Introduction to OCL

Bernhard Beckert

UNIVERSITÄT KOBLENZ-LANDAU

Object Constraint Language

- **Part of the UML standard.**

Object Constraint Language

- Part of the UML standard.
- Formal Specification Language. Precise semantics.

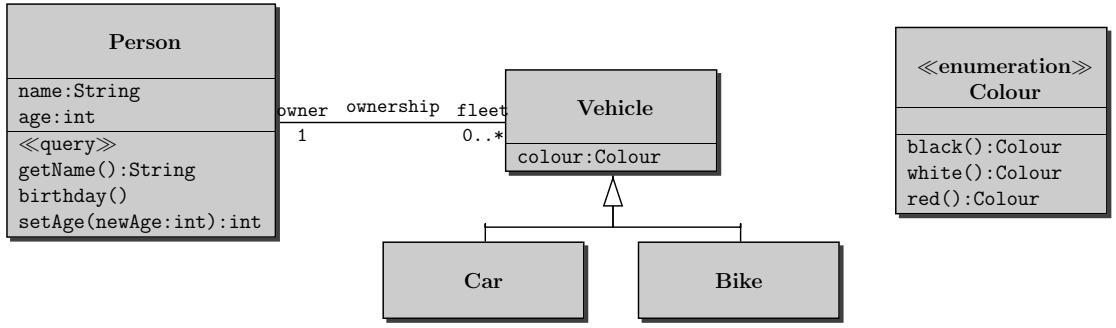
Object Constraint Language

- Part of the UML standard.
- Formal Specification Language. Precise semantics.
- (Quite) easy to read syntax.

Object Constraint Language

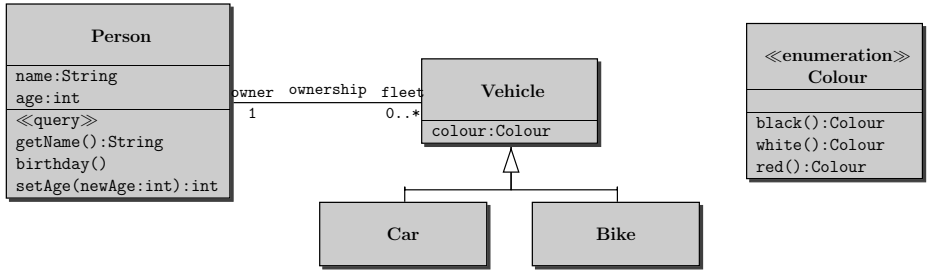
- Part of the UML standard.
- Formal Specification Language. Precise semantics.
- (Quite) easy to read syntax.
- Why? Because UML is not enough!

UML is not enough. . .



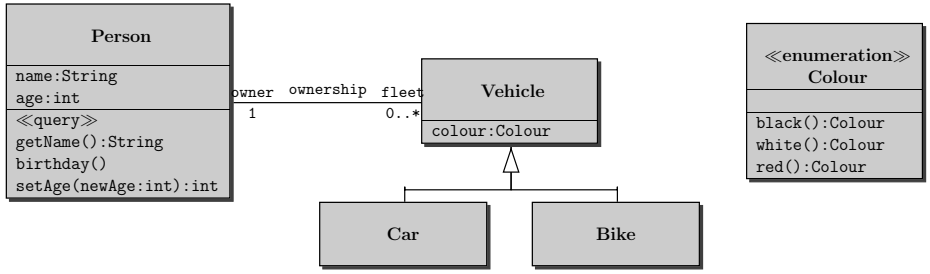
- **Possible number of owners a car can have**
- **Required age of car owners**
- **Requirement that a person may own at most one black car**

Some OCL examples I



“A vehicle owner must be at least 18 years old”:

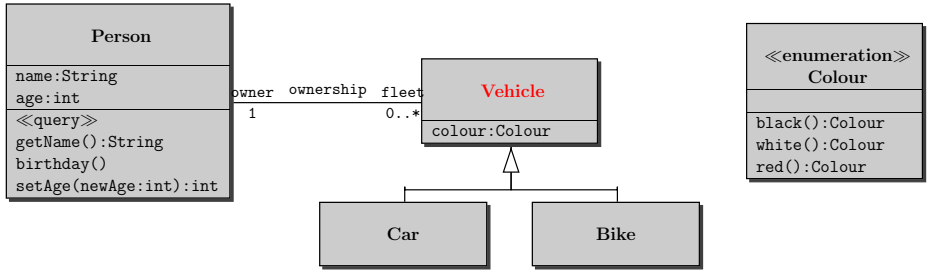
Some OCL examples I



"A vehicle owner must be at least 18 years old":

context Vehicle
inv: self. owner. age $>=$ 18

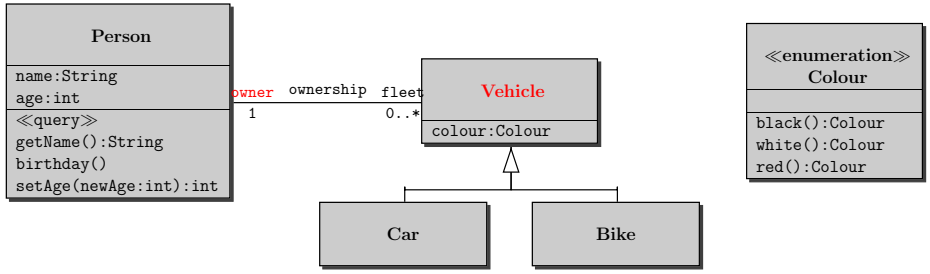
Some OCL examples I



"A vehicle owner must be at least 18 years old":

context **Vehicle**
inv: **self. owner. age** $>=$ **18**

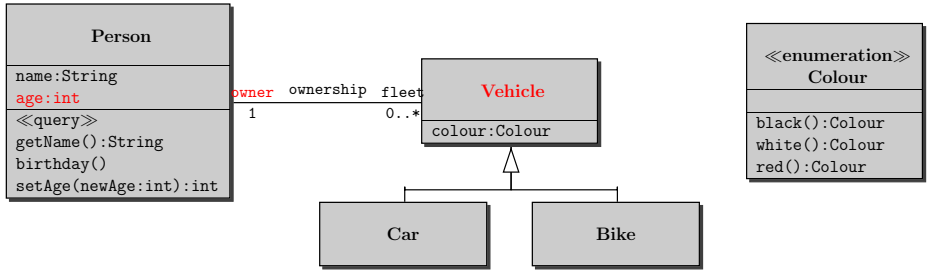
Some OCL examples I



"A vehicle owner must be at least 18 years old":

context Vehicle
inv: self. owner. age $>=$ 18

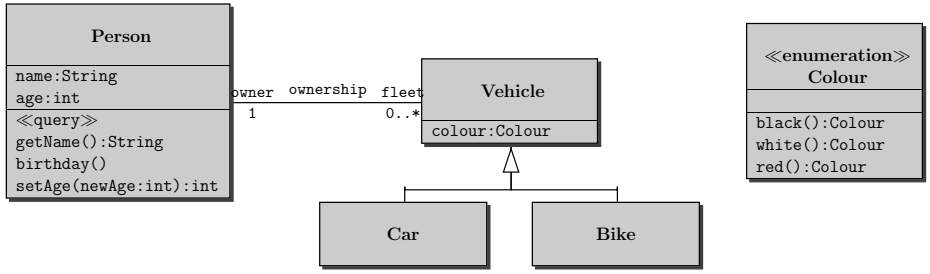
Some OCL examples I



"A vehicle owner must be at least 18 years old":

context Vehicle
inv: self. owner. age $>=$ 18

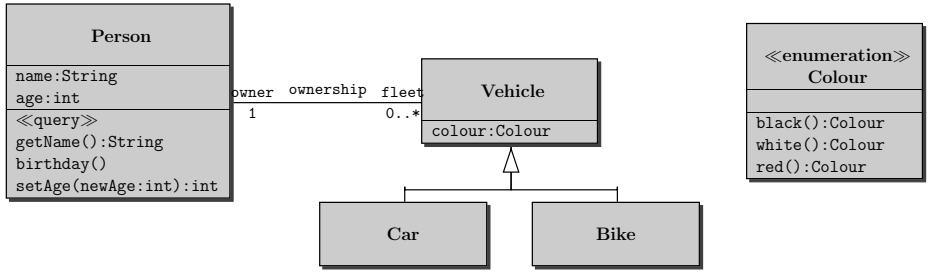
Some OCL examples I



"A vehicle owner must be at least 18 years old":

context Vehicle
inv: self. owner. age $>=$ 18

Some OCL examples I



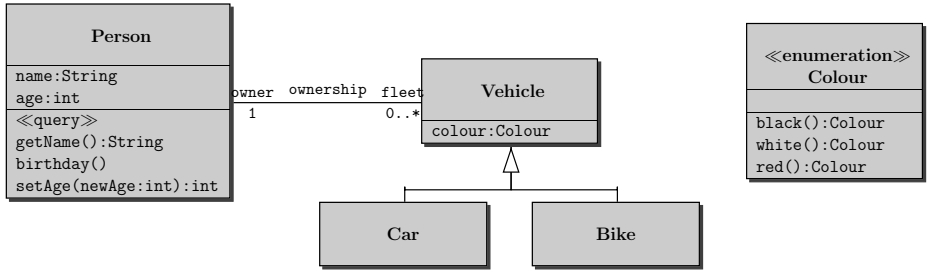
"A vehicle owner must be at least 18 years old":

context Vehicle
inv: self. owner. age $>=$ 18

What does this mean, instead?

context Person
inv: self.age $>=$ 18

Some OCL examples I



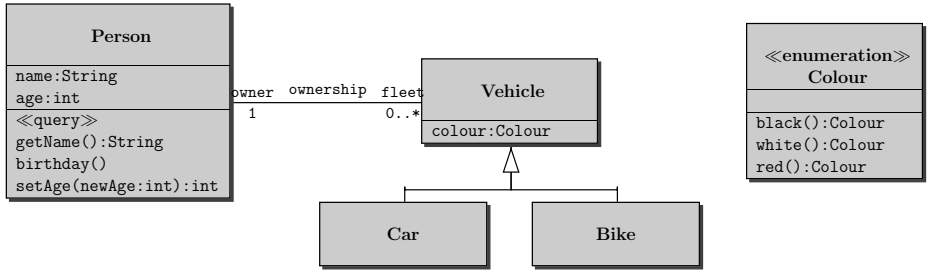
"A vehicle owner must be at least 18 years old":

context Vehicle
inv: self. owner. age $>=$ 18

"A car owner must be at least 18 years old":

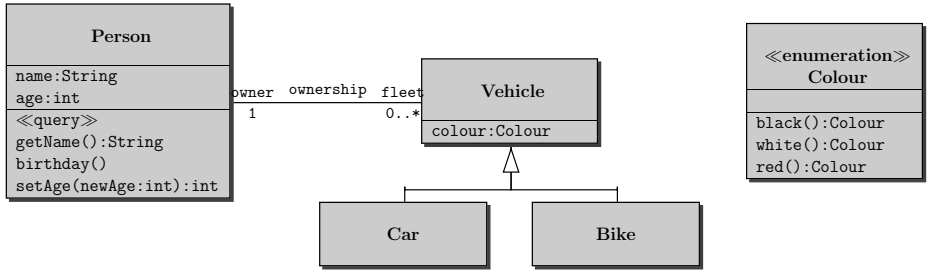
context Car
inv: self.owner.age $>=$ 18

Some OCL examples II



“Nobody has more than 3 vehicles”:

Some OCL examples II

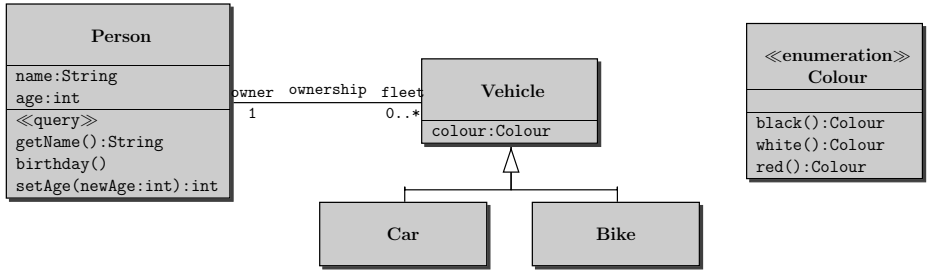


“Nobody has more than 3 vehicles”:

```
context  Person
inv:     self.fleet->size <= 3
```

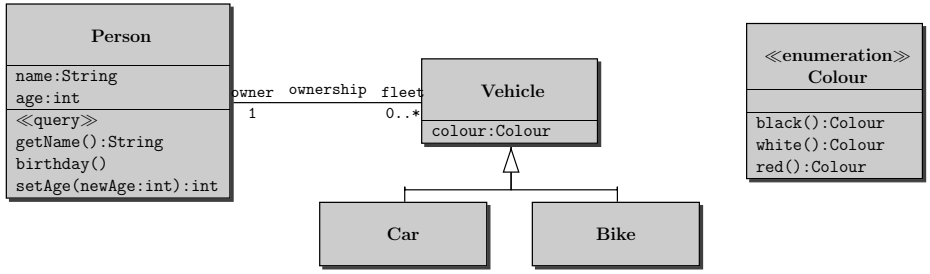
or change multiplicity

Some OCL examples II



“All cars of a person are black”:

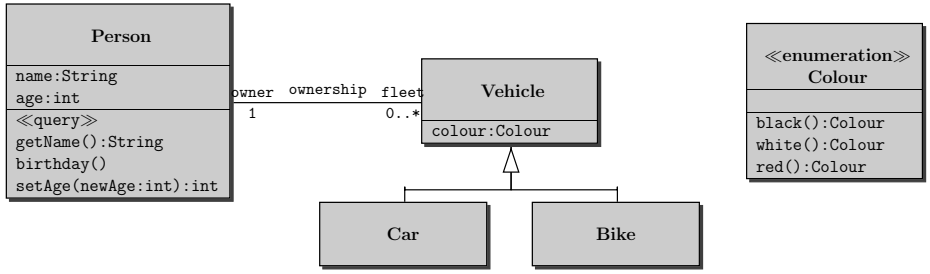
Some OCL examples II



“All cars of a person are black”:

```
context   Person
inv:      self.fleet->forAll(v | v.colour = #black)
```

Some OCL examples II

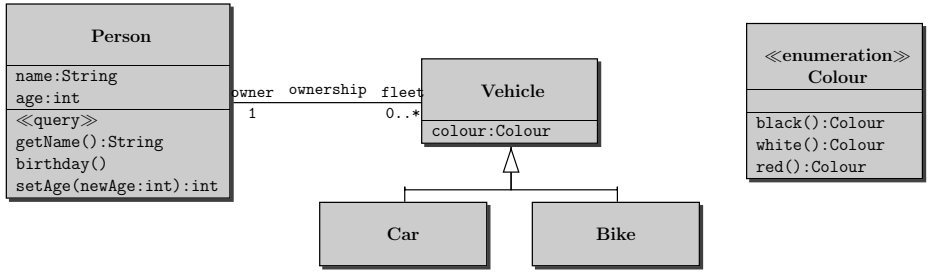


"All cars of a person are black":

```
context   Person
inv:      self.fleet->forAll(v | v.colour = #black)
```

"Nobody has more than 3 black vehicles":

Some OCL examples II



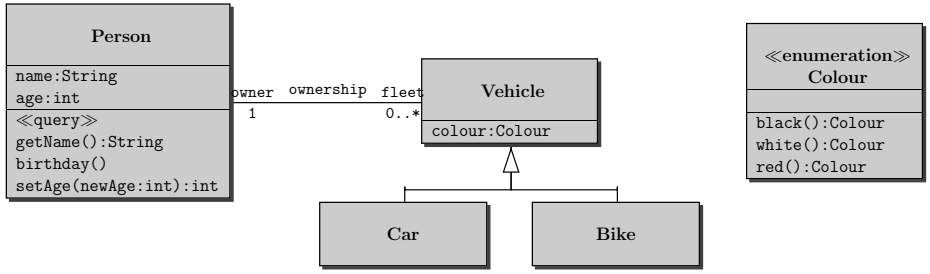
"All cars of a person are black":

```
context   Person
inv:      self.fleet->forAll(v | v.colour = #black)
```

"Nobody has more than 3 black vehicles":

```
context   Person
inv:      self.fleet->select(v | v.colour = #black)->size <= 3
```

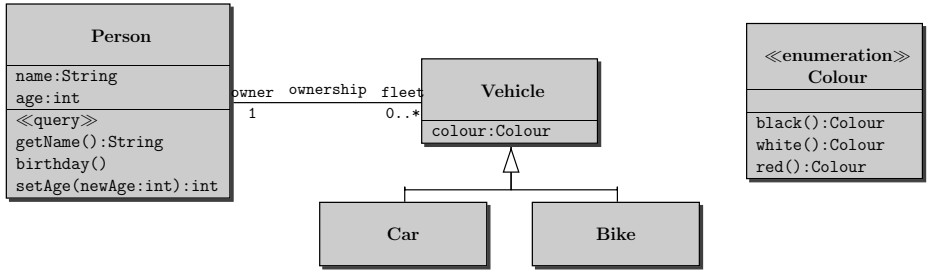
Some OCL examples III — iterate



What does this mean?

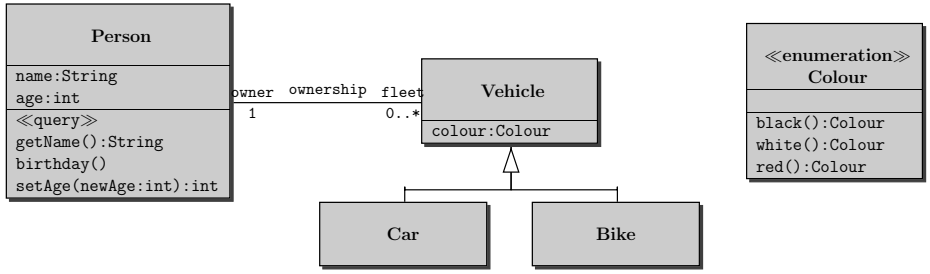
```
context   Person
inv:      self.fleet->iterate(v; acc:Integer=0
                              | if (v.colour=#black)
                              then acc + 1 else acc endif) <=3
```

Some OCL examples IV — oclIsKindOf



context Person
inv: age<18 implies self.fleet->forAll(v | not v.oclIsKindOf(Car))

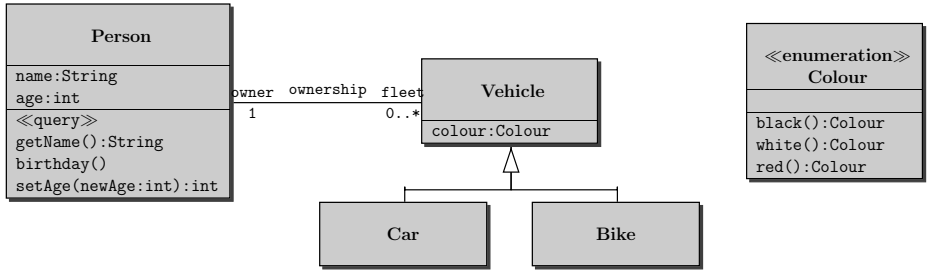
Some OCL examples IV — oclIsKindOf



context Person
inv: age<18 implies self.fleet–>forAll(v | not v.oclIsKindOf(Car))

“A person younger than 18 owns no cars.”

Some OCL examples IV — oclIsKindOf

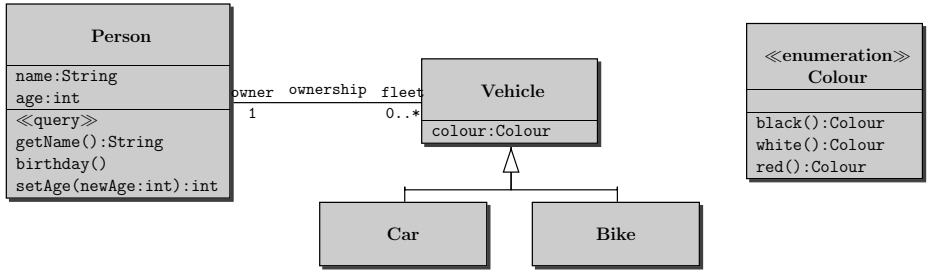


context Person
inv: age<18 implies self.fleet–>forAll(v | not v.oclIsKindOf(Car))

“A person younger than 18 owns no cars.”

“self” can be omitted.

Some OCL examples IV — oclIsKindOf



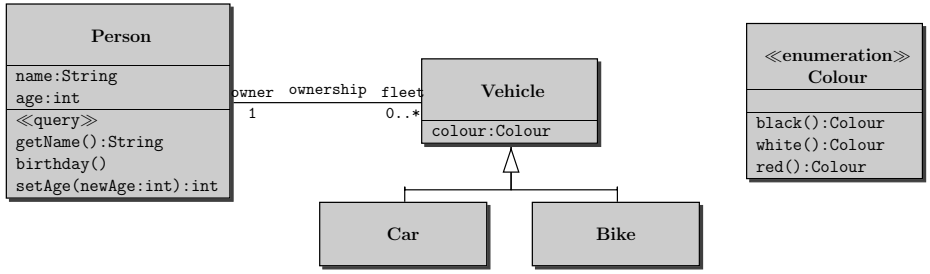
context Person
inv: age<18 implies self.fleet–>forAll(v | not v.oclIsKindOf(Car))

“A person younger than 18 owns no cars.”

“self” can be omitted.

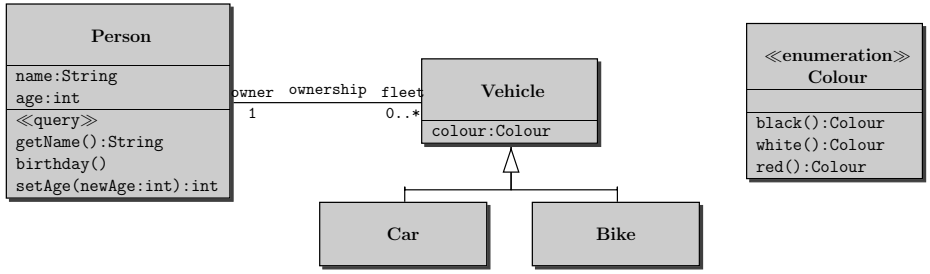
Logical Junctors: and, or, not, implies, if. . . then. . . else. . . endif, =

Some OCL examples V — allInstances



context Car
inv: Car.allInstances()->exists(c | c.colour=#red)

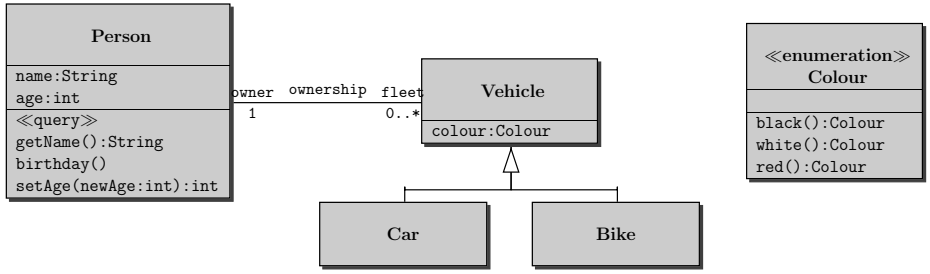
Some OCL examples V — allInstances



context Car
inv: Car.allInstances()->exists(c | c.colour=#red)

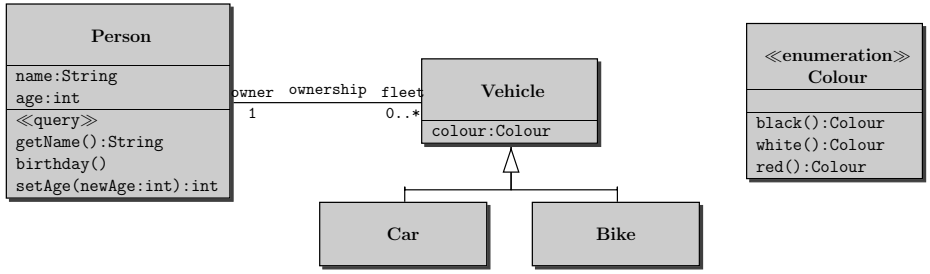
“There is a red car.”

OCL pre-/post conditions — Examples



So far only considered class invariants.

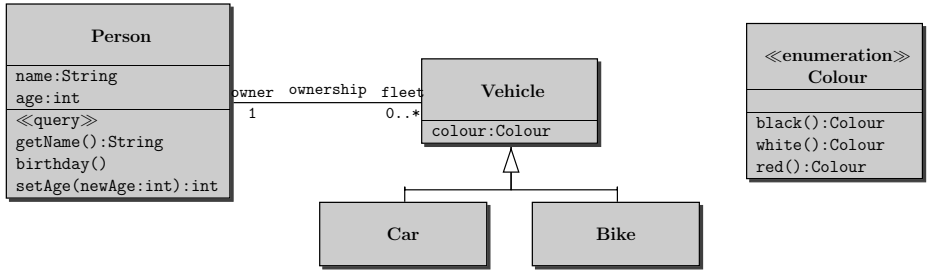
OCL pre-/post conditions — Examples



So far only considered class invariants.

OCL can also specify operations:

OCL pre-/post conditions — Examples



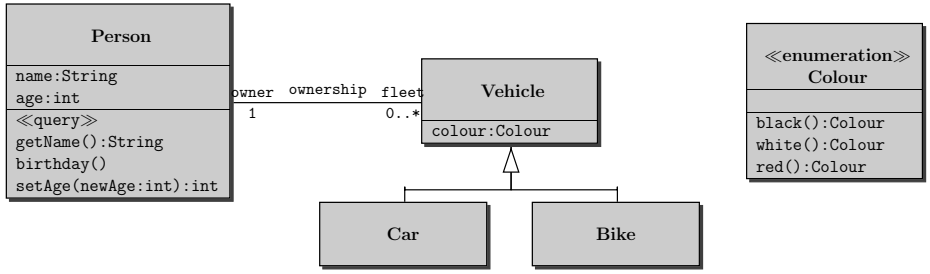
So far only considered class invariants.

OCL can also specify operations:

"If setAge(. . .) is called with a non-negative argument then the argument becomes the new value of the attribute age."

context Person::setAge(newAge:int)
pre: newAge >= 0
post: self.age = newAge

OCL pre-/post conditions — Examples



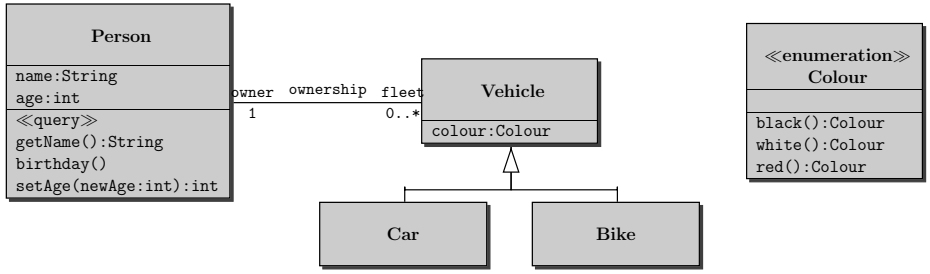
So far only considered class invariants.

OCL can also specify operations:

“Calling birthday() increments the age of a person by 1.”

context Person::birthday()
post: self.age = self.age@pre + 1

OCL pre-/post conditions — Examples



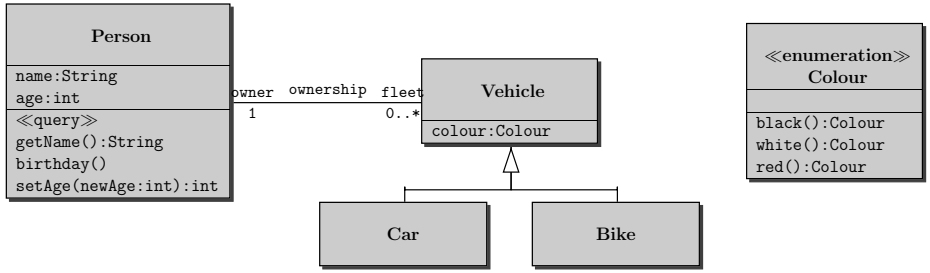
So far only considered class invariants.

OCL can also specify operations:

“Calling getName() delivers the value of the attribute name.”

context Person::getName()
post: result = name

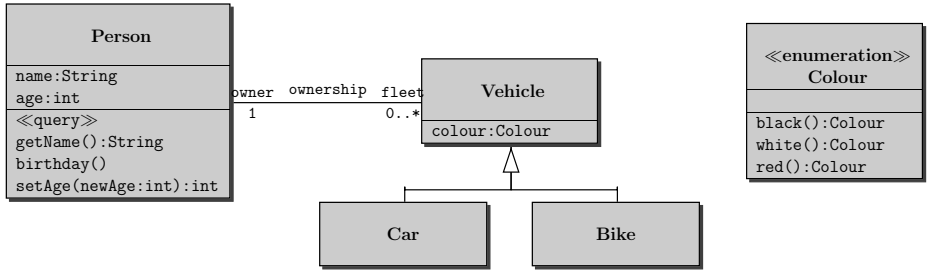
Queries



Special to OCL are operations with a ≪query≫ stereotype:

Only these operations can be used within an OCL expression.

Queries



Special to OCL are operations with a ≪query≫ stereotype:

Only these operations can be used within an OCL expression.

“Calling getName() delivers the value of the attribute name.”

```
context   Person
inv:      self.getName() = name
```

OCL Basics

- **OCL is used to specify invariants of objects and pre- and post conditions of operations. Makes UML (class) diagrams more precise.**

OCL Basics

- **OCL is used to specify invariants of objects and pre- and post conditions of operations. Makes UML (class) diagrams more precise.**
- **OCL expressions use vocabulary of UML class diagram.**

OCL Basics

- OCL is used to specify invariants of objects and pre- and post conditions of operations. Makes UML (class) diagrams more precise.
- OCL expressions use vocabulary of UML class diagram.
- OCL attribute accesses “navigate” through UML class diagram.

OCL Basics

- OCL is used to specify invariants of objects and pre- and post conditions of operations. Makes UML (class) diagrams more precise.
- OCL expressions use vocabulary of UML class diagram.
- OCL attribute accesses “navigate” through UML class diagram.
- “context” specifies about which elements we are talking.

OCL Basics

- OCL is used to specify invariants of objects and pre- and post conditions of operations. Makes UML (class) diagrams more precise.
- OCL expressions use vocabulary of UML class diagram.
- OCL attribute accesses “navigate” through UML class diagram.
- “context” specifies about which elements we are talking.
- “self” indicates the current object. “result” the return value.

OCL Basics (cont.)

- **OCL can talk about collections (here: sets).**

  Operations on collections: –>

  Example operations: select, forAll, iterate

OCL Basics (cont.)

- **OCL can talk about collections (here: sets).**

  Operations on collections: –>

  Example operations: select, forAll, iterate

- **“iterate” can simulate all other operations on collections.**

OCL Basics (cont.)

- **OCL can talk about collections (here: sets).**

  Operations on collections: –>

  Example operations: select, forAll, iterate

- **“iterate” can simulate all other operations on collections.**

- **Queries (= side-effect-free operations) can be used in OCL expressions.**

OCL in TogetherCC/KeY

TogetherCC cannot process OCL constraints. It is however possible to specify textual invariants and pre- and post conditions.

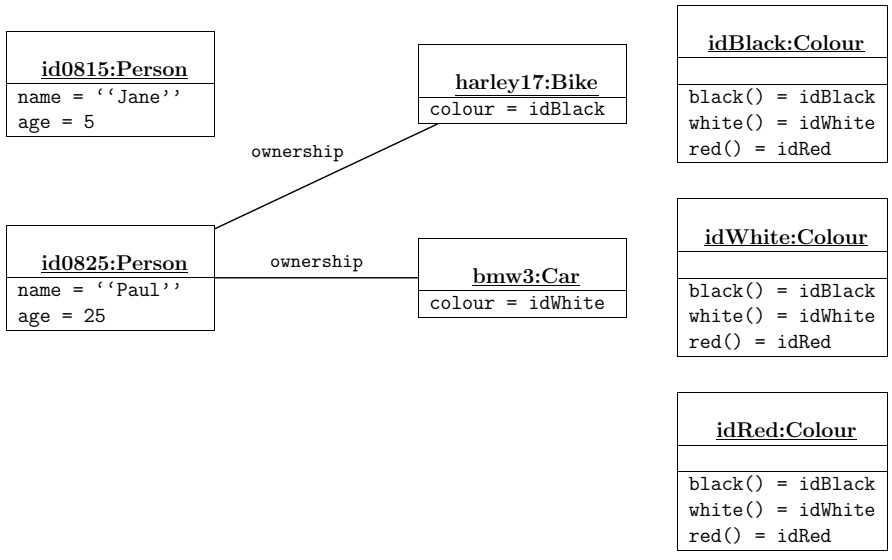
With the KeY extensions to TogetherCC syntax (type) checks of OCL constraints are possible.

System state

(represented by a UML object diagram)

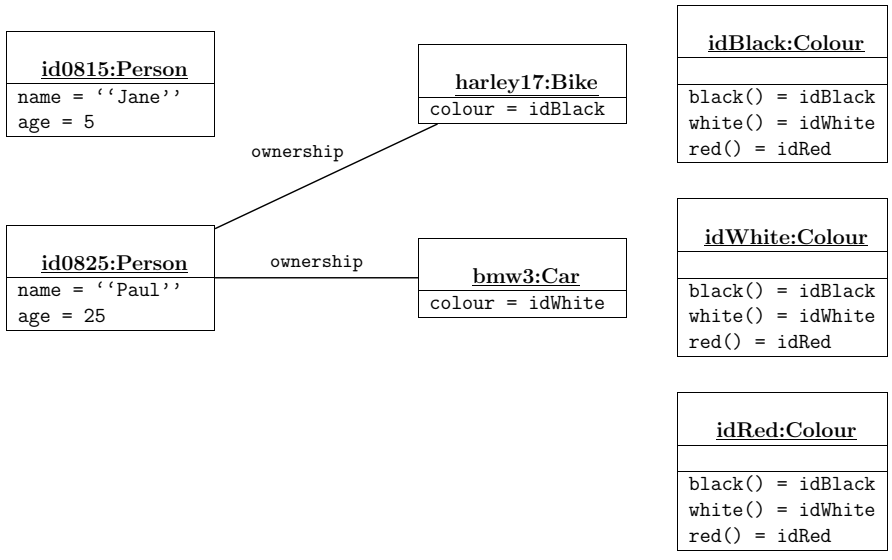
System state

(represented by a UML object diagram)



System state

(represented by a UML object diagram)

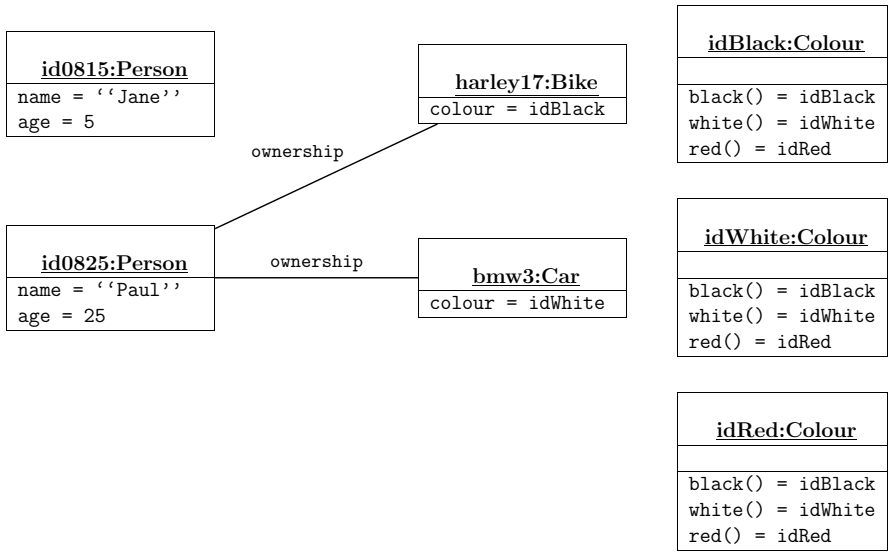


context Vehicle

inv: self.owner.age $>=$ 18

System state

(represented by a UML object diagram)

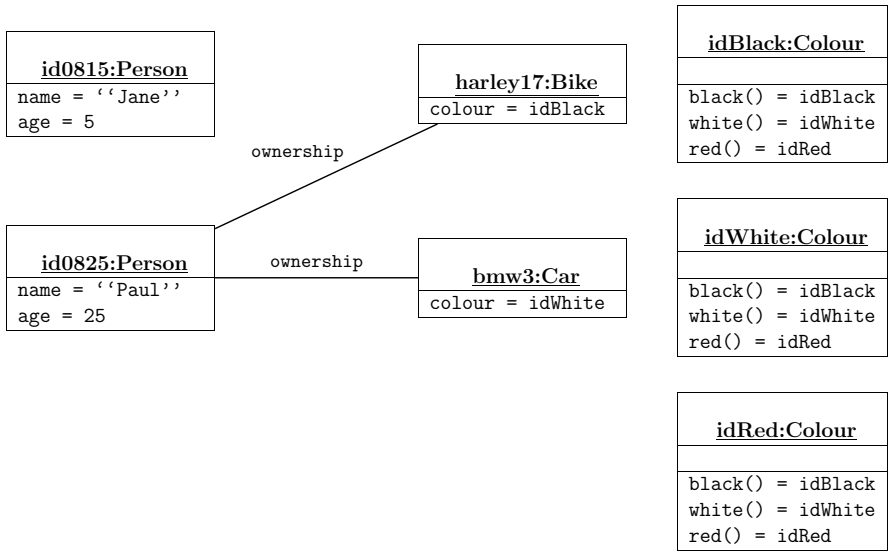


context Vehicle

inv: self.owner.age $>=$ 18 ✓

System state

(represented by a UML object diagram)

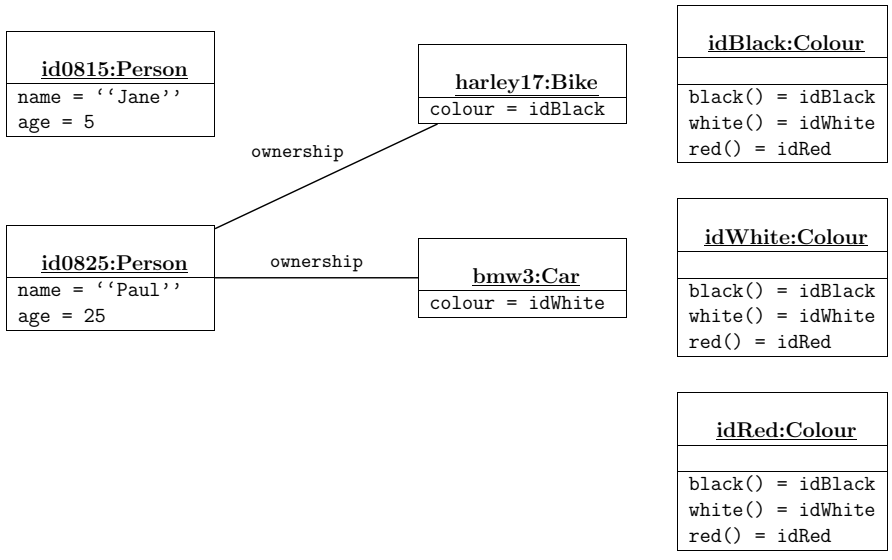


context Vehicle
inv: self.owner.age >= 18 ✓

context Person
inv: self.fleet->forAll(v | v.colour = #black)

System state

(represented by a UML object diagram)

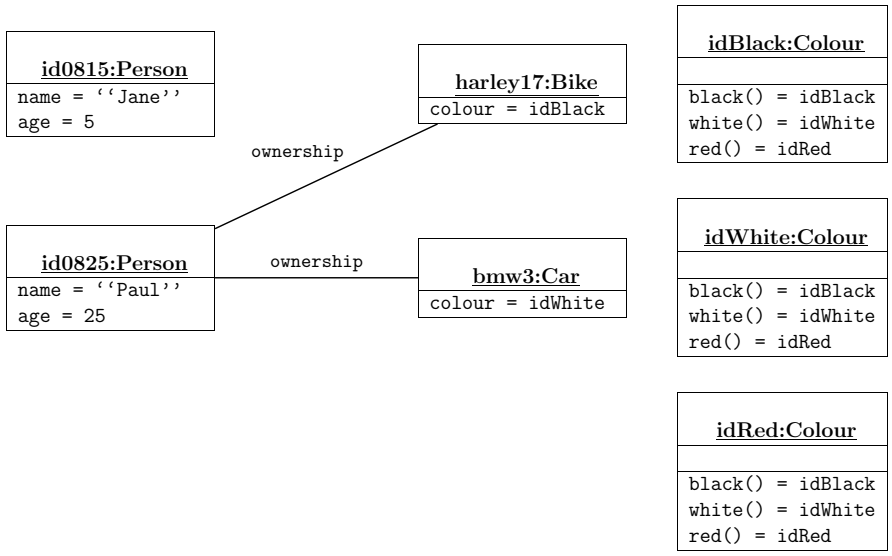


context Vehicle
inv: self.owner.age >= 18 ✓

context Person
inv: self.fleet->forAll(v | v.colour = #black) ☒

System state

(represented by a UML object diagram)



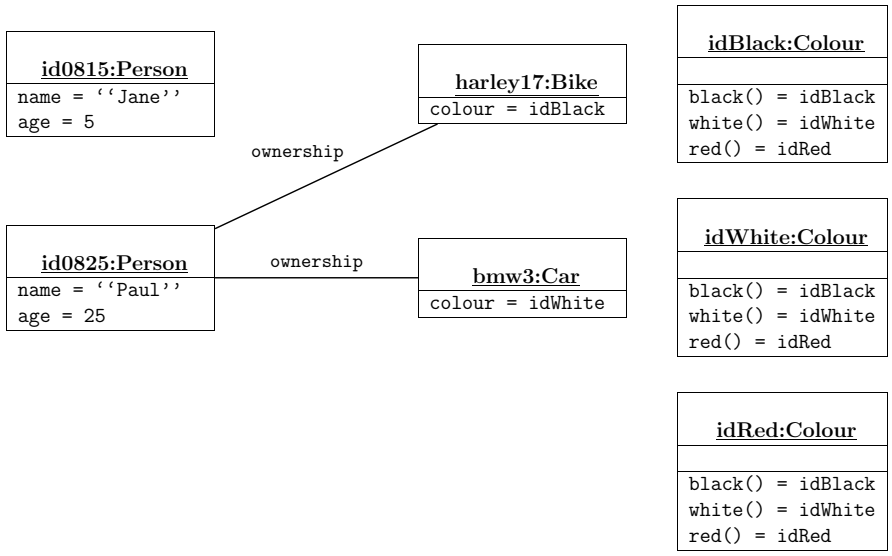
context Vehicle
inv: self.owner.age >= 18 ✓

context Person
inv: self.fleet->forAll(v | v.colour = #black) ☒

context Person
inv: self.fleet->select(v | v.colour = #black)->size <= 3

System state

(represented by a UML object diagram)



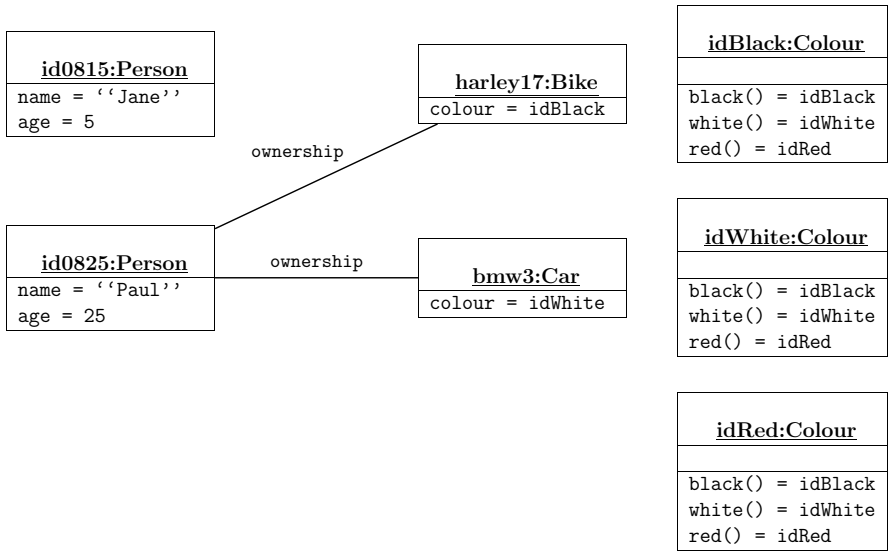
context Vehicle
inv: self.owner.age >= 18 ✓

context Person
inv: self.fleet->forAll(v | v.colour = #black) ☒

context Person
inv: self.fleet->select(v | v.colour = #black)->size <= 3 ✓

System state

(represented by a UML object diagram)



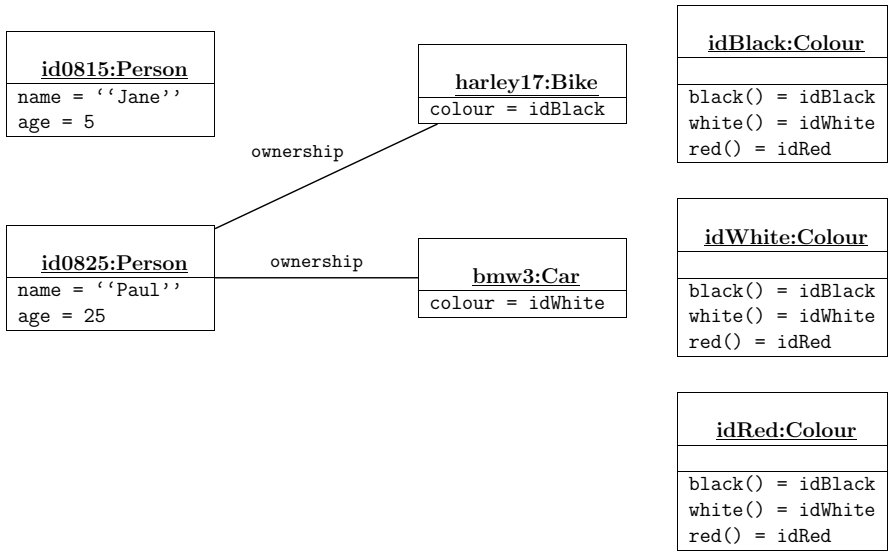
context Vehicle
inv: self.owner.age >= 18 ✓

context Person
inv: self.fleet->forAll(v | v.colour = #black) ☒

context Person
inv: self.fleet->select(v | v.colour = #black)->size <= 3 ✓

System state

(represented by a UML object diagram)



context Vehicle
inv: self.owner.age >= 18 ✓

context Person
inv: self.fleet->forAll(v | v.colour = #black) ☒

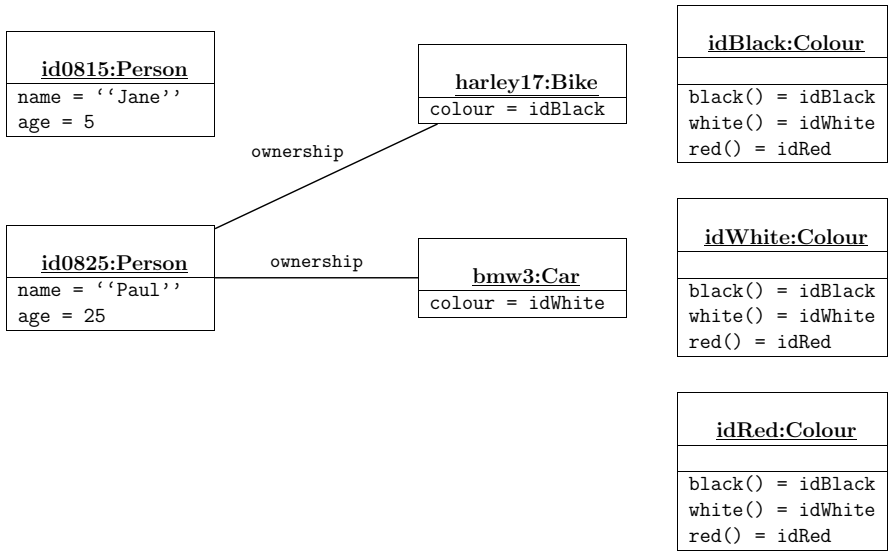
context Person
inv: self.fleet->select(v | v.colour = #black)->size <= 3 ✓

System State

(represented by a UML object diagram)

System State

(represented by a UML object diagram)



context Person::getName()
post: result = name **?**

System State

Given a UML class diagram, a system state (snapshot) is defined by

- **a UML object diagram (for the class diagram), giving**

System State

Given a UML class diagram, a system state (snapshot) is defined by

- **a UML object diagram (for the class diagram), giving**
  - **the set of existing instances,**

System State

Given a UML class diagram, a system state (snapshot) is defined by

- **a UML object diagram (for the class diagram), giving**
  - **the set of existing instances,**
  - **attribute-value-assignments**

System State

Given a UML class diagram, a system state (snapshot) is defined by

- **a UML object diagram (for the class diagram), giving**
  - **the set of existing instances,**
  - **attribute-value-assignments**
  - **instances of associations (“links”)**

System State

Given a UML class diagram, a system state (snapshot) is defined by

- **a UML object diagram (for the class diagram), giving**
  - **the set of existing instances,**
  - **attribute-value-assignments**
  - **instances of associations (“links”)**
- **an interpretation for operations,**

System State

Given a UML class diagram, a system state (snapshot) is defined by

- **a UML object diagram (for the class diagram), giving**
  - **the set of existing instances,**
  - **attribute-value-assignments**
  - **instances of associations (“links”)**
- **an interpretation for operations,**
- **(standard) interpretation for predefined primitive data types (e.g. Integer, String,. . .)**

System State

- **OCL Constraints are satisfied by certain system states.**

System State

- **OCL Constraints are satisfied by certain system states.**
- **Given an implementation of a class diagram, a sequence of system states is reached.**

System State

- **OCL Constraints are satisfied by certain system states.**
- **Given an implementation of a class diagram, a sequence of system states is reached.**
- **The interesting question is: How can we check that constraints are satisfied in all system states that are reached by an implementation?**

System State

- **OCL Constraints are satisfied by certain system states.**
- **Given an implementation of a class diagram, a sequence of system states is reached.**
- **The interesting question is: How can we check that constraints are satisfied in all system states that are reached by an implementation?**

  Answer in three weeks.

Literature

P. Schmitt:

Skriptum „Formale Spezifikationssprachen“

Jos Warmer and Anneke Kleppe:

The Object Constraint Language: Precise Modelling with UML

UML 1.5 OCL Specification.

http://www.omg.org/cgi-bin/apps/doc?ad/03-01-07.pdf

UML 2.0 OCL Revised submission to OMG.

http://www.omg.org/cgi-bin/apps/doc?ad/03-01-07.pdf