SE463
Software Requirements Specification & Analysis

Business Rules, OCL

Business Rules

A business rule is an assertion that defines or constrains some aspect of the Work.

Examples:

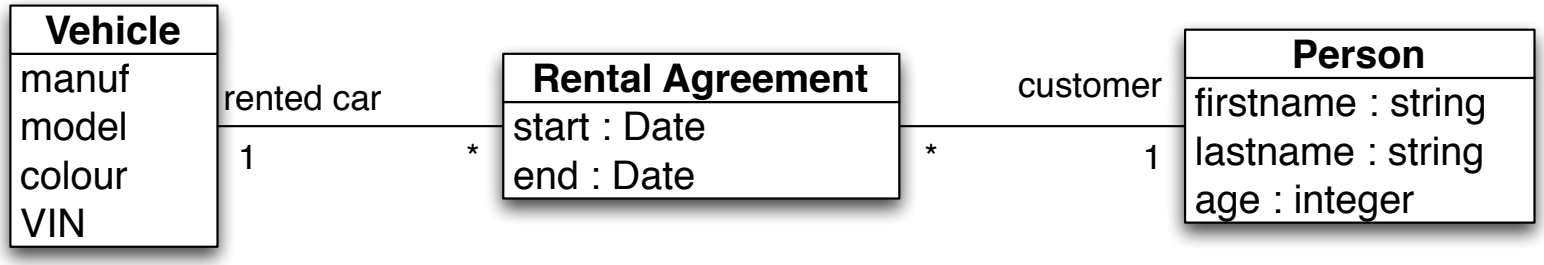
- Rental agreements must be no longer than 4 weeks long
- A customer must be at least 25 years old
- A customer cannot rent more than 3 red cars

Object Constraint Language

- Complements the UML
  - standardized by the Object Management Group (OMG)
  - not one of the UML notations
  - used to express constraints on UML models
  - precise, yet (relatively) easy to read

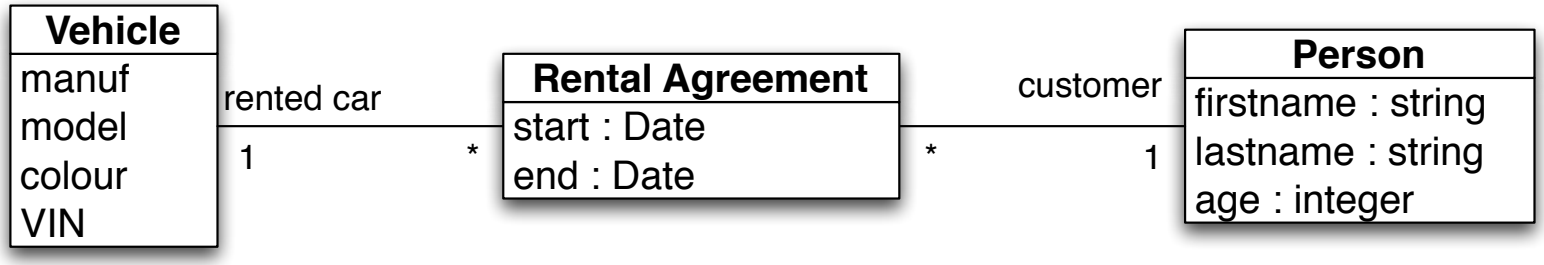
- It has language constructs for
  - relating classes that have no direct association
  - expressing queries over objects and collections of objects

Simple OCL example



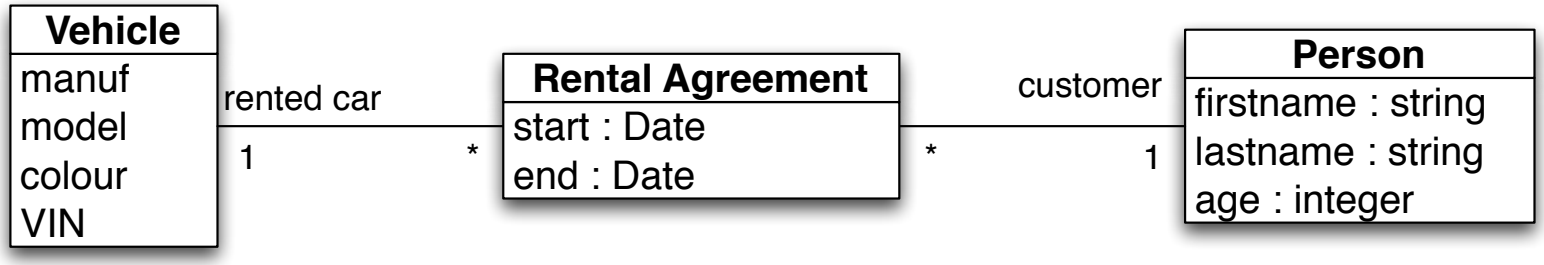
Rental agreements must be less than 4 weeks long

Another example



A customer must be at least 25 years old

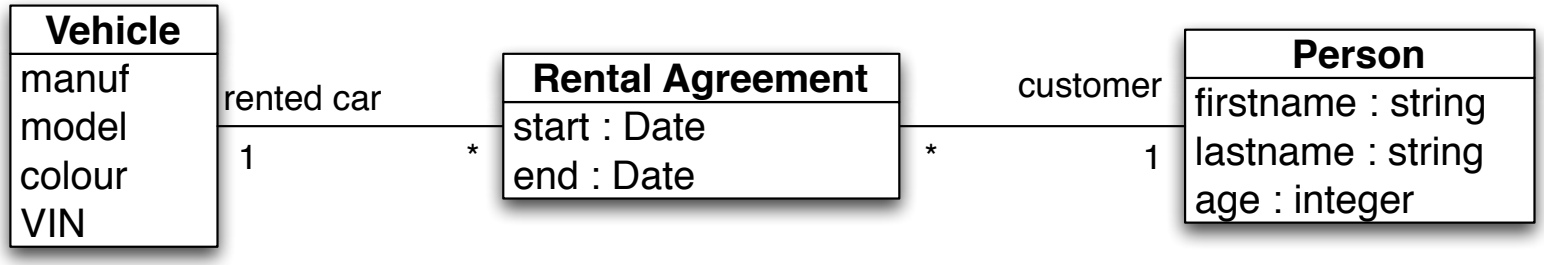
Constraint Expressions



Expressions over

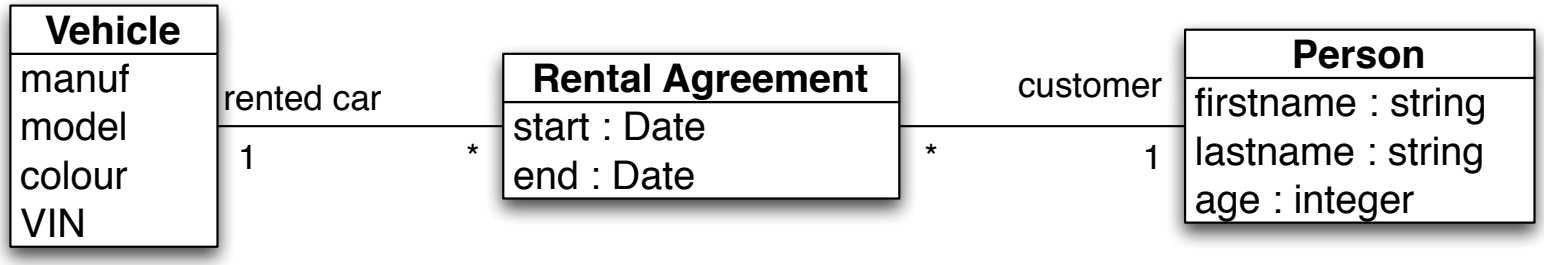
- attributes
- navigations derived from associations (and aggregations)
- rolename on far end of association
- class name on far end of association
- literal values

A third example



A customer cannot rent more than 3 cars

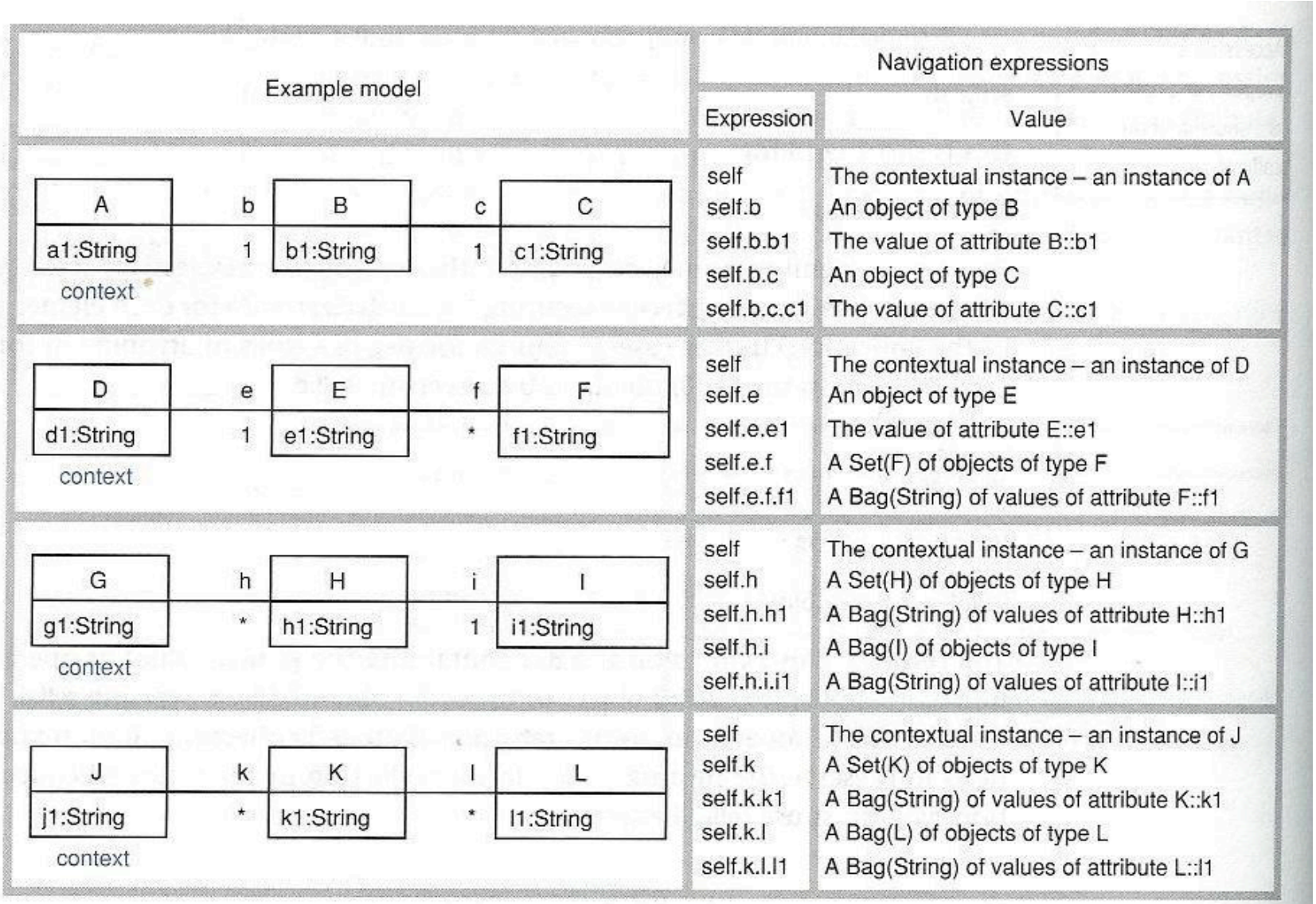
Navigation Across Associations



Consider object p:Person:

Expression	Value
p	
p.RentalAgreement	
p.RentalAgreement.rented_car	
p.RentalAgreement.rented_car.colour	

Navigation Across Associations



Example model	Navigation expressions	
	Expression	Value
A — b — B — c — C	self	The contextual instance – an instance of A
	self.b	An object of type B
a1:String — 1 — b1:String — 1 — c1:String	self.b.b1	The value of attribute B::b1
context	self.b.c	An object of type C
	self.b.c.c1	The value of attribute C::c1
D — e — E — f — F	self	The contextual instance – an instance of D
	self.e	An object of type E
d1:String — 1 — e1:String — * — f1:String	self.e.e1	The value of attribute E::e1
context	self.e.f	A Set(F) of objects of type F
	self.e.f.f1	A Bag(String) of values of attribute F::f1
G — h — H — i — I	self	The contextual instance – an instance of G
	self.h	A Set(H) of objects of type H
g1:String — * — h1:String — 1 — i1:String	self.h.h1	A Bag(String) of values of attribute H::h1
context	self.h.i	A Bag(I) of objects of type I
	self.h.i.i1	A Bag(String) of values of attribute I::i1
J — k — K — l — L	self	The contextual instance – an instance of J
	self.k	A Set(K) of objects of type K
j1:String — * — k1:String — * — l1:String	self.k.k1	A Bag(String) of values of attribute K::k1
context	self.k.l	A Bag(L) of objects of type L
	self.k.l.l1	A Bag(String) of values of attribute L::l1

Figure 25.12

Arlow and Neustadt, *UML 2 and the Unified Process*

Operations on Collections

OCL operations on collections are prefaces with an arrow notation (->), to distinguish from model-defined names or operations

// size() refers to (nonexistent) method ”size" in class Vehicle

context Person **inv:**

self.RentalAgreement.Vehicle.size() <= 3

// size() is OCL operation that counts number of vehicles

context Person **inv:**

self.RentalAgreement.Vehicle->size() <= 3

Basic types

Booleans		
Operation	Notation	Result Type
or	a or b	Boolean
and	a and b	Boolean
exclusive or	a xor b	Boolean
negation	not a	Boolean
equals	a = b	Boolean
not equals	a <> b	Boolean
implies	a implies b	Boolean
if then else	if a then b else b'	Type of b and b'

Strings		
Operation	Notation	Result Type
concatenation	s.concat(t)	String
size	s.size	Integer
to lower case	s.toLower	String
to upper case	s.toUpper	String
substring	s.substring(int,int)	String
equals	s=t	Boolean
not equals	s<>t	Boolean

Integers and Reals		
Operation	Notation	Result Type
equals	a = b	Boolean
not equals	a <> b	Boolean
less	a < b	Boolean
more	a > b	Boolean
less or equal	a <= b	Boolean
more or equal	a >= b	Boolean
plus	a + b	Integer or Real
minus	a - b	Integer or Real
multiplication	a * b	Integer or Real
division	a / b	Real
modulus	a.mod(b)	Integer
integer division	a.div(b)	Integer
absolute value	a.abs	Integer or Real
maximum value	a.max(b)	Integer or Real
minimum value	a.min(b)	Integer or Real
round	a.round	Integer
floor	a.floor	Integer

Arlow and Neustadt, *UML 2 and the Unified Process*

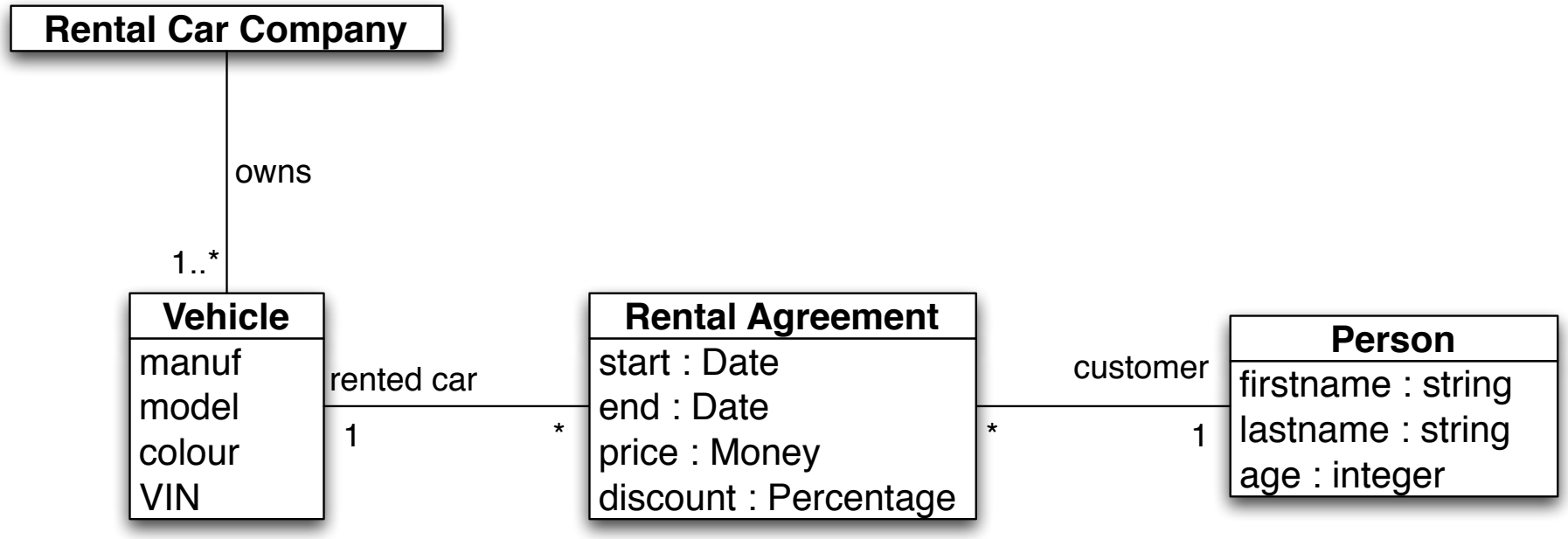
Expressions on collections

Set of T		
Operation	Notation	Result Type
equals	a = b	Boolean
not equals	a <> b	Boolean
size	a–>size()	Integer
sum	a–>sum()	Type T
count	a–>count(t)	Integer
includes	a–>includes(t)	Boolean
excludes	a–>excludes(t)	Boolean
includes all	a–>includesall(b)	Boolean
excludes all	a–>excludesall(b)	Boolean
is empty	a–>isEmpty()	Boolean
not empty	a–>notEmpty()	Boolean
union	a–>union(b)	Set of T
intersection	a–>intersection(b)	Set of T
difference	a - b	Set of T
insert	a–>including(t)	Set of T
remove	a–>excluding(t)	Set of T

a, b : Set of Type T
t : object of Type T

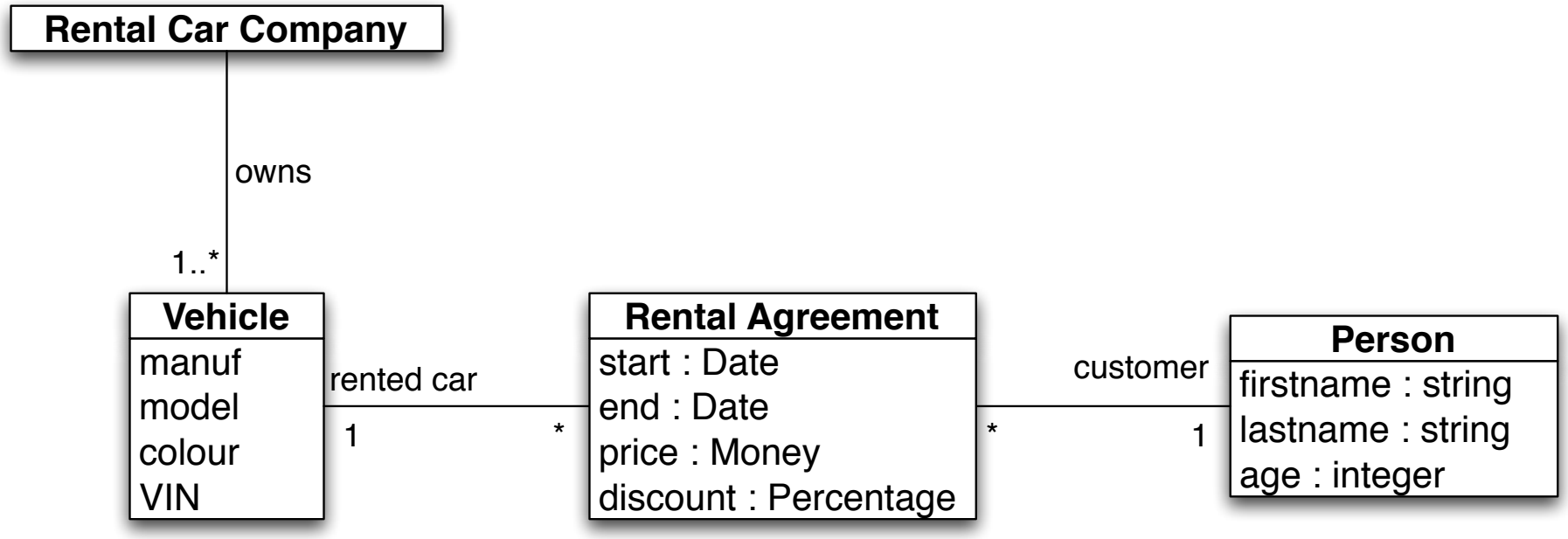
Arlow and Neustadt, *UML 2 and the Unified Process*

More Examples



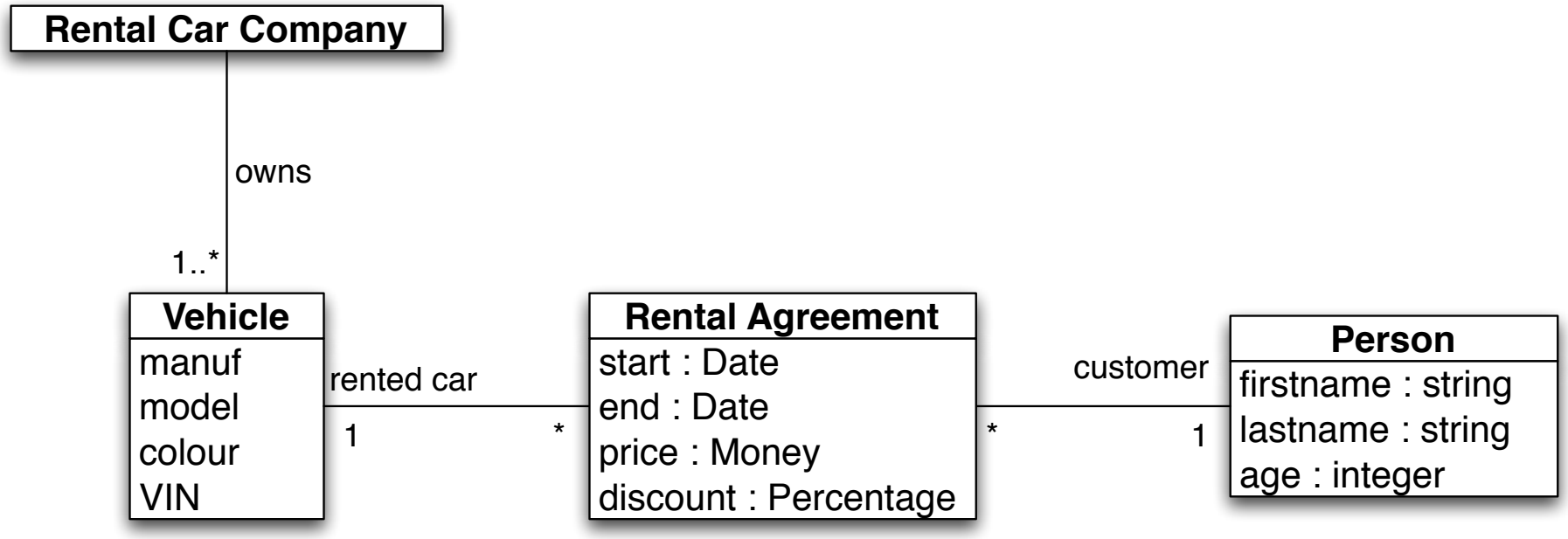
Rental car companies never own red cars

More Examples



Frequent renters (# rentals > 20) receive 10% discount

More Examples



Heavy spenders (sum of rentals > $5000) receive 20% discount

Filtering Operations

There are operations extracting specific elements from an existing collection, based on the value of an expression.

- select returns the elements that satisfy given expression
- reject returns the elements that falsify given expression

Rental car companies never own red cars

context RentalCarCompany **inv:**
self.owns->select(colour="red")->isEmpty()

Customers younger than 30 cannot rent red cars

Quantification

The exists operation is used to assert that at least one element in a collection satisfies some expression. The operation returns a boolean value.

Every customer rents at least one black car
context Person **inv:**
self.RentalAgreement.Vehicle->exists(colour="black")

Every car is being rented today
(i.e., every car is involved in some rental agreement today)
context Vehicle **inv:**
self.RentalAgreement->exists(r : RentalAgreement |
    r.start <= today and today <= r.end)

Quantification

The forAll operation is used to assert that all members of a set satisfy a given expression. The operation returns a boolean.

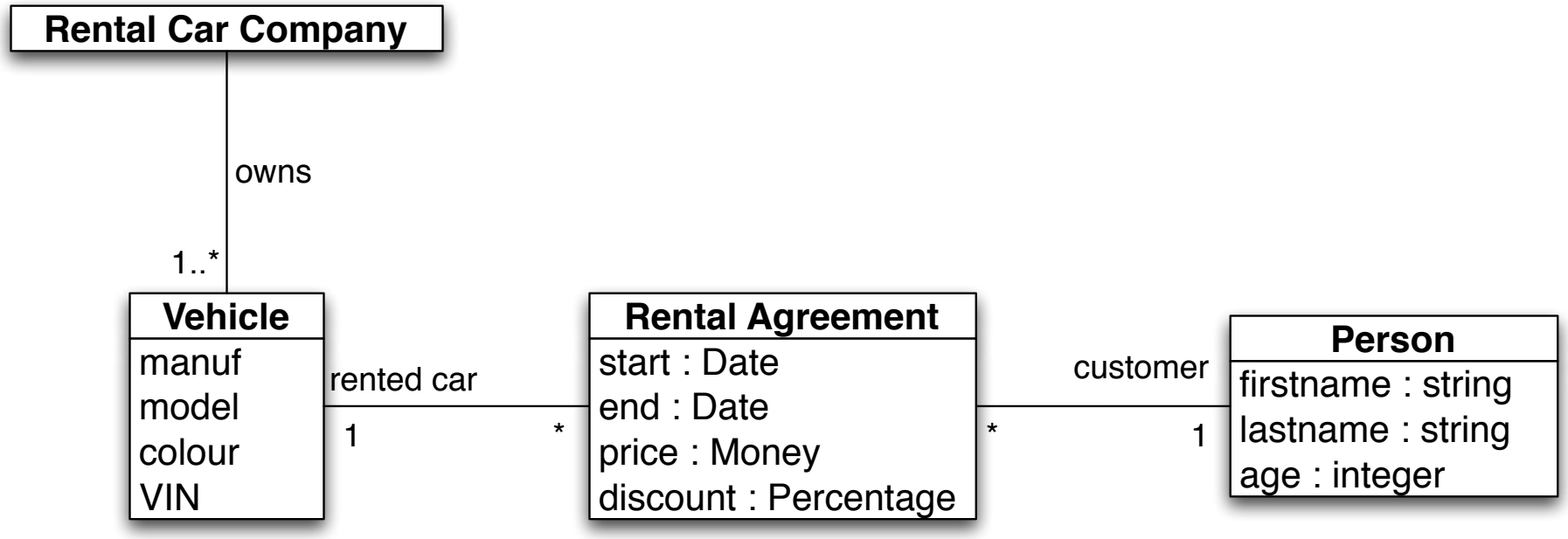
All rental cars are white

context RentalCarCompany **inv:**
self.owns->forall(colour="white")

No car is rented more than once each day

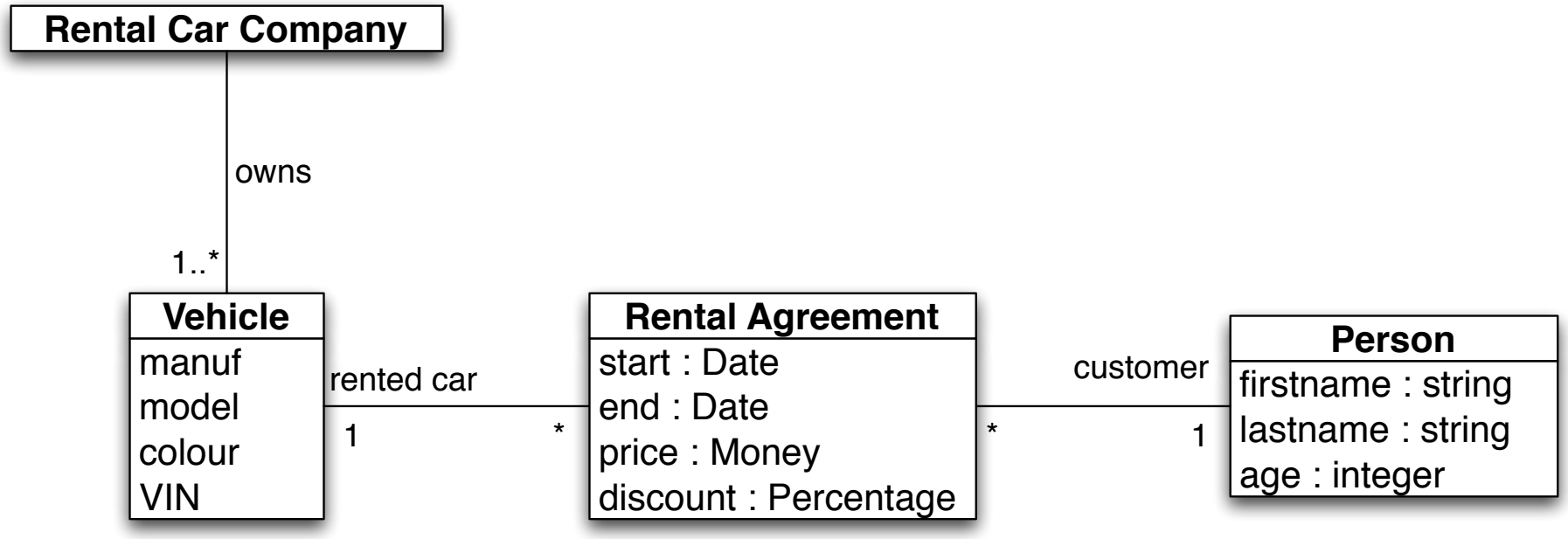
context Vehicle **inv:**
self.RentalAgreement->forall(r1, r2 : RentalAgreement |
    (r1 <> r2) implies (r1.start > r2.end or   r2.start > r1.end))

More Examples



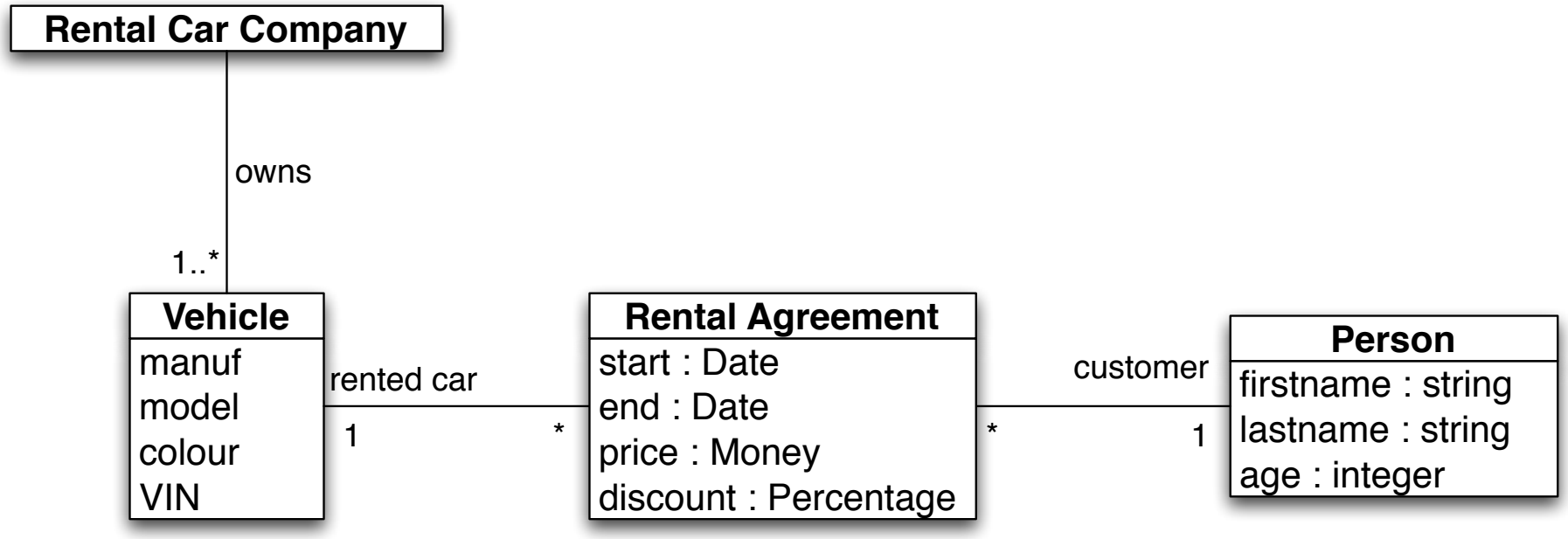
No rental agreement has a price of less than $20

More Examples



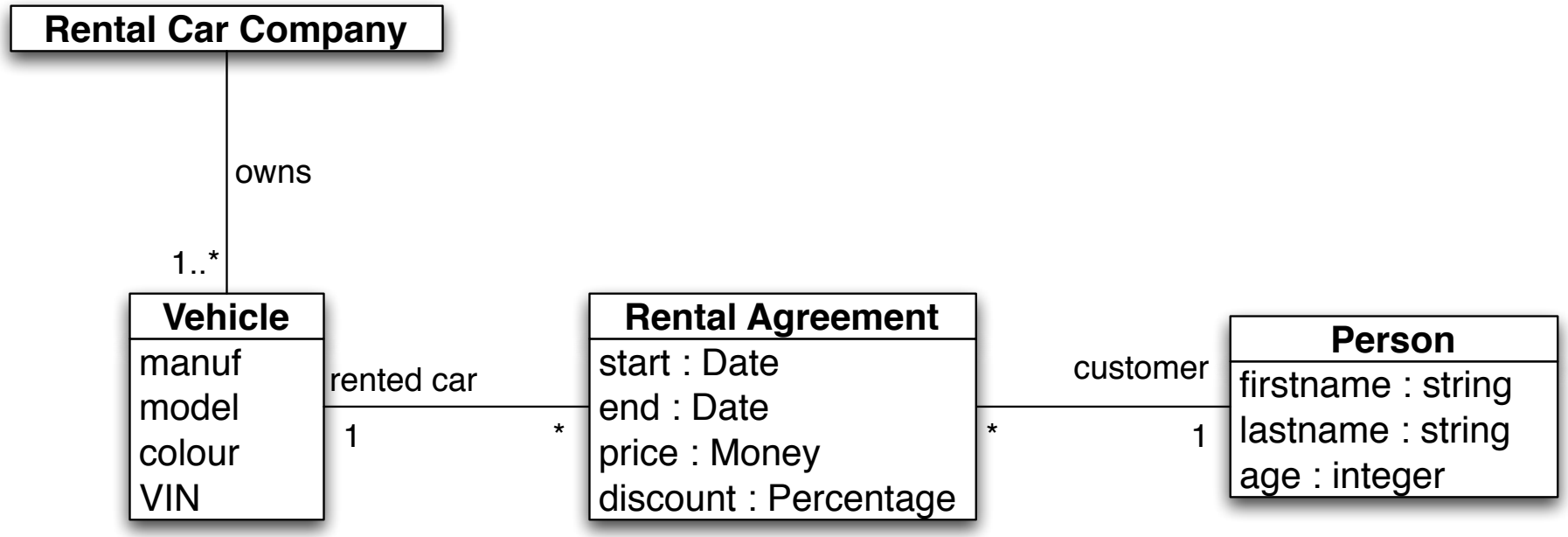
The rental car fleet contains at least one Toyota

More Examples



Red cars are never rented for less $100/day

More Examples



No person rents the same car twice

Broken Rules

Note that business rules simply state what ought to be true. If the execution of the system leads to a case where the rule is not true, we say that the rule is broken or violated.

There is nothing in the description of a business rule that says how to recover from a broken rule.

OCL Tools

There are a number of tools that support OCL, from both universities and industry. These tools range from

- Parsers and type checkers
- Evaluators that can check an OCL expression against all instances of a UML class model
- Debuggers that step through an OCL expression and check each subsection (to locate faulty subexpression)
- Code generators that translate OCL expressions into run-time assertions

Summary

Object Constraint Language (OCL)

- augments UML
- expresses business rules as constraints on the domain model