Softwaretechnik und Programmierparadigmen

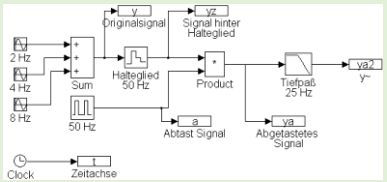
VL 05: Object Constraint Language (OCL)

Prof. Dr. Sabine Glesner
FG Programmierung eingebetteter Systeme
Technische Universität Berlin

Student Assistant wanted for CorMoran

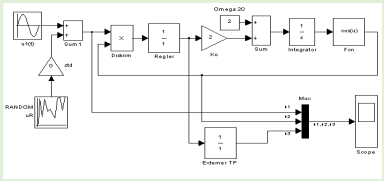
Research Goal

Simulink



Transformation

Simulink



Behavioural equivalent for all models?

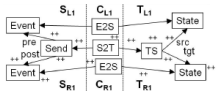
Working Areas

1) Simulink Semantics enhancement

$$\frac{}{\langle r,\sigma\rangle \xrightarrow{o} r}\ \text{CTE} \qquad \frac{}{\langle \ell,\sigma\rangle \xrightarrow{o} \sigma(\ell)}\ \text{VAR}$$

$$\frac{\langle e_1,\sigma\rangle \xrightarrow{o} \text{true} \quad \langle e_2,\sigma\rangle \xrightarrow{o} r_2}{\langle \text{if } (e_1,e_2,e_3),\sigma\rangle \xrightarrow{o} r_2}\ \text{THEN}$$

2) Verification Techniques

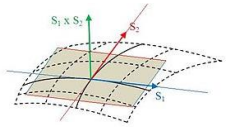


3) Implementation of transformations and test system

Skills

1)

- Interest in formal methods, semantics, logic
- beneficial: differential equations

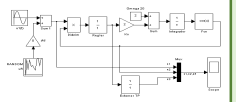


2)

- Interest in formal methods, semantics, logic
- beneficial: Graph theory, Simulink knowledge

3)

- Knowledge in Eclipse Modeling Framework & tools
- beneficial: Simulink knowledge



Interest?

Send email to Sebastian.Schlesinger@tu-berlin.de

Application Deadline: Nov 18, 2014
Contract Period: Nov 2014 – Dec 2015

Klassendiagramme und Bedingungen

- Klassen, Assoziationen, Kardinalitäten, etc. schränken die Objektmodelle ein
- Allerdings: Keine Möglichkeit komplexe Bedingungen oder Invarianten zu beschreiben oder Methoden zu spezifizieren
- Mögliche Lösung: Informelle Annotationen an Klassendiagramme
- Bessere Lösung: Standardisierte Sprache zum Beschreiben von Bedingungen

Object Constraint Language - Überblick

- Ergänzung der UML, standardisiert durch die OMG
- Mitte der 90er von IBM entwickelt und dann in die UML aufgenommen
- Derzeitige Version OCL 2.4
- Formale Sprache zur Beschreibung von Constraints auf UML Modellen
- Typisierte und typsichere Sprache
- Logische Sprache → seiteneffektfrei!

- Kernsprache von Modelltransformationssprachen wie QVT

Übersicht

- OCL Ausdrücke
- Constraints auf Klassendiagrammen
- Design-by-Contract
- OCL als Query Language

Ausdrücke und Bedingungen

- Ausdrücke
  - Werte/ Variablen
  - Operationen auf Werten (keine Seiteneffekte!)
  - Sind getypt: Real, Integer, String, Boolean, …
- Bedingungen
  - Invarianten (von Klassen): Bedingungen, die im Kontext einer Klasse immer wahr sein müssen
  - Vor-/Nachbedingungen (von Methoden): müssen vor bzw. nach Ausführung erfüllt sein
  - Guard (von Transitionen, z.B. in Aktivitätsdiagrammen): Anwendungsbedingung

Kontext und Auswertung

- Jede OCL Bedingung ist an einen Kontext gebunden
  - Invariante bzgl. Klasse, Interface, etc.
  - Vor-/Nachbedingung bzgl. Operation
- Ein Ausdruck wird bzgl. eines Objektgraphs ausgewertet, der den aktuellen Zustand des Systems widerspiegelt

OCL Typen: Überblick

- Klassentypen
- Basistypen
  - Boolean: true, false
  - Integer: -4, -1, 0, 1, …
  - Real: -1.4142, 2.1234, 0, 3, …
  - String: „Heute ist ein kalter Tag“, …
- Collection Typ: Set, Sequence, Bag
- Aufzählungstypen

Operationen auf Basistypen

- Boolean: and, or, not, implies, …
- Integer/Real: *,+,-,/,abs,…
- String: size, toUpper, concat, …

- Notation von Operationen
  - Infix
  - Oft auch postfix mit „.“ als Trennsymbol
- Beispiele
  - x + y
  - x.add(y)

OCL Ausdrücke: Navigationsausdrücke

- Navigiere von einem Objekt zu assoziierten Objekten
- Über navigierbare Assoziationen kann mit dem Punkt „.“ Operator navigiert werden
  - object.associationEndName
- Liefert einzelnes Objekt (nur bei Multiplizität „0..1“ oder „1“) oder Menge von Objekten
  - Einzelnes Objekt darf aber immer auch als einelementige Menge interpretiert werden
- Mengenausdrücke werden mit einem Pfeil „->“ vorangestellt

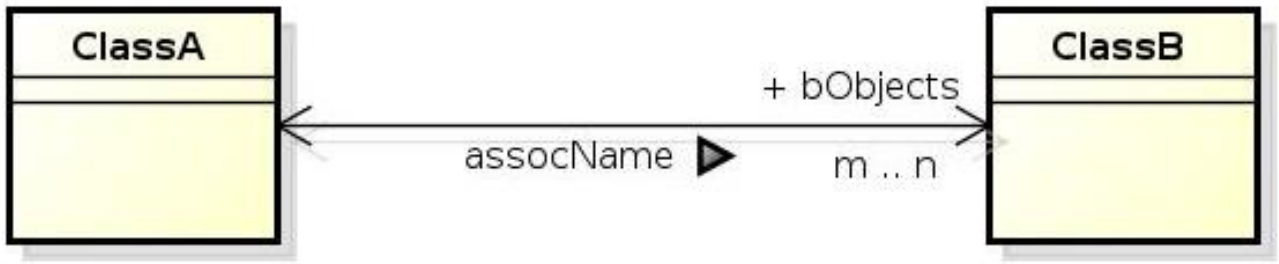
Navigationsenden

- Wenn das Assoziationsende explizit benannt wurde, kann darüber navigiert werden
  - Im Kontext von ClassA: **self.bObjects**

Liefert einzelnes Objekt oder Menge – abhängig von m und n

- Wenn Assoziationsende nicht spezifiziert wurde, kann der kleingeschriebene Klassenname als Assoziationsendenname verwendet werden, wenn es nicht zu Mehrdeutigkeiten kommt
  - Im Kontext von ClassB: **self.classA**

Liefert eine Menge



OCL Ausdrücke: Mengenausdrücke

- **select/reject**: Wählen/Ausschließen derjenigen Elemente aus einer Menge, auf die ein bestimmtes Prädikat zutrifft (analog zu „filter“)
- **collect**: Menge (genauer Bag – Elemente dürfen mehrfach vorkommen) von abgeleiteten Werten bzgl. einer Basismenge (analog zu „map“)
- **iterate**: Iterieren über eine Menge in nichtdefinierter Reihenfolge mit Verwendung eines Akkumulators (analog zu „reduce“)
- **forall/exists**: Trifft ein Prädikat auf alle/ein Element(e) einer Menge zu

Semantik von Mengenausdrücken

- C->forall(c:Type | P(c)) -- $\forall\, c \in C.\, P(c)$
- C->exists(c:Type | P(c)) -- $\exists\, c \in C.\, P(c)$
- C->select(c:Type | P(c)) -- $\{c \in C \mid P(c)\}$
- C->collect(c:Type | E(c)) -- $\{e \mid \exists\, c \in C.\ e = E(c)\}$

- C->including(c) -- $C \cup \{c\}$
- C->union(C‘) -- $C \cup C'$
- C->includes(c) -- $c \in C$
- C->includesAll(C‘) -- $C' \subseteq C$

ähnlich wie **filter** in funktionalen Sprachen

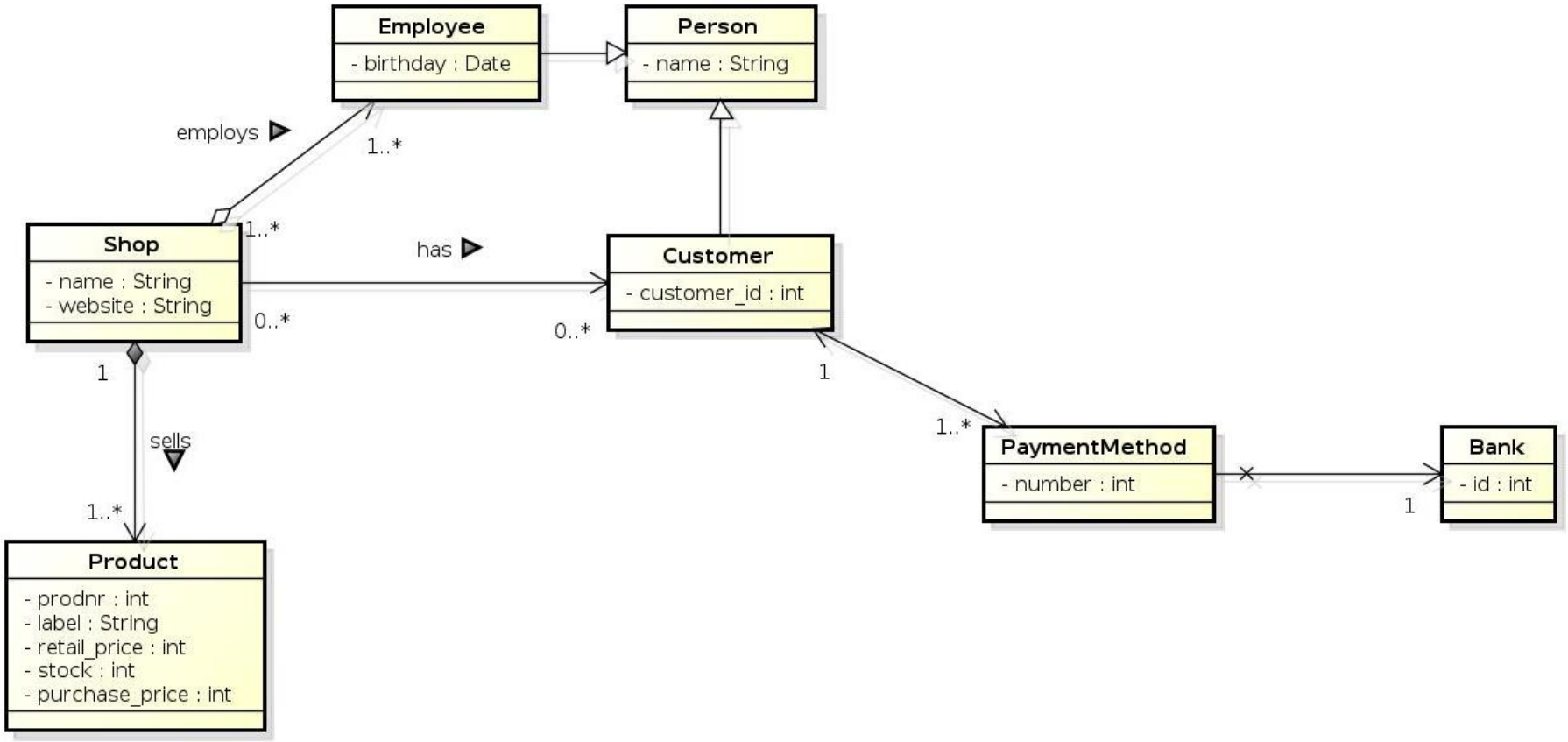
ähnlich wie **map** in funktionalen Sprachen

Liefert eine Bag!

Syntactic Sugar: let-in Ausdrücke

- Bei komplexen Ausdrücken können Teilausdrücke mit „let … in“ ausgelagert werden
- Mehrfach vorkommende Ausdrücke können einmal am Anfang definiert werden

Beispielklassenmodell



Beispiele für Ausdrücke

- Im Kontext von einem Shop
  - Menge aller Produkte
    self.product
  - Menge aller Produkte von Levis
    self.product->select(p:Product | p.label = „Levis“)
  - Menge (Bag) aller Produktnummern
    self.product->collect(p : Product | p.prod_nr)

Beispiele für Boolesche Ausdrücke

- Im Kontext von Customer
  - Geburtsjahr liegt vor 1996
    self.birthday.year() < 1996
  - Alle Zahlmethoden haben unterschiedliche Nummern
    self.paymentMethod->size()
    == self.paymentMethod->collect(number)
    ->asSet()->size()

Die collect-Operation liefert keine Menge, sondern eine Bag(Elemente können mehrfach vorkommen). Mit der Operation asSet() erhalten wir wieder eine Menge.

Übersicht

- OCL Ausdrücke
- Constraints auf Klassendiagrammen
- Design-by-Contract
- OCL als Query Language

Bedingungen

- Ausdrücke vom Typ Boolean
- Werden in 3-wertiger Logik interpretiert (true, false, undefinded)

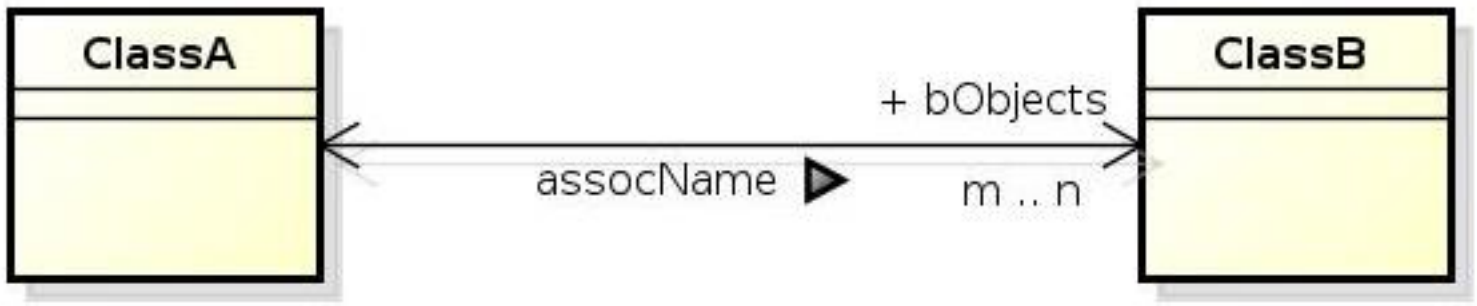
Invarianten

context [instName:] TypeName **inv** [exprName:] expr

- Im Kontext der Klasse „TypeName“
- Ggf. bezogen auf eine konkrete Instanz „instName“
- soll der Ausdruck expr immer zu true evaluieren
- Ggf. kann der Ausdruck mit „exprName“ benannt werden

Ausdrücken von Multiplizitäten mit OCL

- OCL ermöglicht eine präzise Definition von UML Konzepten



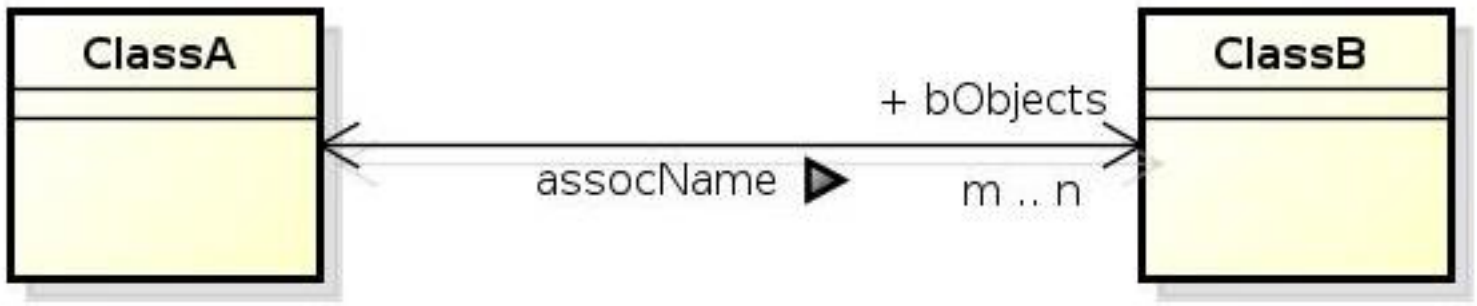
context ClassA

inv: self.bObjects->size()>=m and
self.bObjects->size()<=n

Da der Kontext klar ist, kann „self“ auch weggelassen werden.

Ausdrücken von Multiplizitäten mit OCL

- OCL ermöglicht eine präzise Definition von UML Konzepten



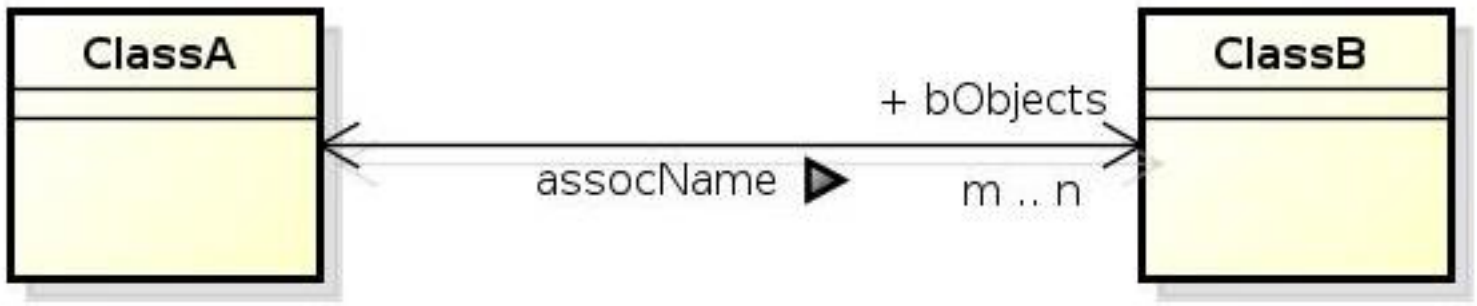
context ClassA

inv: bObjects->size()>=m and
bObjects->size()<=n

Wir können über den Instanznamen auch eine neue Bezeichnung für „self“ einführen.

Ausdrücken von Multiplizitäten mit OCL

- OCL ermöglicht eine präzise Definition von UML Konzepten



context ca : ClassA

inv: ca.bObjects->size()>=m and
ca.bObjects->size()<=n

Wir können über den Instanznamen auch eine neue Bezeichnung für „self“ einführen.

Beispiel für Invariante

- Der Online-Shop hat jederzeit mindestens 5 Mitarbeiter

  context Shop
  inv: self.employee->size() >= 5

Komplexeres Beispiel

- Alle Mitarbeiter sind vor 1996 geboren

context Shop
inv: self.employee->forall(p : Employee |
p.birthday.year() < 1996)

Übersicht

- OCL Ausdrücke
- Constraints auf Klassendiagrammen
- Design-by-Contract
- OCL als Query Language

Design by Contract

- Entwickelt von Bertrand Meyer in den 80ern
  - Integration in Eiffel Programmiersprache
  - In vielen anderen Sprachen (auch Java) verfügbar
- Basiert auf Hoare Kalkül (siehe nächste Vorlesung)
  {P} C {Q}
  Wenn P in einem Zustand gilt und Code C in diesem gestartet wird, dann gilt Q danach
- Contracts können als detaillierte Interface-Spezifikationen aufgefasst werden
- Typen (in Programmiersprachen) als (sehr) eingeschränkte Form von Contracts

Wozu Contracts?

- Contracts können als Requirements Spezifikation dienen
- Erleichtert/Ermöglicht Verifikation und Validierung
- Ermöglicht es statisch zu bestimmen, ob bestimmte Abfolgen von Operationen möglich sind

Vor- und Nachbedingungen

- Beschreibung von Zuständen vor und nach Ausführung der Methode über Prädikate
- Vorbedingung: Welche Bedingungen müssen für Inputs und Systemzustand gelten
- Nachbedingung: Welche Bedingungen müssen für Outputs und Systemzustand gelten
- Klasseninvarianten müssen bewahrt werden!
  → Implizite weitere Bedingungen für Vor- und Nachbedingung
  → Müssen bei späterer Implementierung explizit berücksichtigt werden

Vor- und Nachbedingungen

- **context** Typename::operationName(param1: Type1, …): ReturnType
  pre [preName]: preExpr
  post [postName]: result = postExpr
- In der Postcondition kann auf einen Wert vor Ausführung der Operation mit dem Zusatz „@pre“ zugegriffen werden

Beispiel

- Aufnehmen eines neuen Kunden
  context Shop::addCustomer(p) : void
  pre: not self.customer->includes(p)
  post: self.customer = self.customer@pre ->
  including(p)

Problem: keine direkte Erzeugung neuer Objekte

Ende: 13.11.2014

- Nochmal Aufnehmen eines neuen Kunden

context Shop::addCustomer(n,cid) : void
pre: n ≠ „“
and not(customer->collect(customer_id)
->exists(x | x==cid))
post: customer->size() = customer->size()@pre+1
and customer->exists(p | p.customer_id == cid and
p.name == n)

Problem: keine direkte Erzeugung neuer Objekte

- Nochmal Aufnehmen eines neuen Kunden

context Shop::addCustomer(n,cid) : void
pre: n ≠ „“
and not(customer->collect(customer_id)
->exists(x | x==cid))
post: customer->exists(p | p.customer_id == cid and
p.name == n and
not Customer.allInstances()@pre-> includes(p)
)

Problem: keine direkte Erzeugung neuer Objekte

- Nochmal Aufnehmen eines neuen Kunden

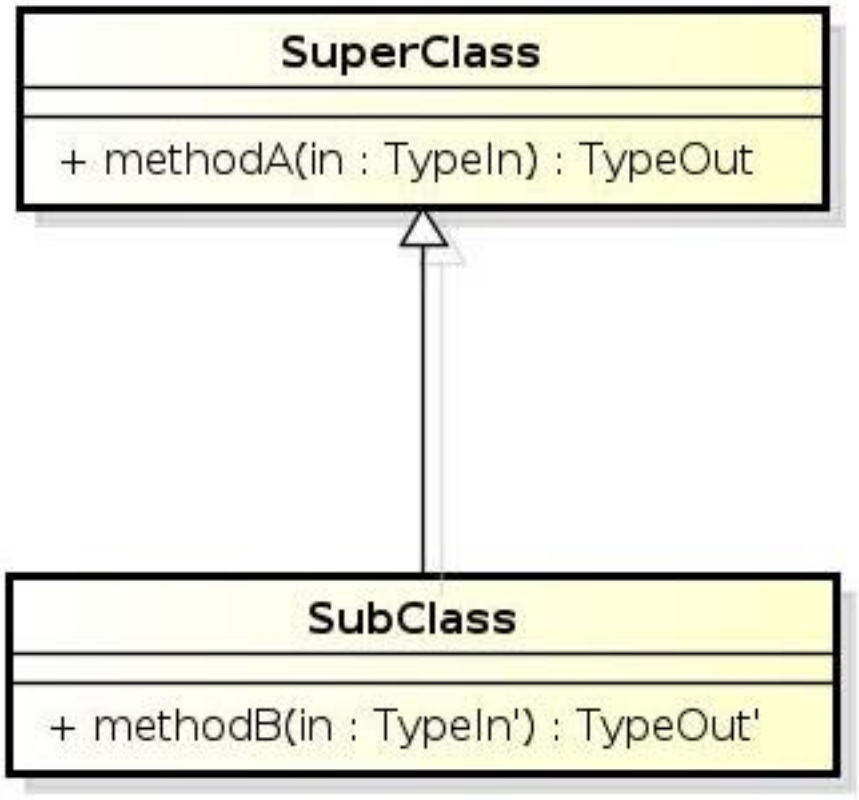
context Shop::addCustomer(n,cid) : void
pre: n ≠ „“
and not(customer->collect(customer_id)
->exists(x | x==cid))
post: customer->exists(p | p.customer_id == cid and
p.name == n and
p.oclIsNew())

Nur in Postconditions erlaubt. Drückt aus, dass eine neue Instanz geschaffen wurde.

Informationen zu QISPOS-Anmeldung

- Anmeldefrist: 28.11.2014
- Es sollten sich jetzt alle (außer WiIng-Studierende) elektronisch anmelden können.
- An alle, die sich (immer noch) nicht elektronisch anmelden können: Bitte meldet Euch mit einem „gelben Zettel“ im Prüfungsamt an und gebt diesen in unserem Sekretariat ab

Contracts und Vererbung



Contracts und Vererbung

- Substitutionsprinzip
  - Wenn vor der Ausführung einer Methode in der Unterklasse die Vorbedingungen der entsprechenden Methode der Oberklasse gelten, dann muss die Methode der Unterklasse ausführbar sein und anschließend die Nachbedingungen der Oberklasse garantieren.
- Vorbedingungen dürfen nicht verschärft werden
- Nachbedingungen dürfen nicht aufgeweicht werden

Contracts und Vererbung

- TypeIn‘ ist ein Obertyp von TypeIn
- TypeOut‘ ist ein Untertyp von TypeOut
- Vorbedingungen von methodB werden von Vorbedingungen von methodA impliziert
- Nachbedingungen von MethodB implizieren Nachbedingungen von MethodA

- Diese Art der Vererbungs-Konfomität wird auch *Kontra-Kovarianz* genannt
- Entsprechend lassen sich auch andere Varianten von Konformität definieren
- Ko-Kontravarianz spielt zum Beispiel bei *Spezialisierung* eine Rolle

Beispiel

- **context** Collection::some_elem():int
  pre: self.coll /= emptySet
  post: coll.elem(result)
- **context** Ordered Collection::first_elem():int
  pre: true
  post: result = coll.fst()

Übersicht

- OCL Ausdrücke
- Constraints auf Klassendiagrammen
- Design-by-Contract
- OCL als Query Language

OCL als query language

- Queries ermitteln gewisse Informationen aus einem Modell
- stellen keine Invarianten dar
  - Werte können sich ändern
- stellen keine Vor-/Nachbedingungen dar
  - Modell wird nicht geändert

Derived Values (von einem Attribut)

- Wert eines abgeleiteten Attributs (*derived value*) ergibt sich immer aus den Werten anderer Attribute

context Employee::age : int

derive: years(today() – self.birthday)

OCL queries

- Schlüsselwort **body**, um querys (reine Abfragen auf dem Modell) zu spezifizieren

- Sortierte Auflistung aller Mitarbeiter

  context Shop:: allEmployees : set(Employee)
  body: self.employee->sortBy(self.name)

Bedeutung des select Statements

- Select dient der Ableitung einer Teilmenge
- Alle Kunden, deren Name mit einem bestimmten Buchstaben beginnt

  context Shop :: getCustomers(l : char) : set(Customer)
  body: self.customer->select(c:Customer | c.name.beginsWith(l))

- Derartige Abfragen finden sich häufig in Datenbanksystemen

Übersetzung von OCL queries

- OCL Queries können leicht in SQL Queries überführt werden
- SELECT * FROM Customers
  WHERE name.beginsWith(l);

name	customer_id	payment_method
John Locke	4247	…
Jean-Jacques Rousseau	9764	…
Voltaire	1239	…
David Hume	3264	…
Immanuel Kant	7324	…
Denis Diderot	4653	…

Zusammenfassung

- OCL ist umfangreiche Sprache zur Präzisierung von UML Modellierungskonzepten
- Invarianten (Klassenconstraints)
- Vor- und Nachbedingungen (Spezifikation von Methoden)
- OCL als mächtige Query-Language
  - Datenbankoperationen können direkt abgeleitet werden

Literatur

- OCL Reference Manual
  http://www.omg.org/spec/OCL/2.4/PDF
  - sehr umfangreich, aber leichter Aufbau
- Goos, Zimmermann. Vorlesungen Über Informatik – Band 2. 2006