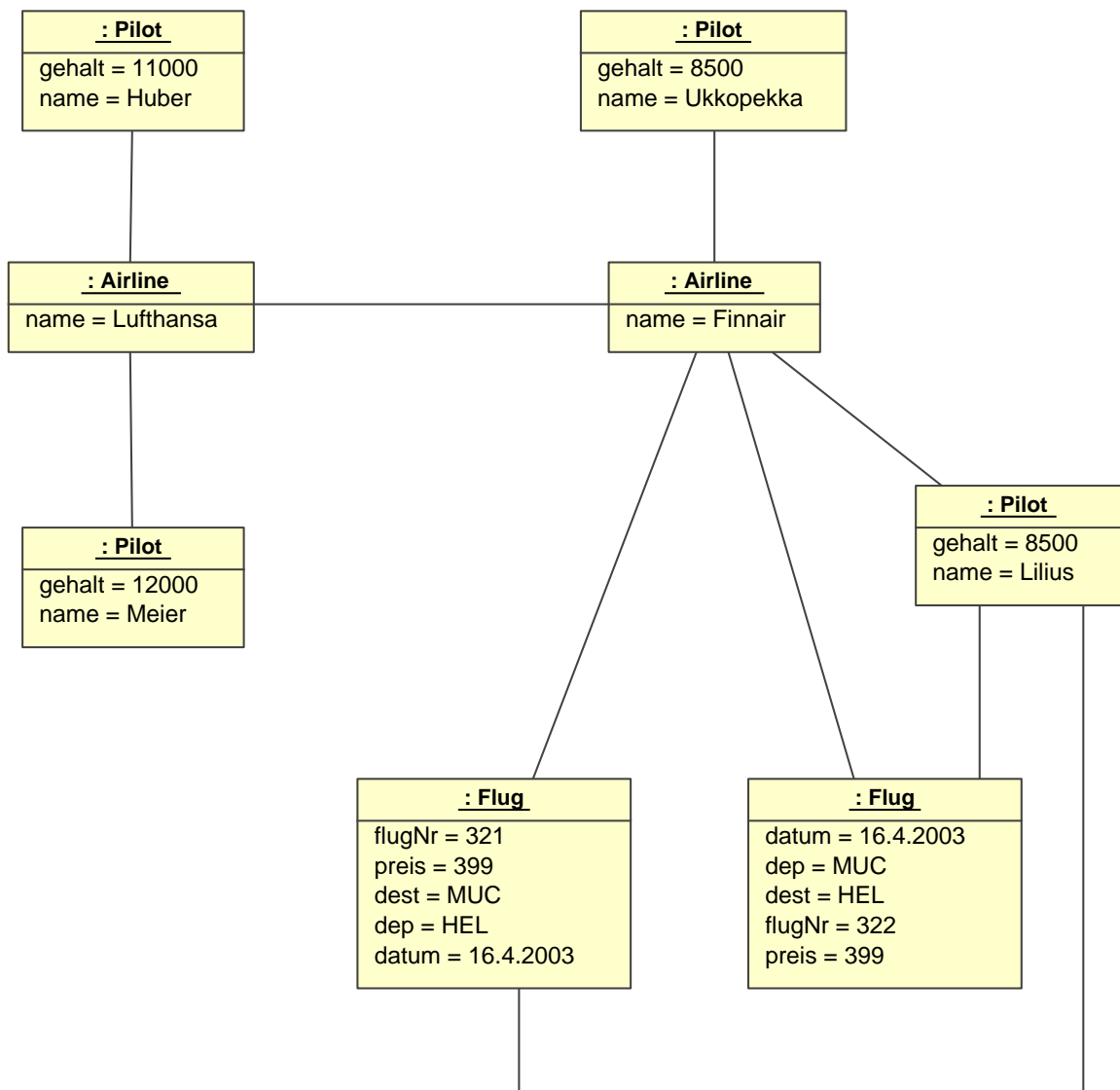


Übungen zu Methoden des Software-Engineering: Lösungsvorschläge
(Florian Hacklinger, Dr. N. Koch, Prof. Dr. M. Wirsing)

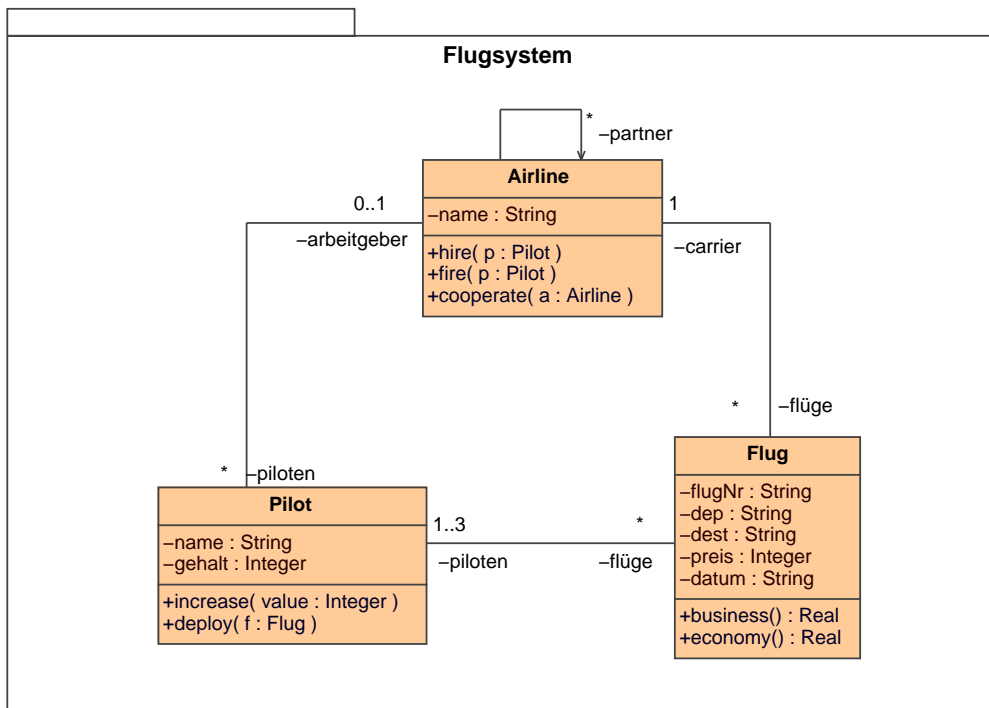
Aufgabe 1

Signaturen und OCL

a) Objektdiagramm:



b) Klassendiagramm:



Nicht ausdrückbar sind: Generelle Einschränkungen an die Zustände einzelner Objekte (z. B. das Gehalt eines Piloten ist nicht negativ), sowie Einschränkungen an Objektkonfigurationen (z. B. Airlines kooperieren nicht mit sich selbst, Piloten fliegen nur Flüge der eigenen Airline).

c) Klasseninvarianten

gehaltCheck ist eine Hilfsoperation, die überprüft, ob für eine Menge von Piloten gilt: Das Gehalt des Piloten weicht nicht mehr als 10% vom durchschnittlichen Gehalt der Piloten ab.

```

context Airline
def: gehaltsCheck(gehälter : Set(Integer)) : Boolean =
  if (gehälter->size() = 0)
  then
    true
  else
    let mw : Real =
      gehälter->sum() / gehälter->size()
    in
      gehälter->forall(g |
        abs(mw - g) / g <= 0.1 )
    end
  endif
  
```

```

context Airline
inv: Airline.allInstances()->forall(a1, a2 |
  
```

```
(a1.name = a2.name) implies (a1 = a2))
```

```
context Flug
```

```
inv: preis > 0
```

```
context Pilot
```

```
inv: self.gehalt >= 0
```

```
inv: self.airline.gehaltsCheck(self.airline.piloten.gehalt)
```

Zusätzlich: Airlines kooperieren nicht mit sich selbst:

```
context Airline
```

```
inv: Airline.allInstances()->forall(a |  
    a.partners->excludes(a))
```

Oder alternativ:

```
self.partners->excludes(self)
```

d) Operationsspezifikationen

```
context Airline::hire(p : Pilot)
```

```
pre: Airline.allInstances().piloten->flatten().excludes(p)  
    and gehaltCheck(self.piloten->including(p.gehalt))
```

```
post: piloten = piloten@pre->including(p)
```

```
context Airline::fire(p : Pilot)
```

```
pre: piloten->includes(p)  
    and gehaltCheck(piloten->excluding(p).gehalt)
```

```
post: piloten = piloten@pre->excluding(p)
```

```
context Pilot::increase(a : Integer)
```

```
pre: a > 0 and airline.gehaltCheck(self.airline.  
    piloten.gehalt->including(a))
```

```
post: gehalt = gehalt@pre + a
```

```
context Pilot::deploy(f : Flug)
```

```
pre: self.flüge->forall(f1 |  
    ((f1.datum = f.datum) and  
    (f <> f1)) implies  
    ((f1.dep = f.dest) and  
    (f.dep = f1.dest)) and  
    f.piloten->size() < 3 and  
    and f.piloten->excludes(self)  
    and f.carrier = self.airline
```

```
post: flüge = flüge@pre->including(f)
```

```

context Flug::economy() : Real
pre: true
post: result = preis

```

```

context Flug::business() : Real
pre: true
post: result = 2.5 * preis

```

```

context Airline::cooperate(Airline a)
pre: a <> self
post: self.partner = self.partner@pre->including(a)

```

e) Implizite Komponenteninvarianten (gegeben durch die Assoziationen):

```

context FlugSystem
inv: Airline.allInstances()->forall(a|
    a.partner.isOclTypeOf(Set(Airline)))

inv: Airline.allInstances()->forall(a|
    a.piloten->forall(p|
        p.arbeitgeber = a))
and
Pilot.allInstances()->forall(p|
    p.arbeitgeber <> null implies
        p.arbeitgeber.piloten->includes(p))

inv: Airline.allInstances()->forall(a|
    a.flüge.forAll(f|f.carrier = a)
and Flug.allInstances()->forall(f|
    f.carrier.flüge->includes(f))

inv: Pilot.allInstances()->forall(p|
    p.flüge->forall(f|f.piloten->includes(p)))
and
Flug.allInstances()->forall(f|
    f.piloten->size() <= 3 and f.piloten.size() > 0
    and f.piloten->forall(p|p.flüge->includes(f)))

```

Explizite Komponenteninvarianten:

```

context FlugSystem
inv: Airline.allInstances()->forall(a|
    a.flüge->includesAll(a.piloten.flüge->flatten())) and
Airline.allInstances()->forall(a|
    a.piloten->includesAll(a.flüge.piloten->flatten()))

```