



Technische Universität Berlin

Software and Embedded Systems Engineering Group

Prof. Dr. Sabine Glesner

www.sese.tu-berlin.de Sekr. TEL 12-4 Ernst-Reuter-Platz 7 10587 Berlin



Softwaretechnik und Programmierparadigmen WiSe 2023/2024

Prof. Dr. Sabine Glesner
Simon Schwan
Julian Klein

Übungsblatt 15

Beispiellösung

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 createCustomer('bob', 'strasse 7, 1000 berlin', '03012345678')
precondition 'pre3' is true
precondition 'pre4' is true
use> !new Customer('c3')
use> !c3.name:='bob'
use> !c3.address:='strasse 7, 1000 berlin'
use> !c3.telefon:='03012345678'
use> !c3.id:=4
use> !insert(m,c3) into creates
use> !opexit
postcondition 'post3' is true
```

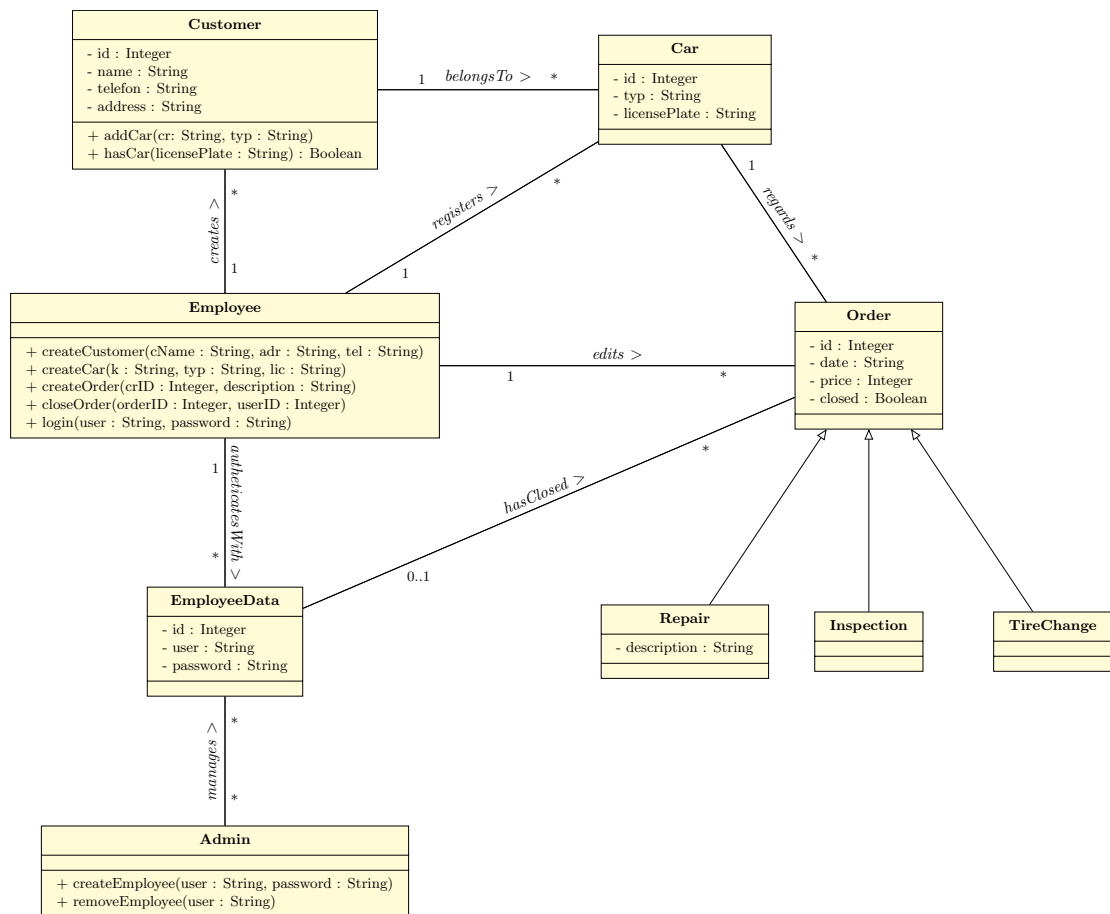
















Abbildung 1: Klassendiagramm der Autowerkstatt wie bei `autowerkstatt.use`

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.

- Die ID jedes Mitarbeitenden muss größer als 0 sein. 
- Der Name jedes Mitarbeitenden darf kein leerer String sein.  
- Die ID's aller Mitarbeitenden müssen eindeutig sein. 
- Zu jedem Fahrzeug darf es höchstens einen Inspektionsauftrag geben, der noch nicht abgeschlossen ist.  
- Der Besitzer/die Besitzerin 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 Mitarbeitenden 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 EmployeeData inv: self.id > 0`
- b) `context EmployeeData inv: user <> '' and user <> null`
- c) `context EmployeeData inv: EmployeeData.allInstances()->
 select(m:EmployeeData | (self.id = m.id))->size() = 1`
- d) `context Car inv: self.order -> select(a:Order | not a.closed
 and a.oclIsTypeOf(Inspection)) -> size() <= 1`
- e) `context Car inv: self.customer.car -> includes(self)`
- f) `context Order inv: self.oclIsTypeOf(Repair) or
 self.oclIsTypeOf(Inspection) or self.oclIsTypeOf(TireChange)`
- g) `context Order inv: self.closed implies self.closing <> Undefined`
- h) `context Car inv: let
 aufts = self.order -> select(a:Order | not a.closed)
 in
 aufts -> select(oclIsTypeOf(Repair)) -> size() <= 1 and
 aufts -> select(oclIsTypeOf(Inspection)) -> size() <= 1 and
 aufts -> select(oclIsTypeOf(TireChange)) -> size() <= 1`
- i) `context Car inv: typ = 'jaguar' implies
 self.order -> forAll(a:Order | a.price > 1000)`
- j) `context Inspection inv: self.closed implies
 self.car.order -> exists(a:Order | a.oclIsTypeOf(Repair))`

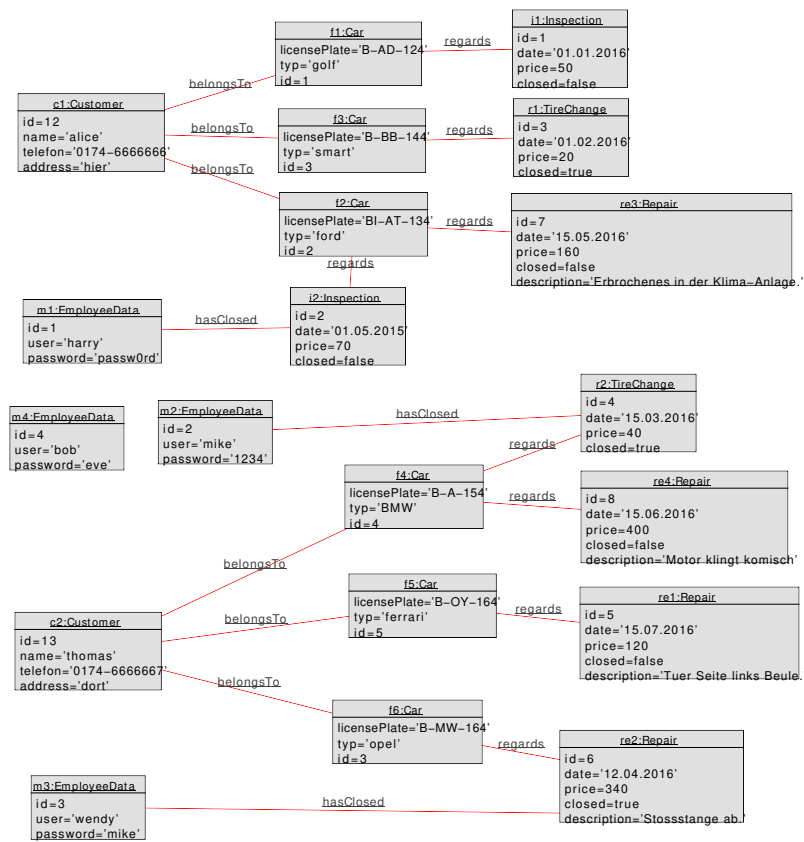







Abbildung 2: Beispiel-Objektdiagramm wie bei autowerkstatt.soil

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 `createCustomer` erhält die Daten Name, Adresse und Telefonnummer des Kunden/der Kundin. Es dürfen keine leeren Daten gespeichert werden und der Kunde/die Kundin darf auch nicht mehrfach existieren. Außerdem muss eine eindeutige ID generiert werden. 
- b) Die Operation `addCar` in der Klasse `Customer` 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 `hasCar` in der Klasse `Customer` 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** Employee::createCustomer(cName: String, adr: String, tel: String)
 pre:
 cName <> '' and adr <> '' and tel <> '' and
 not Customer.allInstances() -> exists(k: Customer | k.name = cName)

 post:
 self.customer -> exists (k:Customer | k.name =cName and k.address =
 adr and
 k.telefon = tel and k.ocllsNew() and
 Customer.allInstances() -> select(k1:Customer | k1.id= k.id) -> size()
 = 1)
- b) **context** Customer::addCar(cr:String, typ:String)
 pre:
 cr <> '' and cr <> null and
 not Car.allInstances()->exists(licensePlate=cr)
 post:

```

    Car.allInstances()->exists (f |
      f.licensePlate=cr and f.typ=typ and f.oclIsNew() and
      Car.allInstances().id->count(f.id) = 1 and
      self.car = self.car@pre->including(f) and
      employee.car->includes(f)
    )

```

c) **context** Customer::hasCar(licensePlate: String): Boolean

pre:

 true

post:

 result = self.car -> exists(f:Car | f.licensePlate = licensePlate)