, committed = 0, aborted = 0;
    // number of corresponding transactions
    // can only increase one by one    (server-controlled)
    // started >= committed + aborted          (invariant)

    // for each invocation of start() returning x, either
    // commit(x) or abort(x) must be invoked exactly once
    public int start() { ...; return started++; }
    public void commit(int x) { ...; committed++; }
    public void abort(int x)  { ...; aborted++; }
}
```

Aufgabe: Invariante vs. History-Constraint

Kann man Server-kontrollierte History-Constraints durch Invarianten ersetzen?

A: Ja, weil sie sich auf Variablen beziehen und vom Server kontrolliert werden.

B: Ja, weil sie wie Invarianten eine Form von Nachbedingungen darstellen.

C: Nein, weil nur Clients die Aufrufreihenfolge kennen.

D: Nein, weil sie prinzipiell nicht vor Methodenausführung überprüfbar sind.

Aufgabe: Vorbedingung vs. History-Constraint

Sind Client-kontrollierte History-Constraints durch Vorbedingungen ersetzbar?

A: Ja, weil wie bei Vorbedingungen der Client dafür zuständig ist.

B: Nein, weil Aufrufreihenfolgen im Server nicht sichtbar sind.

C: Nur in der sequenziellen Programmierung, da keine Synchronisation nötig.

D: Nein, weil sonst hätte man dafür keine eigene Kategorie eingeführt.

Typen und Zusicherungen

Zusicherungen gehören zu nominalen Typen

Objekttyp besteht aus	Schnittstelle (= Signatur) Name von Klasse oder Interface Zusicherungen

Änderung von Zusicherung = Typänderung

→ Auswirkungen auf andere Programmteile

Genauigkeit von Zusicherungen

Genauigkeit durch Programmierer(inn)en bestimmbar:

genau: große Abhängigkeit zwischen Client und Server

ungenau: kleine Abhängigkeit zwischen Client und Server

Tipp: keine versteckten Zusicherungen

Tipp: schwache Abhängigkeiten (= wenige Zusicherungen)

Ersetzbarkeit und Verhalten (klassisch)

U ist nur dann Untertyp von T wenn gilt:

Vorbedingungen in Untertypen sind **schwächer oder gleich**

bei Vererbung: Verknüpfung mit **oder**

Nachbedingungen in Untertypen sind **stärker oder gleich**

bei Vererbung: Verknüpfung mit **und**

Invarianten in Untertypen sind **stärker oder gleich**

aber wenn Variable von außen schreibbar, dann Invarianten **gleich**

bei Vererbung: Verknüpfung mit **und** (wenn nicht gleich)

Ersetzbarkeit und History-Constraints

U ist nur dann Untertyp von T wenn gilt:

Wenn **Server-kontrollierte History-Constraints** in T verhindern,
dass eine Variable vom Zustand X in den Zustand Y kommt,
dann müssen dies auch Constraints in U tun.

> U kann Zustandsänderungen stärker einschränken als T
> aber wenn Variable von außen schreibbar, dann **gleiche** Constraints

Jede von **Client-kontrollierten History-Constraints** in T erlaubte
Aufrufreihenfolge muss auch in U erlaubt sein.

> TraceSet(T) $\subseteq$ TraceSet(U)
> Menge aller Clients betrachten, nicht nur einen einzelnen Client

Zusicherungen und Ersetzbarkeit – Beispiel

```
public class Set {
    public void insert(int x)
        // inserts x into set iff not already there
        // x is in set immediately after invocation
        { ... }
    public boolean inSet(int x)
        // returns true if x is in set, otherwise false
        { ... }
    ...
}

public class SetWithoutDelete extends Set {
    // elements in the set always remain in the set
}
```

Beispiel fortgesetzt

```
Set s = ...;
s.insert(42);
doSomething(s);
if (s.inSet(42)) { doSomeOtherThing(s); }
else { doSomethingElse(); }


SetWithoutDelete s = ...;
s.insert(42);
doSomething(s);
doSomeOtherThing(s); // s.inSet(42) always returns true
```

Faustregeln zu Zusicherungen

Zusicherungen sollen

- stabil sein (vor allem an Wurzel der Typhierarchie)
- keine unnötigen Details festlegen
- explizit im Programm stehen
- unmissverständlich formuliert sein
- während Programmentwicklung ständig überprüft werden