

Softwaretechnik und Programmierparadigmen WiSe 2022/2023

Prof. Dr. Sabine Glesner
Milko Monecke
Simon Schwan

Übungsblatt 13

Beispiellösung

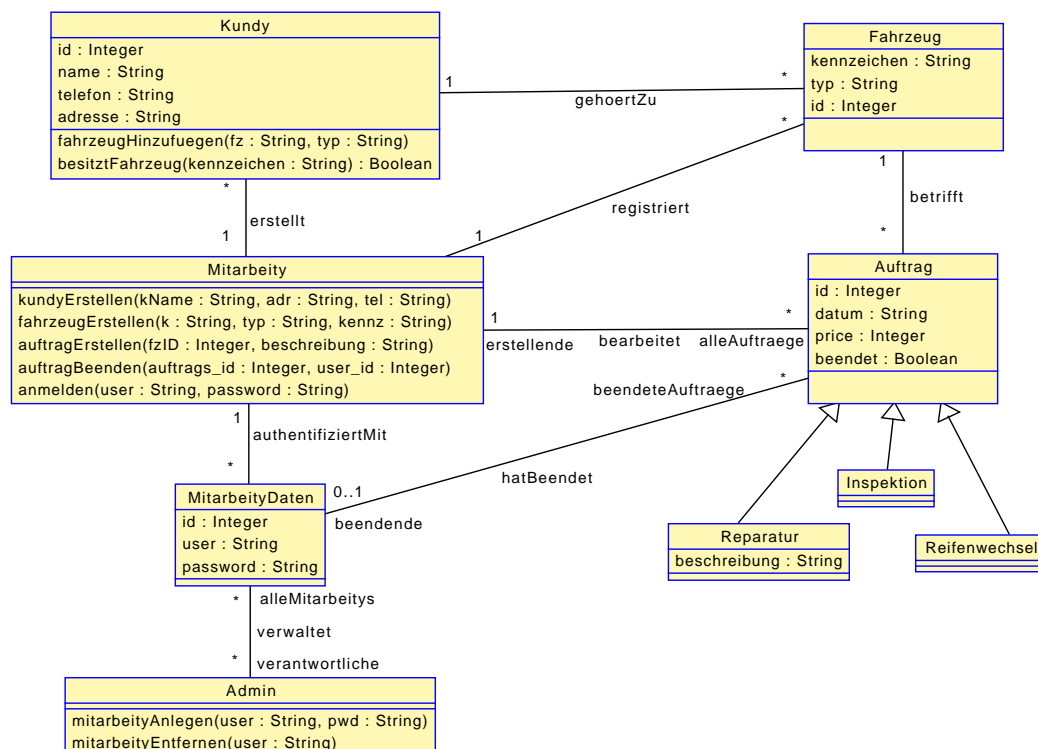


Abbildung 1: Klassendiagramm der Autowerkstatt wie bei autowerkstatt.use

Wie auch auf dem letzten Übungsblatt könnt ihr auch alle in dieser Übung entwickelten OCL Invarianten und Contracts mit dem Tool USE-OCL der Uni Bremen testen.

Um Invarianten mit USE-OCL zu evaluieren, könnt ihr den "Create class invariant view" button verwenden (mit dem gelben Blitz). Dort seht ihr für jede Invariante, ob sie für euer aktuelles Objektdiagramm erfüllt ist. Durch Doppelklick auf eine Zeile könnt ihr außerdem sehen, wie die einzelnen Teilbedingungen einer Invariante belegt sind.















Um Contracts mit USE-OCL zu evaluieren, könnt ihr einzelne Operationen über die Kommandozeile ausführen. Hierzu werden in der Konsole die Operationen **!openter** und **!opexit** verwendet. Dazwischen muss die Operation simuliert werden!

Beispiel:

```
use> !openter m kundeErstellen('bob','strasse 7, 1000 berlin','03012345678')
precondition 'pre3' is true
precondition 'pre4' is true
use> !new Kunde('c3')
use> !c3.name:='bob'
use> !c3.adresse:='strasse 7, 1000 berlin'
use> !c3.telefon:='03012345678'
use> !c3.id:=4
use> !insert(m,c3) into erstellt
use> !opexit
postcondition 'post3' is true
```

Aufgabe 1: OCL Invarianten

Invarianten sind Bedingungen, die zu jeder Zeit und für jedes Objekt einer Klasse gelten müssen. Formalisiert die folgenden Invarianten für das gegebene Klassendiagramm aus Abbildung 1. Welche Invarianten sind in dem Zustand der Abbildung 2 verletzt? Korrigiert den Zustand, damit alle Invarianten erfüllt sind.

- a) Die ID jedes Mitarbeiters muss größer als 0 sein. 
- b) Der Name jedes Mitarbeiters darf kein leerer String sein.  
- c) Die ID's aller Mitarbeiters müssen eindeutig sein. 
- d) Zu jedem Fahrzeug darf es höchstens einen Inspektionsauftrag geben, der noch nicht abgeschlossen ist.  
- e) Das Besitzer eines Fahrzeugs hat dieses auch in der Menge seiner Fahrzeuge. 
- f) Ein Auftrag muss immer vom Typ Reparatur, Inspektion oder Reifenwechsel sein. 
- g) Jeder Auftrag, der beendet ist, muss einem Mitarbeiter zugewiesen sein, der ihn beendet hat.  
- h) Zu jedem Fahrzeug gibt es höchstens einen offenen Auftrag von jedem Typ. 
- i) Alle Aufträge für Fahrzeuge vom Typ "jaguar" sollen mehr als 1000 Euro kosten. 
- j) Für jede beendete Inspektion existiert mindestens ein Reparaturauftrag für das gleiche Fahrzeug.  

Lösung:

- a) `context MitarbeiterDaten inv: self.id > 0`
- b) `context MitarbeiterDaten inv: user <> '' and user <> null`
- c) `context MitarbeiterDaten inv: MitarbeiterDaten.allInstances()->select(m:MitarbeiterDaten | (self.id = m.id))->size() = 1`
- d) `context Fahrzeug inv: self.auftrag -> select(a:Auftrag | not a.beendet and a.ocIsTypeOf(Inspektion)) -> size() <= 1`
- e) `context Fahrzeug inv: self.kund.fahrzeug -> includes(self)`
- f) `context Auftrag inv: self.ocIsTypeOf(Reparatur) or self.ocIsTypeOf(Inspektion) or self.ocIsTypeOf(Reifenwechsel)`
- g) `context Auftrag inv: self.beendet implies self.beendende <> Undefined`

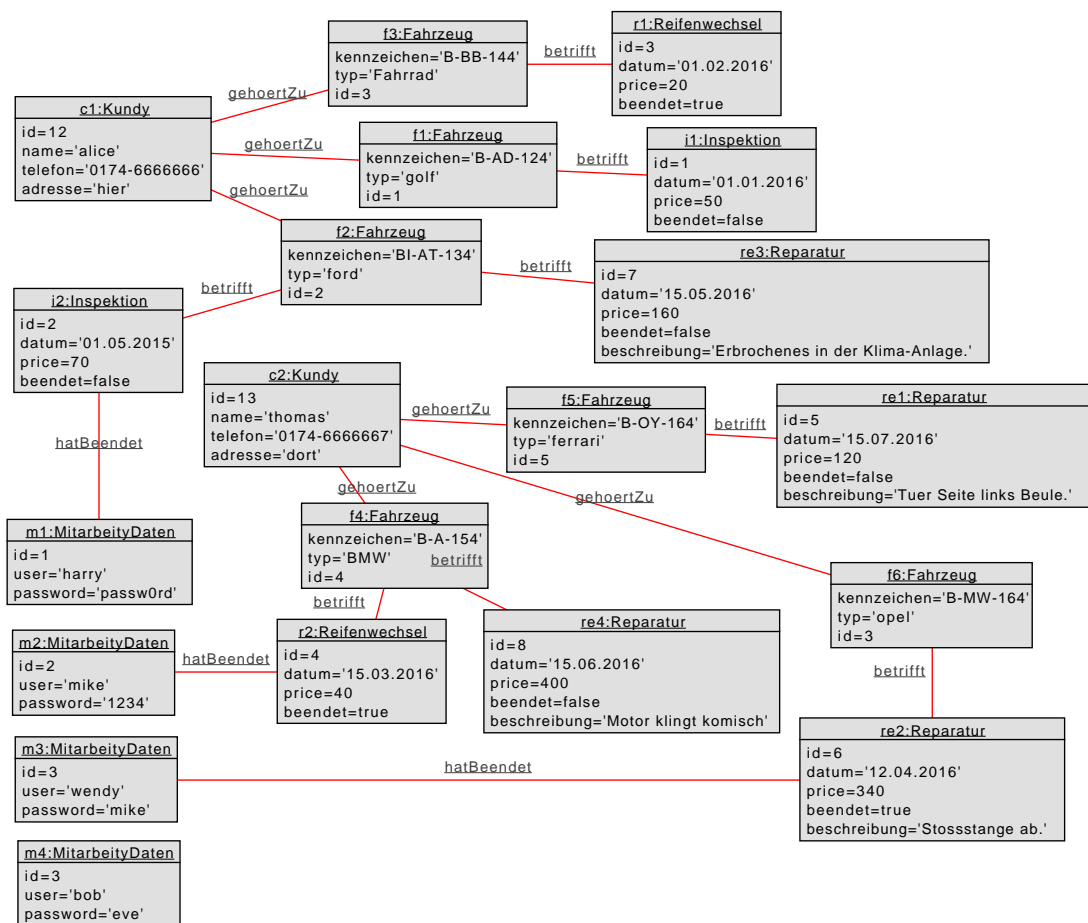


Abbildung 2: Beispiel-Objektdiagramm wie bei autowerkstatt.soil






- h) context Fahrzeug inv: let
 aufts = self.auftrag -> select(a:Auftrag | not a.beendet)
 in
 aufts -> select(oclIsTypeOf(Reparatur)) -> size() <= 1 and
 aufts -> select(oclIsTypeOf(Inspektion)) -> size() <= 1 and
 aufts -> select(oclIsTypeOf(Reifenwechsel)) -> size() <= 1
- i) context Fahrzeug inv: typ = 'jaguar' implies
 self.auftrag -> forAll(a:Auftrag | a.price > 1000)
- j) context Inspektion inv: self.beendet implies
 self.fahrzeug.auftrag -> exists(a:Auftrag | a.oclIsTypeOf(Reparatur))

Aufgabe 2: OCL Contracts

Vor- und Nachbedingungen von Operationen beschreiben den Systemzustand und die Eingabe- bzw. Ausgabeparameter. Zusammen mit Invarianten lässt sich dadurch formal feststellen, ob z.B. eine bestimmte Abfolge von Operationen möglich ist. Formalisiert die folgenden Vor- und Nachbedingungen. Benutzt hierzu das gegebene Klassenmodell. Wie könnte man sicherstellen, dass eine Implementierung die Contracts und Invarianten erfüllt?

Hinweis

Ihr könnt davon ausgehen, dass kein Eingabeargument mit **null** belegt ist.

- a) Die Operation **kundyErstellen** erhält die Kundydaten Name, Adresse und Telefonnummer. Es dürfen keine leeren Daten gespeichert werden und das Kundy darf auch nicht mehrfach existieren. Außerdem muss eine eindeutige ID generiert werden. 
- b) Die Operation **fahrzeugHinzufuegen** in der Klasse Kundy wird zusätzlich benötigt.   Auch sie soll spezifiziert werden und sie erhält ein Kennzeichen als String. Das Fahrzeug mit dem übergebenen Kennzeichen wird neu erstellt. Ein anderes Fahrzeug mit dem gleichen Kennzeichen darf vorher nicht existieren.
- c) Die Operation **besitztFahrzeug** in der Klasse Kundy soll prüfen ob ein Fahrzeug   mit einem bestimmten Kennzeichen existiert und das Ergebnis als Bool zurückgeben.

Lösung: Siehe auch das USE-Modell (auf ISIS verfügbar).

- a) context Mitarbeity::kundyErstellen(kName: String, adr: String, tel: String)
 pre:
 kName <> '' and adr <> '' and tel <> '' and
 not Kunde.allInstances() -> exists(k: Kunde | k.name = kName)

post:

```
self.kundy -> exists (k:Kunde | k.name =kName and k.adresse = adr and  
k.telefon = tel and k.ocllsNew() and  
Kunde.allInstances() -> select(k1:Kunde | k1.id= k.id) -> size() = 1)
```

b) **context** Kundy::fahrzeugHinzufuegen(fz:String, typ:String)

pre:

```
fz <> '' and fz <> null and  
not Fahrzeug.allInstances()->exists(kennzeichen=fz)
```

post:

```
Fahrzeug.allInstances()->exists (f |  
f.kennzeichen=fz and f.typ=typ and f.ocllsNew() and  
Fahrzeug.allInstances().id->count(f.id) = 1 and  
self.fahrzeug = self.fahrzeug@pre->including(f) and  
mitarbeits.fahrzeug->includes(f)  
)
```

c) **context** Kundy::besitztFahrzeug(kennzeichen: String): Boolean

pre:

```
true
```

post:

```
result = self.fahrzeug -> exists(f:Fahrzeug | f.kennzeichen = kennzeichen)
```