

Vorlesung "Software-Engineering"

Prof. Ralf Möller, TUHH, Arbeitsbereich STS
Übung: Miguel Garcia

- Heute:
 - Object Constraint Language (OCL)

Formal Object-oriented Software Development

Richard Bubel and Andreas Roth

November 5, 2004

Formal Specification with UML/OCL

Software Development: Building Models

**Real World
Rough Idea of System**

Java Implementation

Software Development: Building Models

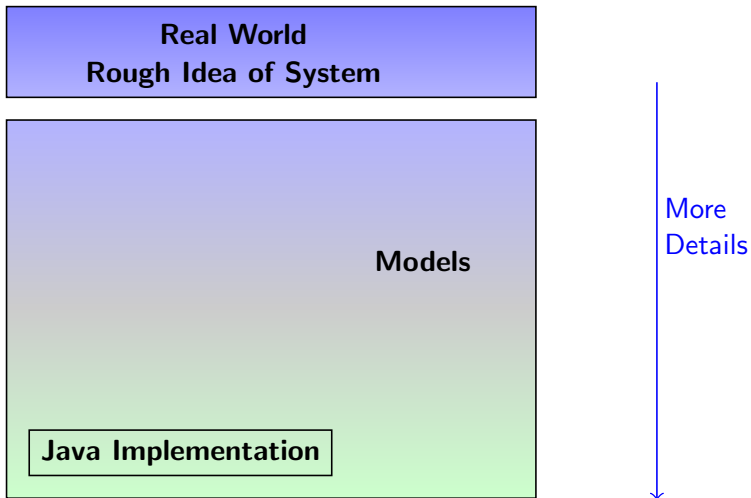
**Real World
Rough Idea of System**

Java Implementation

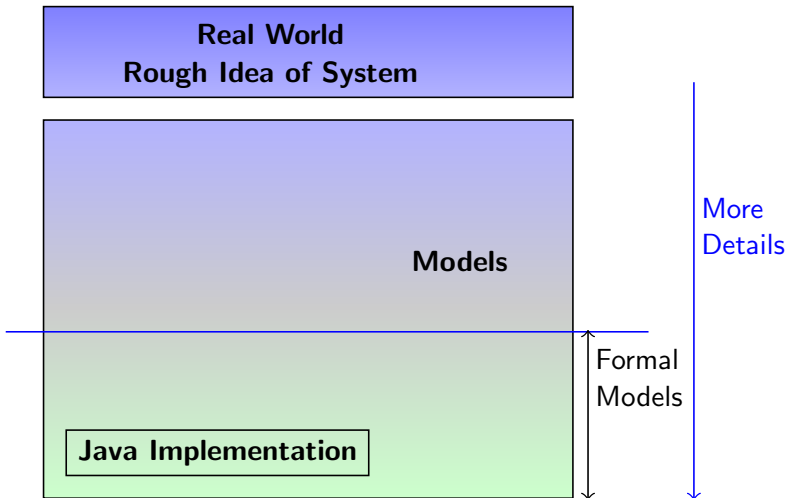
More
Details



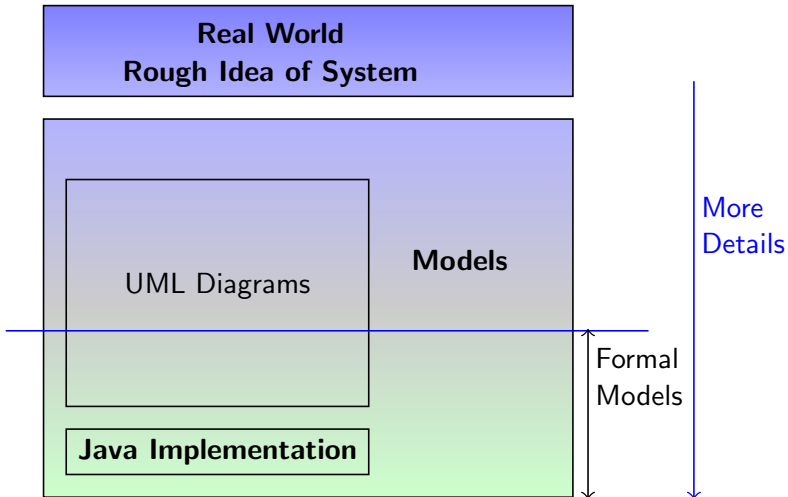
Software Development: Building Models



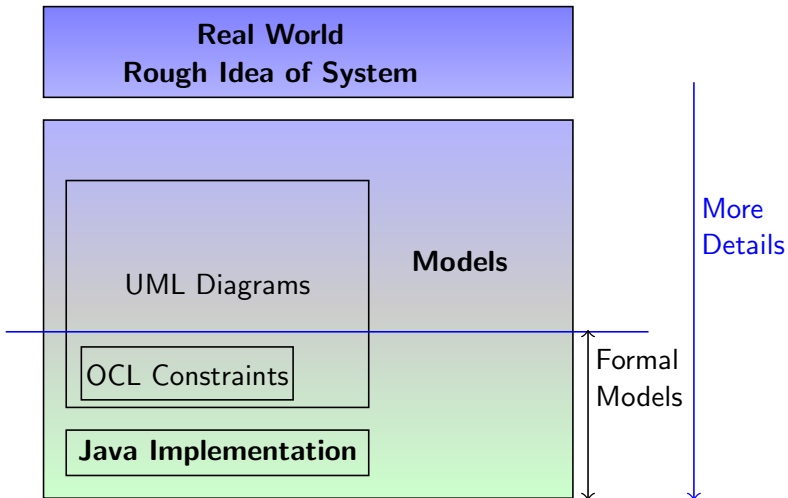
Software Development: Building Models



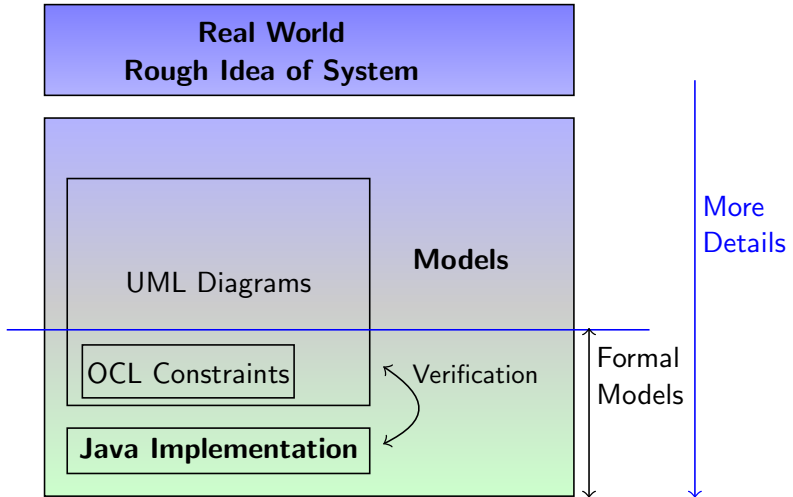
Software Development: Building Models



Software Development: Building Models



Software Development: Building Models



Why formal models/specification?

Specification means **Requirements Specification**

Advantages of formal requirements spec **before** implementation:

- No need to decide on algorithm, but sufficient to describe result

Why formal models/specification?

Specification means **Requirements Specification**

Advantages of formal requirements spec **before** implementation:

- No need to decide on algorithm, but sufficient to describe result
- Parts of behaviour can be left open (underspecification)

Why formal models/specification?

Specification means **Requirements Specification**

Advantages of formal requirements spec **before** implementation:

- No need to decide on algorithm, but sufficient to describe result
- Parts of behaviour can be left open (underspecification)
- Possibility of code generation, platform/technology independency
model-driven architecture

Why formal models/specification?

Specification means **Requirements Specification**

Advantages of formal requirements spec **before** implementation:

- No need to decide on algorithm, but sufficient to describe result
- Parts of behaviour can be left open (underspecification)
- Possibility of code generation, platform/technology independency
model-driven architecture
- Formalisation exhibits bugs & missing requirements in early stage

Why formal models/specification?

Specification means **Requirements Specification**

Advantages of formal requirements spec **before** implementation:

- No need to decide on algorithm, but sufficient to describe result
- Parts of behaviour can be left open (underspecification)
- Possibility of code generation, platform/technology independency
model-driven architecture
- Formalisation exhibits bugs & missing requirements in early stage

Why formal models/specification?

Specification means **Requirements Specification**

Advantages of formal requirements spec **before** implementation:

- No need to decide on algorithm, but sufficient to describe result
- Parts of behaviour can be left open (underspecification)
- Possibility of code generation, platform/technology independency
model-driven architecture
- Formalisation exhibits bugs & missing requirements in early stage

Two independent formal models (specification, code):

- Possibility of formal verification

Why formal models/specification?

Specification means **Requirements Specification**

Advantages of formal requirements spec **before** implementation:

- No need to decide on algorithm, but sufficient to describe result
- Parts of behaviour can be left open (underspecification)
- Possibility of code generation, platform/technology independency
model-driven architecture
- Formalisation exhibits bugs & missing requirements in early stage

Two independent formal models (specification, code):

- Possibility of formal verification
- Find more bugs

Why formal models/specification?

Specification means **Requirements Specification**

Advantages of formal requirements spec **before** implementation:

- No need to decide on algorithm, but sufficient to describe result
- Parts of behaviour can be left open (underspecification)
- Possibility of code generation, platform/technology independency
model-driven architecture
- Formalisation exhibits bugs & missing requirements in early stage

Two independent formal models (specification, code):

- Possibility of formal verification
- Find more bugs
- More trust in resulting system

Class Diagrams

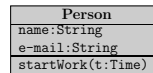
Key concepts:

Class Diagrams

Key concepts:

Class

Collection of similar objects in a system

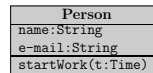


Class Diagrams

Key concepts:

Class

Collection of similar objects in a system



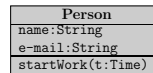
Attribute

Class Diagrams

Key concepts:

Class

Collection of similar objects in a system



Attribute

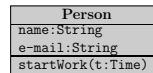
Operation/Method

Class Diagrams

Key concepts:

Class

Collection of similar objects in a system

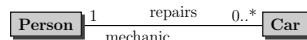


Attribute

Operation/Method

Association

Relation between classes (relates pairs of class instances)

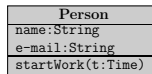


Class Diagrams

Key concepts:

Class

Collection of similar objects in a system

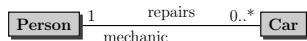


Attribute

Operation/Method

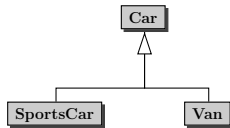
Association

Relation between classes (relates pairs of class instances)

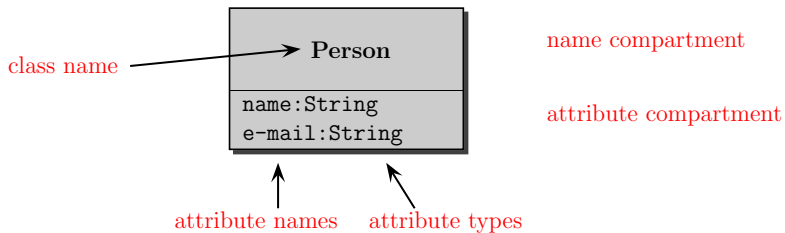


Generalisation

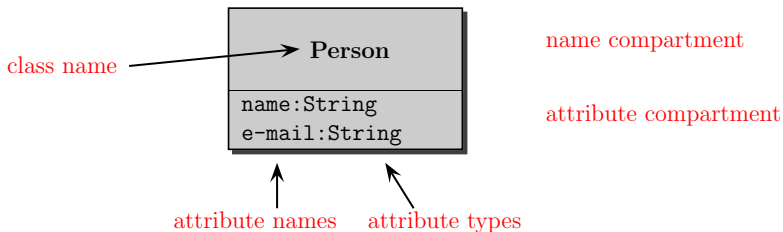
Specialisation-/Generalisation relationship between classes



Semantics of Classes

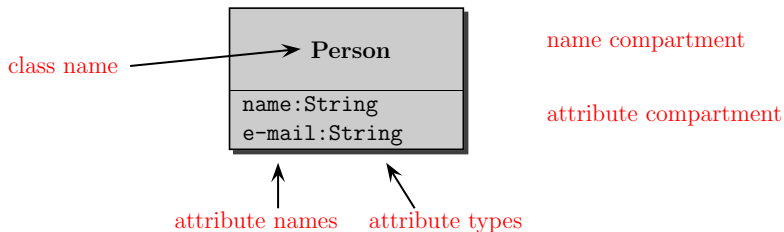


Semantics of Classes



$\mathcal{I}(\text{Person})$ is a (possibly empty) set of **objects**

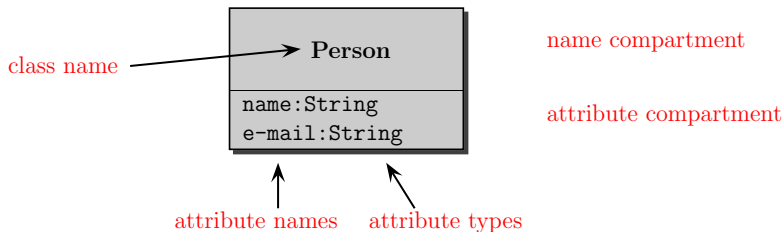
Semantics of Classes



$\mathcal{I}(\text{Person})$ is a (possibly empty) set of **objects**

$\mathcal{I}(\text{name})$ is a partial function from $\mathcal{I}(\text{Person})$ to $\mathcal{I}(\text{String})$

Semantics of Classes

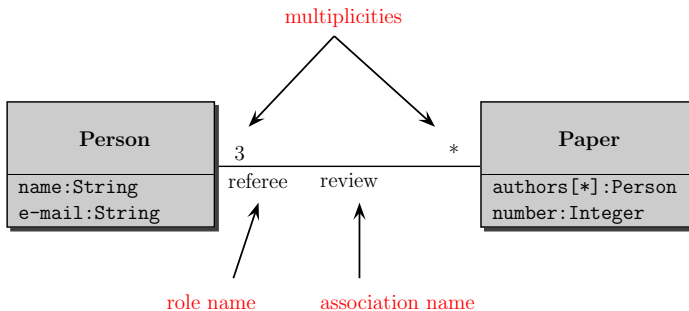


$\mathcal{I}(\text{Person})$ is a (possibly empty) set of **objects**

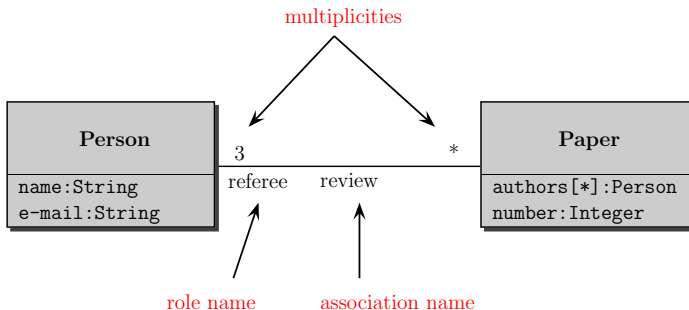
$\mathcal{I}(\text{name})$ is a partial function from $\mathcal{I}(\text{Person})$ to $\mathcal{I}(\text{String})$

$\mathcal{I}(\text{name})(\underline{aPerson})$ gives a string or is undefined

Semantics of Associations

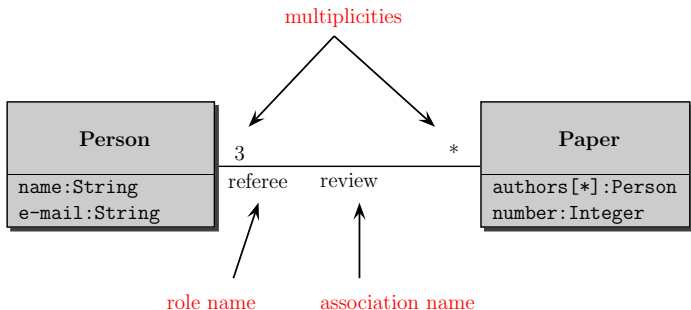


Semantics of Associations



$\mathcal{I}(\text{review})$ is a relation between $\mathcal{I}(\text{Person})$ and $\mathcal{I}(\text{Paper})$

Semantics of Associations



$\mathcal{I}(\text{review})$ is a relation between $\mathcal{I}(\text{Person})$ and $\mathcal{I}(\text{Paper})$

Multiplicity 3 requires:

for all $pap \in \mathcal{I}(\text{Paper})$:

$\text{card}(\{\text{pers} \in \mathcal{I}(\text{Person}) \mid \text{review}(\text{pers}, \text{pap})\}) \in \mathcal{I}(3) = \{3\}$

Semantics of Multiplicities

M	$\mathcal{I}(M)$
0..1	{0, 1}

Semantics of Multiplicities

M	$\mathcal{I}(M)$
0..1	$\{0, 1\}$
0..*	\mathbb{N}

Semantics of Multiplicities

M	$\mathcal{I}(M)$
0..1	$\{0, 1\}$
0..*	\mathbb{N}
*	\mathbb{N}

Semantics of Multiplicities

M	$\mathcal{I}(M)$
0..1	$\{0, 1\}$
0..*	\mathbb{N}
*	\mathbb{N}
1..3	$\{1, 2, 3\}$

Semantics of Multiplicities

M	$\mathcal{I}(M)$
0..1	$\{0, 1\}$
0..*	\mathbb{N}
*	\mathbb{N}
1..3	$\{1, 2, 3\}$
7	$\{7\}$

Semantics of Multiplicities

M	$\mathcal{I}(M)$
0..1	$\{0, 1\}$
0..*	\mathbb{N}
*	\mathbb{N}
1..3	$\{1, 2, 3\}$
7	$\{7\}$
15..19	$\{15, 16, 17, 18, 19\}$

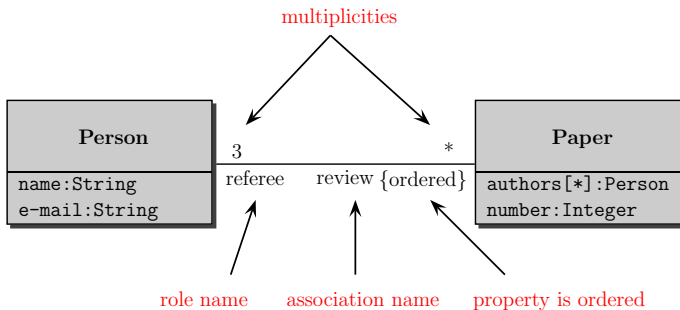
Semantics of Multiplicities

M	$\mathcal{I}(M)$
0..1	$\{0, 1\}$
0..*	\mathbb{N}
*	\mathbb{N}
1..3	$\{1, 2, 3\}$
7	$\{7\}$
15..19	$\{15, 16, 17, 18, 19\}$
1..3, 7, 15..19	$\{1, 2, 3, 7, 15, 16, 17, 18, 19\}$

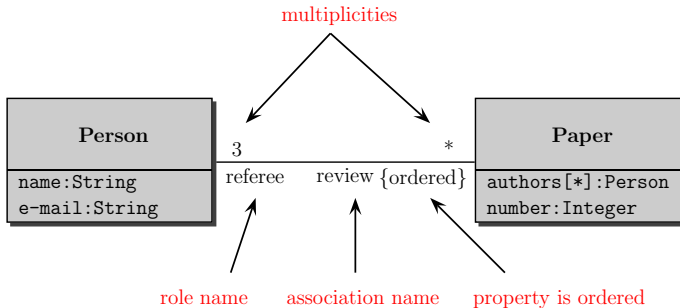
Semantics of Multiplicities

M	$\mathcal{I}(M)$
0..1	$\{0, 1\}$
0..*	\mathbb{N}
*	\mathbb{N}
1..3	$\{1, 2, 3\}$
7	$\{7\}$
15..19	$\{15, 16, 17, 18, 19\}$
1..3, 7, 15..19	$\{1, 2, 3, 7, 15, 16, 17, 18, 19\}$ (i.e., the separator "," acts as set theoretic union)

Semantics of Role Names

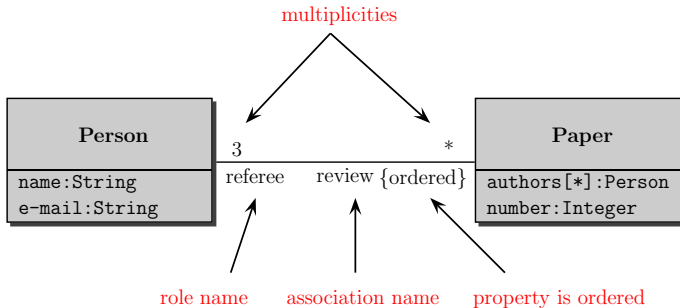


Semantics of Role Names



$$\mathcal{I}(\text{referee}) : \mathcal{I}(\overbrace{\text{Paper}}^{\text{Client}}) \rightarrow \text{Set}(\mathcal{I}(\overbrace{\text{Person}}^{\text{Supplier}}))$$

Semantics of Role Names



$$\begin{aligned}
 \mathcal{I}(\text{referee}) &: \mathcal{I}(\overbrace{\text{Paper}}^{\text{Client}}) \rightarrow \text{Set}(\mathcal{I}(\overbrace{\text{Person}}^{\text{Supplier}})) \\
 \mathcal{I}(\text{paper}) &: \mathcal{I}(\text{Person}) \rightarrow \text{Sequence}(\mathcal{I}(\text{Paper})) \quad (\text{default role})
 \end{aligned}$$

Snapshots

A **snapshot** of a given class diagram \mathcal{C} is a particular semantics \mathcal{I} of \mathcal{C}

- UML object diagram (for \mathcal{C}) including

Snapshots

A **snapshot** of a given class diagram \mathcal{C} is a particular semantics \mathcal{I} of \mathcal{C}

- UML object diagram (for \mathcal{C}) including
 - the set of currently existing instances $\mathcal{I}(C)$ of each class C

Snapshots

A **snapshot** of a given class diagram \mathcal{C} is a particular semantics \mathcal{I} of \mathcal{C}

- UML object diagram (for \mathcal{C}) including
 - the set of currently existing instances $\mathcal{I}(C)$ of each class C
 - maps $\mathcal{I}(a) : C \rightarrow C'$ for all attributes a of type C' of class C

Snapshots

A **snapshot** of a given class diagram \mathcal{C} is a particular semantics \mathcal{I} of \mathcal{C}

- UML object diagram (for \mathcal{C}) including
 - the set of currently existing instances $\mathcal{I}(C)$ of each class C
 - maps $\mathcal{I}(a) : C \rightarrow C'$ for all attributes a of type C' of class C
 - instances of associations (“links”): elements of $\mathcal{I}(C) \times \mathcal{I}(C')$

Snapshots

A **snapshot** of a given class diagram \mathcal{C} is a particular semantics \mathcal{I} of \mathcal{C}

- UML object diagram (for \mathcal{C}) including
 - the set of currently existing instances $\mathcal{I}(C)$ of each class C
 - maps $\mathcal{I}(a) : C \rightarrow C'$ for all attributes a of type C' of class C
 - instances of associations (“links”): elements of $\mathcal{I}(C) \times \mathcal{I}(C')$
- an interpretation for operations

Snapshots

A **snapshot** of a given class diagram \mathcal{C} is a particular semantics \mathcal{I} of \mathcal{C}

- UML object diagram (for \mathcal{C}) including
 - the set of currently existing instances $\mathcal{I}(C)$ of each class C
 - maps $\mathcal{I}(a) : C \rightarrow C'$ for all attributes a of type C' of class C
 - instances of associations (“links”): elements of $\mathcal{I}(C) \times \mathcal{I}(C')$
- an interpretation for operations
- (standard) interpretation of predefined primitive data types and their operations (Integer, String, ...)

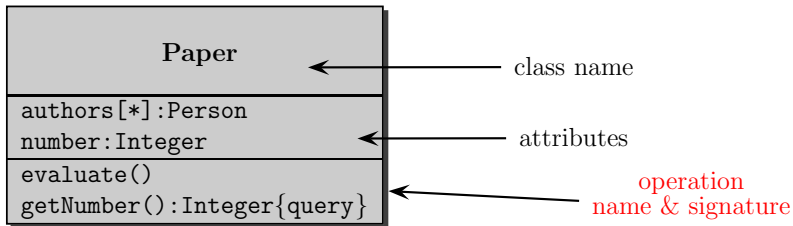
Snapshots

A **snapshot** of a given class diagram \mathcal{C} is a particular semantics \mathcal{I} of \mathcal{C}

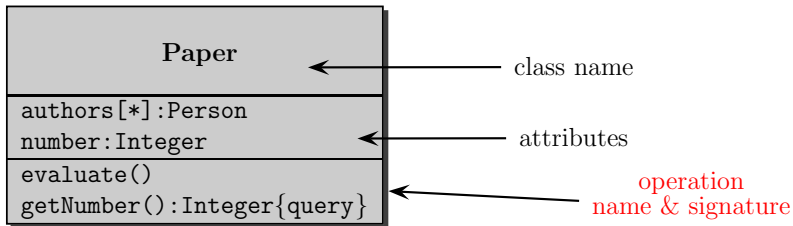
- UML object diagram (for \mathcal{C}) including
 - the set of currently existing instances $\mathcal{I}(C)$ of each class C
 - maps $\mathcal{I}(a) : C \rightarrow C'$ for all attributes a of type C' of class C
 - instances of associations (“links”): elements of $\mathcal{I}(C) \times \mathcal{I}(C')$
- an interpretation for operations
- (standard) interpretation of predefined primitive data types and their operations (Integer, String, ...)

Multiplicities and constraints restrict set of admissible snapshots

Semantics of Operations / Queries

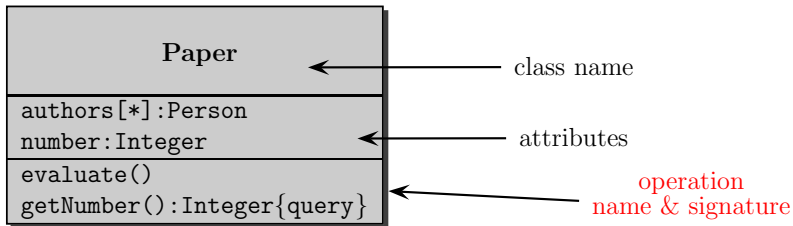


Semantics of Operations / Queries



Transition from snapshot to snapshot
(relation between sets of snapshots)

Semantics of Operations / Queries



Transition from snapshot to snapshot
(relation between sets of snapshots)

Queries: Partial function from owner and argument classes to result class

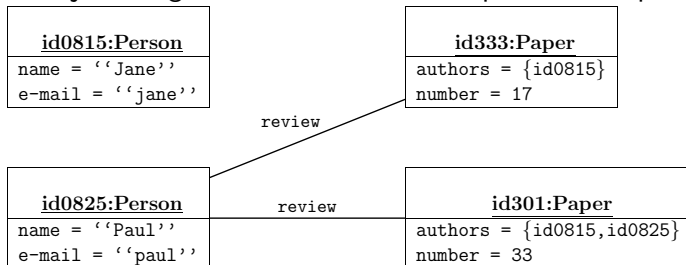
Object Diagram



Object Diagram



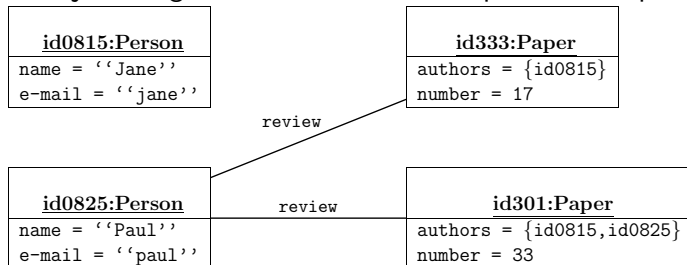
UML Object Diagrams describe the static part of a snapshot:



Object Diagram

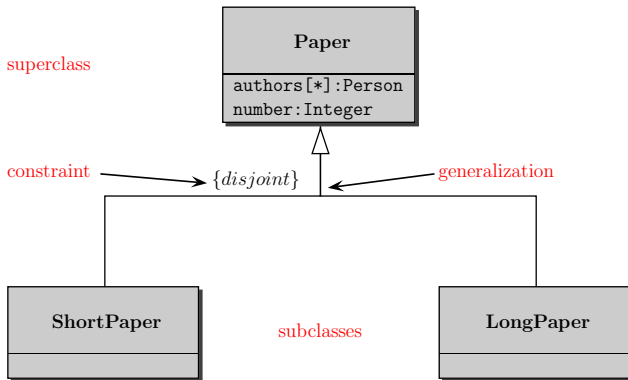


UML Object Diagrams describe the static part of a snapshot:



Illegal (and unintended) instance — Why?

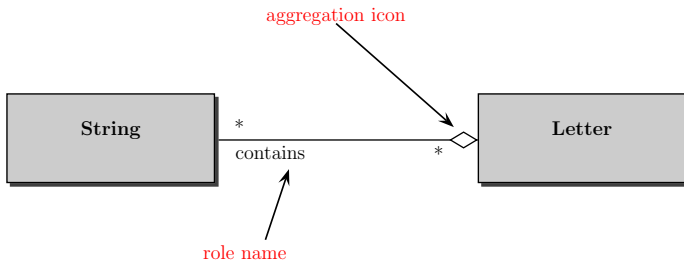
Semantics of Subclasses



subclass relation: $\mathcal{I}(\text{ShortPaper}) \subseteq \mathcal{I}(\text{Paper})$

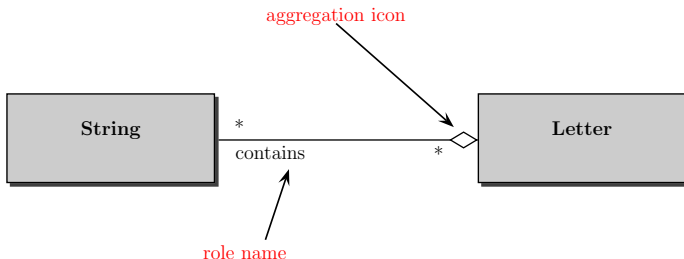
here disjointness-constraint: $\mathcal{I}(\text{ShortPaper}) \cap \mathcal{I}(\text{LongPaper}) = \emptyset$

Semantics of Aggregations



Same (formal) semantics as an association

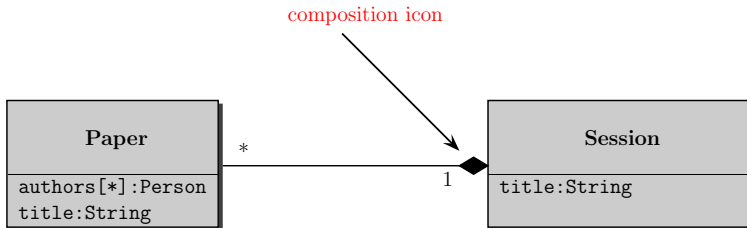
Semantics of Aggregations



Same (formal) semantics as an association

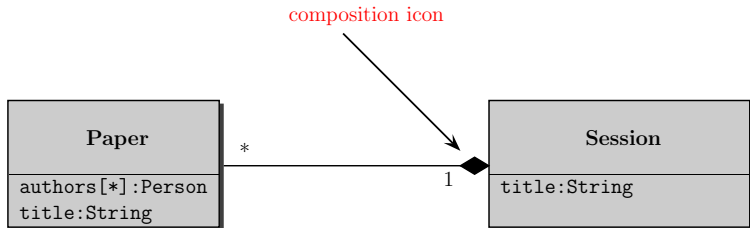
Pragmatics: Part-of relationship, though may be shared with other objects.

Semantics of Compositions



Same (formal) semantics as an association

Semantics of Compositions



Same (formal) semantics as an association

Pragmatics: Owned-by, object lifetime controlled by owner. Owner multiplicity 0..1 or 1

The Object Constraint Language (OCL)

- Part of the UML standard

The Object Constraint Language (OCL)

- Part of the UML standard
- Formal Specification Language
 - Standardised* formal semantics from OCL 2.0
 - In this course OCL 1.3; semantics by mapping to typed FOL

The Object Constraint Language (OCL)

- Part of the UML standard
- Formal Specification Language
 - Standardised* formal semantics from OCL 2.0
 - In this course OCL 1.3; semantics by mapping to typed FOL
- Syntax easier to read than Z, RSL, FOL, etc.

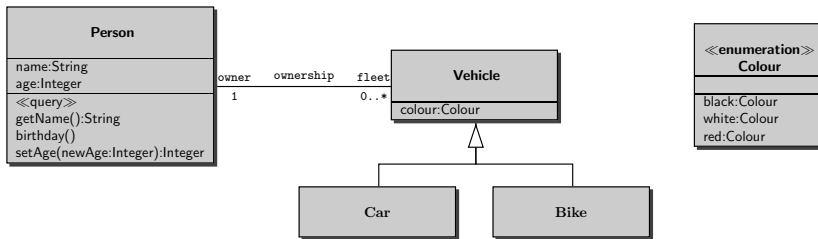
The Object Constraint Language (OCL)

- Part of the UML standard
- Formal Specification Language
 - Standardised* formal semantics from OCL 2.0
 - In this course OCL 1.3; semantics by mapping to typed FOL
- Syntax easier to read than Z, RSL, FOL, etc.
- Why?

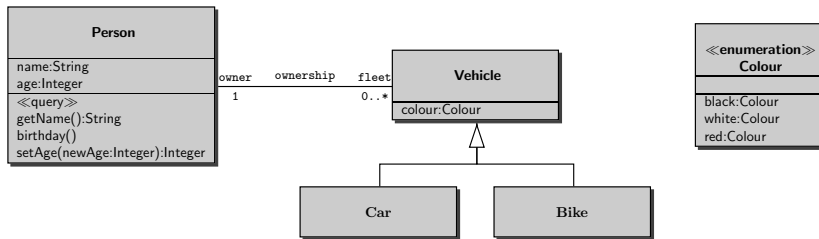
The Object Constraint Language (OCL)

- Part of the UML standard
- Formal Specification Language
 - Standardised* formal semantics from OCL 2.0
 - In this course OCL 1.3; semantics by mapping to typed FOL
- Syntax easier to read than Z, RSL, FOL, etc.
- Why? UML is not expressive enough!

UML is not enough. . .

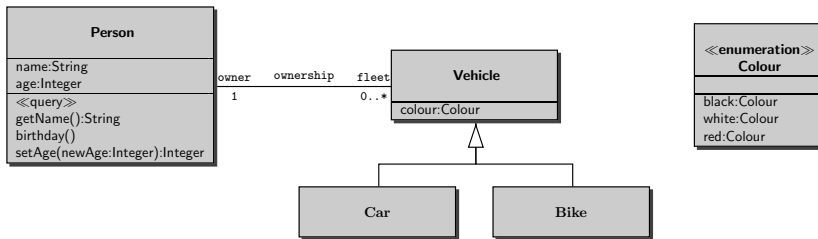


UML is not enough. . .



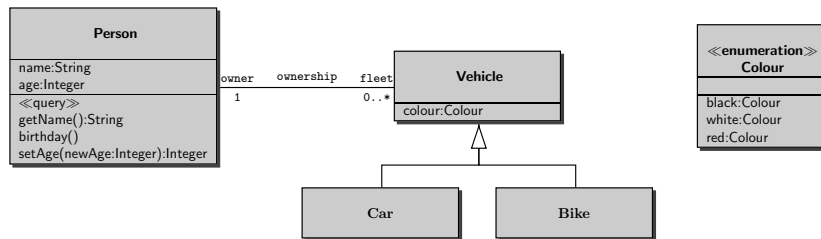
- How many persons can own a car?

UML is not enough. . .



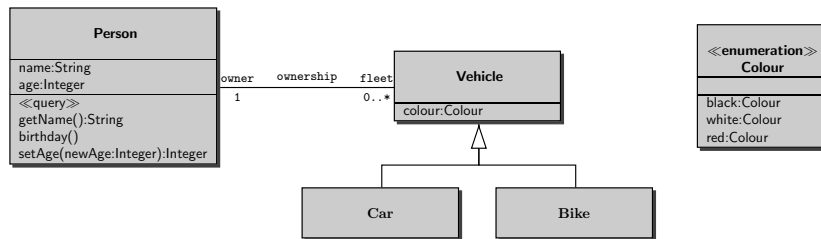
- How many persons can own a car?
- How old must a car owner be?

UML is not enough. . .



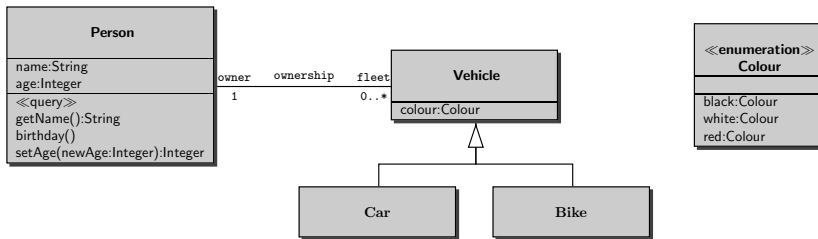
- How many persons can own a car?
- How old must a car owner be?
- How to express that a person can own at most own one black car?

UML is not enough. . .



- How many persons can own a car?
- How old must a car owner be?
- How to express that a person can own at most own one black car?
- How to specify that value of age is `i` after calling `setAge(i)`?

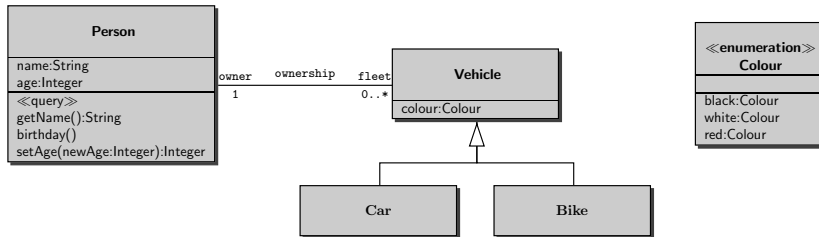
UML is not enough. . .



- How many persons can own a car?
- How old must a car owner be?
- How to express that a person can own at most own one black car?
- How to specify that value of age is *i* after calling `setAge(i)`?

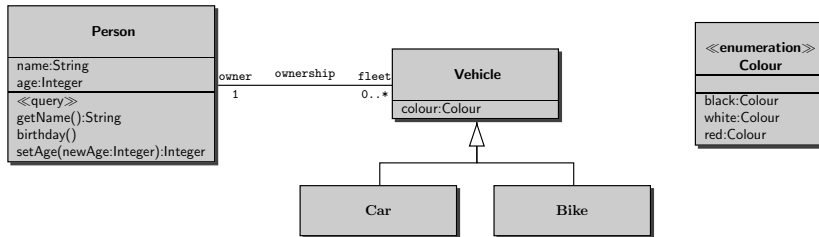
UML class diagrams unsuitable to express **semantical details** of design

OCL Examples



“No person owns more than 3 vehicles.”

OCL Examples

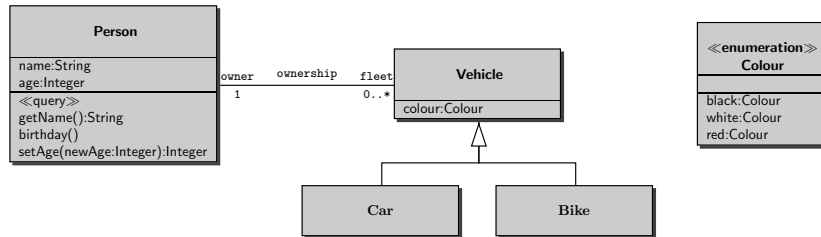


“No person owns more than 3 vehicles.”

context p:Person

inv: p.fleet->size <= 3

OCL Examples



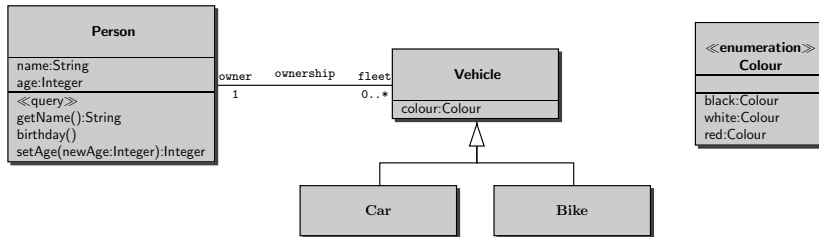
“No person owns more than 3 vehicles.”

context p:Person

inv: p.fleet->size <= 3

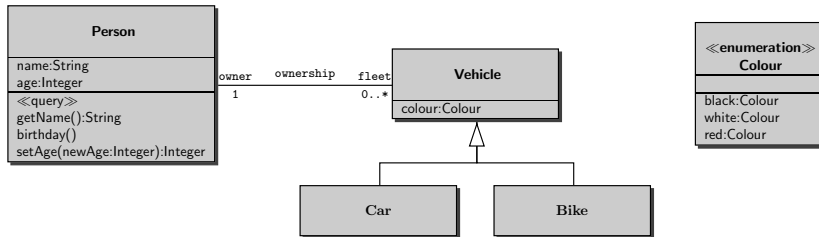
or change multiplicity

OCL Examples



“No person owns more than 3 **black** vehicles.”

OCL Examples

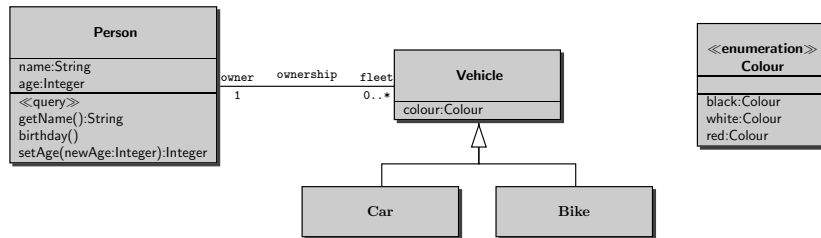


“No person owns more than 3 **black** vehicles.”

context p:Person

inv: p.fleet->select(v | v.colour = v.colour.black())->size <= 3

OCL Examples



“No person owns more than 3 **black** vehicles.”

context p:Person

inv: p.fleet→select(v | v.colour = v.colour.black())→size <= 3

OCL Constraints

There are **two** kinds of OCL constraints:

OCL Constraints

There are two kinds of OCL constraints:

- Invariant Constraints

Definition

An invariant ϕ_c attached to a class c is a boolean OCL expression. Its intended meaning is: ϕ_c **holds in every visible state of every instance of c .**

OCL Constraints

There are two kinds of OCL constraints:

- Invariant Constraints

Definition

An invariant ϕ_c attached to a class c is a boolean OCL expression. Its intended meaning is: ϕ_c **holds in every visible state of every instance of c .**

- Pre- and Post Conditions

Definition

A pre- and postcondition (ϕ, ψ) attached to an operation p is a pair of boolean OCL expressions.

Its intended meaning is: **If ϕ holds before the invocation of p then ψ holds after the invocation of p .**

OCL Constraints

There are two kinds of OCL constraints:

- Invariant Constraints

Definition

An invariant ϕ_c attached to a class c is a boolean OCL expression. Its intended meaning is: ϕ_c **holds in every visible state of every instance of c .**

- Pre- and Post Conditions

Definition

A pre- and postcondition (ϕ, ψ) attached to an operation p is a pair of boolean OCL expressions.

Its intended meaning is: **If ϕ holds before the invocation of p then ψ holds after the invocation of p .**

The Context

The **Context** determines to which element a constraint is attached.

The Context

The Context determines to which element a constraint is attached.

context c: typeName
inv: OclExpression₁
:
inv: OclExpression_{*n*}

The Context

The Context determines to which element a constraint is attached.

```
context   c: typeName
inv:     OclExpression1
           :
inv:     OclExpressionn
```

Example

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

The Context

The Context determines to which element a constraint is attached.

context c:typeName::opName($p_1:\text{type}_1, \dots, p_n:\text{type}_n$):rtype

pre: OclExpression₁^{pre}

post: OclExpression₁^{post}

:

pre: OclExpression_n^{pre}

post: OclExpression_n^{post}

The Context

The Context determines to which element a constraint is attached.

```
context   c:typeName::opName(p1:type1, . . . , pn:typen):rtype
pre:     OclExpression1pre
post:    OclExpression1post
           :
pre:     OclExpressionnpre
post:    OclExpressionnpost
```

Example

```
context   c:Person::getName():String
post:    result = c.name
```

The Context

The Context determines to which element a constraint is attached.

```

context   c:typeName::opName(p1:type1, . . . , pn:typen):rtype
pre:     OclExpression1pre
post:    OclExpression1post
           :
pre:     OclExpressionnpre
post:    OclExpressionnpost
  
```

Example

```

context   c:Person::getName():String
post:    result = c.name
  
```

“Calling getName() results in the value of the attribute name”

Pre-/ and Postconditions

Pre-/postconditions are like clauses in a **contract** of operation:

If *the caller* fulfills the precondition before the operation is called,
then the *called object* ensures the postcondition to hold after execution of the operation

Pre-/ and Postconditions

Pre-/postconditions are like clauses in a **contract** of operation:

If *the caller* fulfills the precondition before the operation is called,
then the *called object* ensures the postcondition to hold after execution of the operation

NOT

“Before executing an operation its precondition must hold”

“Whenever the precondition holds, the operation is called”

Pre-/ and Postconditions

Pre-/postconditions are like clauses in a **contract** of operation:

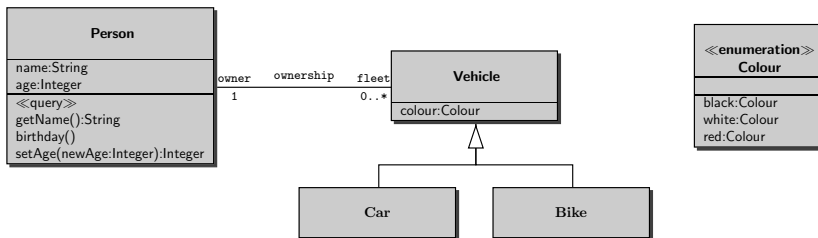
If *the caller* fulfills the precondition before the operation is called,
then the *called object* ensures the postcondition to hold after execution of the operation

NOT

“Before executing an operation its precondition must hold”

“Whenever the precondition holds, the operation is called”

OCL Expressions: Navigation

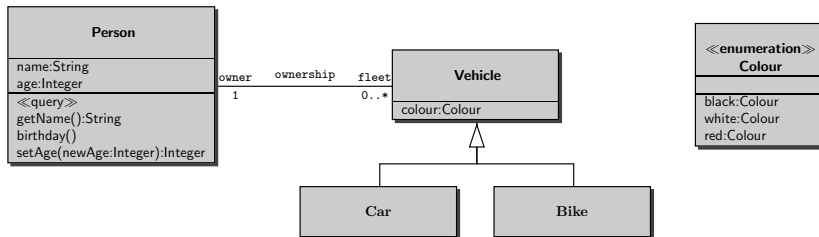


context p:Person

OCL expressions:

p	(Predefined) Variables
p.age	Attributes
p.getName()	Queries, not all operations
p.fleet	
p.fleet->size	Predefined Operations

OCL Expressions: Navigation



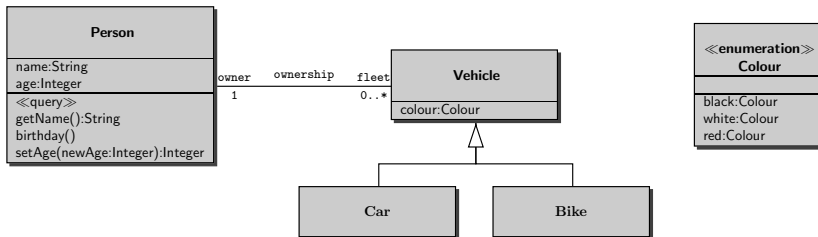
context p:Person

OCL expressions:

p	(Predefined) Variables
p.age	Attributes
p.getName()	Queries, not all operations
p.fleet	
p.fleet->size	Predefined Operations

OCL is a typed language.

OCL Expressions: Navigation



context p:Person

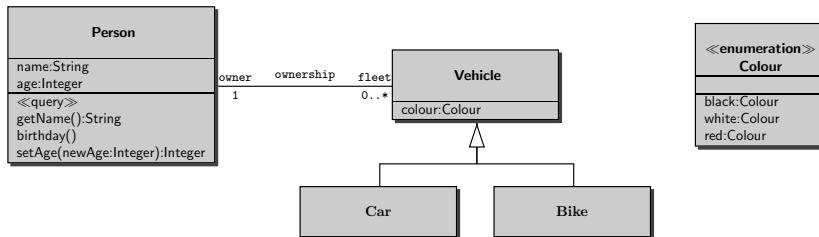
OCL expressions:

p	(Predefined) Variables
p.age	Attributes
p.getName()	Queries, not all operations
p.fleet	
p.fleet->size	Predefined Operations

OCL is a typed language.

Types:

OCL Expressions: Navigation



context p:Person

OCL expressions:

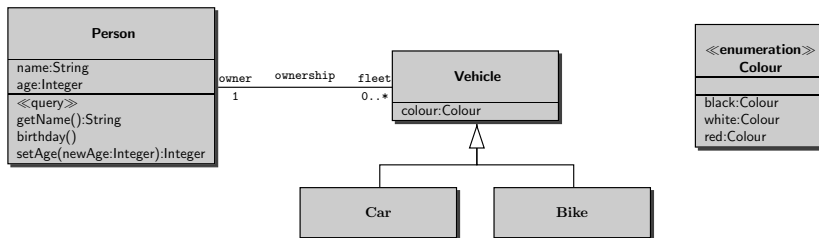
p	(Predefined) Variables
p.age	Attributes
p.getName()	Queries, not all operations
p.fleet	
p.fleet->size	Predefined Operations

OCL is a typed language.

Types:

Person

OCL Expressions: Navigation



context p:Person

OCL expressions:

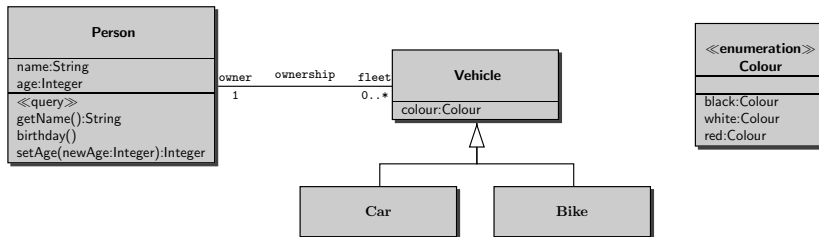
p	(Predefined) Variables
p.age	Attributes
p.getName()	Queries, not all operations
p.fleet	
p.fleet->size	Predefined Operations

OCL is a typed language.

Types:

Person
Integer

OCL Expressions: Navigation



context p:Person

OCL expressions:

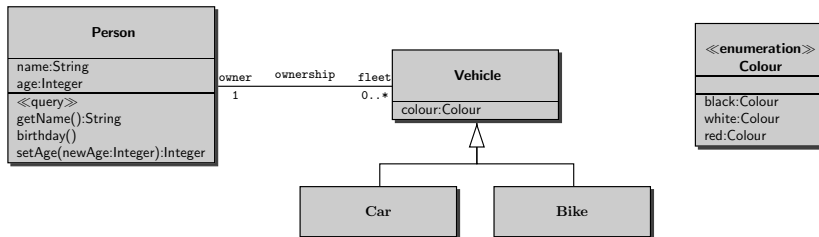
p	(Predefined) Variables
p.age	Attributes
p.getName()	Queries, not all operations
p.fleet	
p.fleet->size	Predefined Operations

OCL is a typed language.

Types:

Person
Integer
String

OCL Expressions: Navigation



context p:Person

OCL expressions:

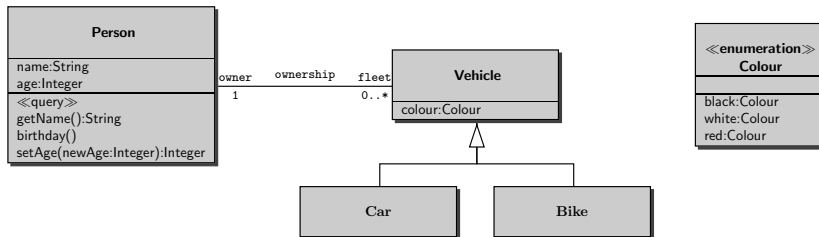
p	(Predefined) Variables
p.age	Attributes
p.getName()	Queries, not all operations
p.fleet	
p.fleet->size	Predefined Operations

OCL is a typed language.

Types:

Person
Integer
String
Set(Vehicle)

OCL Expressions: Navigation



context p:Person

OCL expressions:

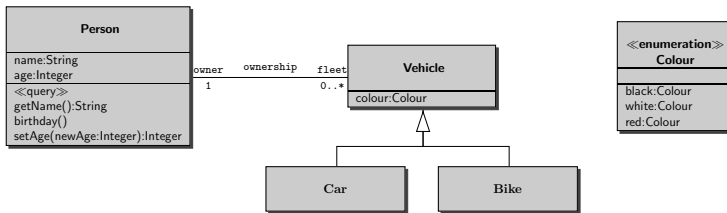
p	(Predefined) Variables
p.age	Attributes
p.getName()	Queries, not all operations
p.fleet	
p.fleet->size	Predefined Operations

OCL is a typed language.

Types:

Person
Integer
String
Set(Vehicle)
Integer

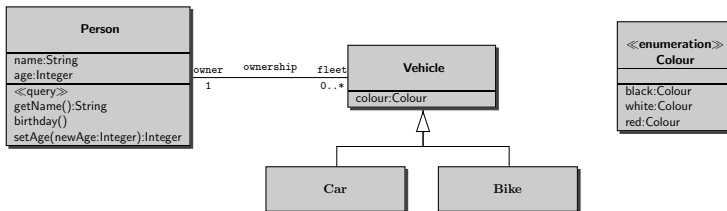
OCL Expressions: Predefined Variables



Context

Predefined Variables

OCL Expressions: Predefined Variables



Context

context p:Person

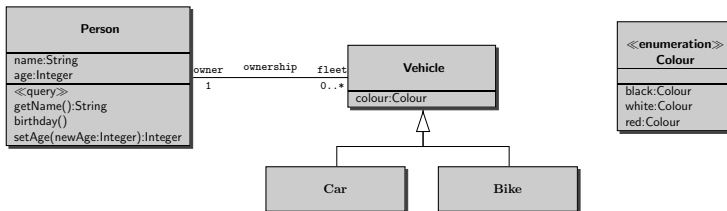
Predefined Variables

p

Example

context p:Person**inv:** p.fleet→size <= 3

OCL Expressions: Predefined Variables



Context

context p:Person**context** Person

Predefined Variables

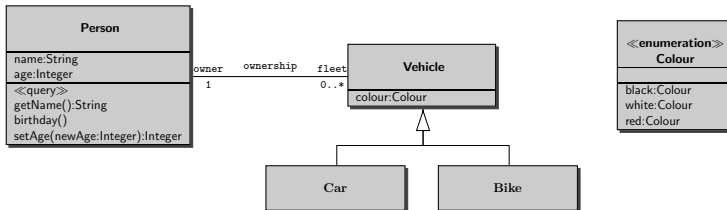
p

self

Example

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

OCL Expressions: Predefined Variables



Context

context p:Person

context Person

context Person::getName():String

Predefined Variables

p

self

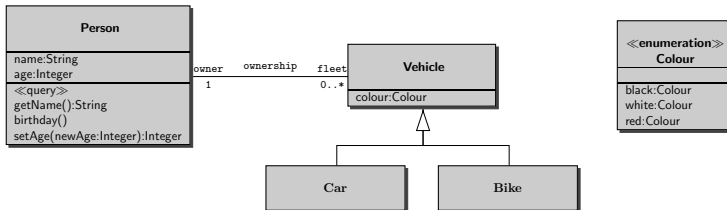
self, result

Example

context Person::getName():String

post: result = self.name

OCL Expressions: Predefined Variables



Context

context p:Person

context Person

context Person::getName():String

context Person::setAge(newAge: Integer)

Predefined Variables

p

self

self, result

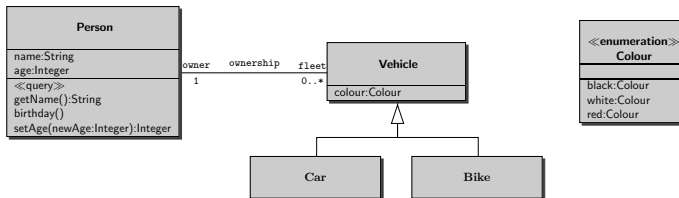
self, newAge

Example

context Person::setAge(newAge: Integer)

post: self.age=newAge

OCL Expressions: Predefined Operators (on collections)

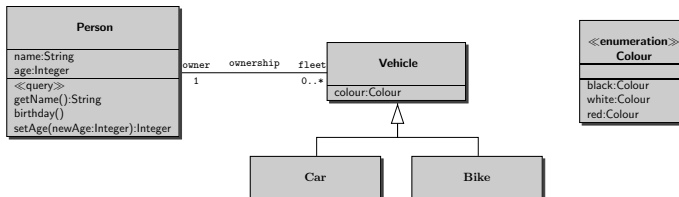


Operators	Result type	Meaning
<code>size</code>	Integer	Determines the size of a collection

Example

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

OCL Expressions: Predefined Operators (on collections)



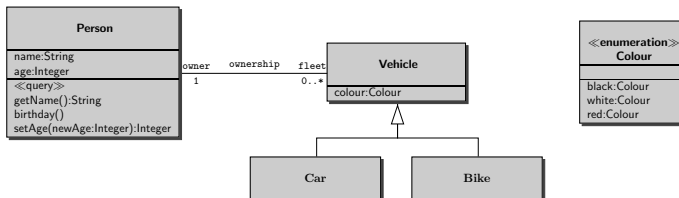
Operators	Result type	Meaning
size	Integer	Determines the size of a collection
select	Collection	Filters elements according to a condition

Example

context Person

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

OCL Expressions: Predefined Operators (on collections)



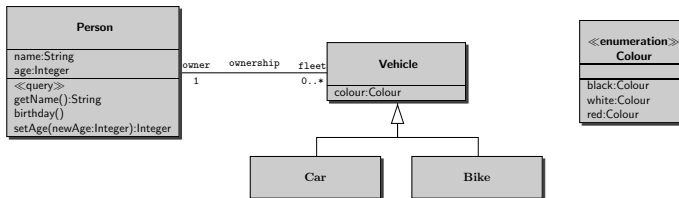
Operators	Result type	Meaning
size	Integer	Determines the size of a collection
select	Collection	Filters elements according to a condition
collect	Collection	Applies function to all elements

Example

context Person

inv: self.fleet->collect(v | v.colour)->asSet->size = 1
(asSet removes duplicates)

OCL Expressions: Predefined Operators (on collections)



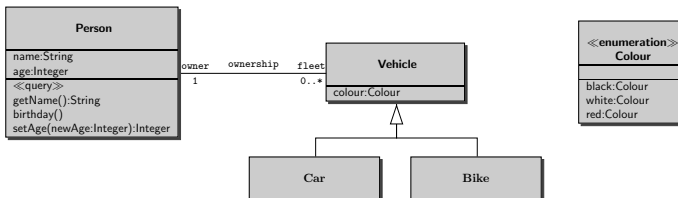
Operators	Result type	Meaning
size	Integer	Determines the size of a collection
select	Collection	Filters elements according to a condition
collect	Collection	Applies function to all elements
exist	Boolean	Does any element satisfy the condition?

Example

context Person

inv: self.fleet->exist(v | v.colour = v.colour.black())

OCL Expressions: Predefined Operators (on collections)



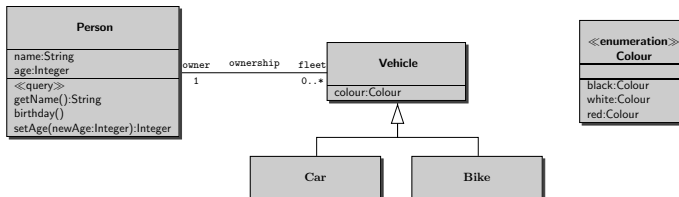
Operators	Result type	Meaning
<code>size</code>	Integer	Determines the size of a collection
<code>select</code>	Collection	Filters elements according to a condition
<code>collect</code>	Collection	Applies function to all elements
<code>forAll/exist</code>	Boolean	Does every/any element satisfy the condition?

Example

context Person

inv: `self.fleet->select(v | v.colour = v.colour.black())`
`->forAll(b | b.oclsKindOf(Bike))`

OCL Expressions: Predefined Operators (on collections)



Operators	Result type	Meaning
size	Integer	Determines the size of a collection
select	Collection	Filters elements according to a condition
collect	Collection	Applies function to all elements
forAll/exist	Boolean	Does every/any element satisfy the condition?
includes	Boolean	Is the specified element in the collection?

Example

context Person

inv: self.fleet->includes(self.favourite)

Notational Variants

context Person

inv: self.age \geq 18

Notational Variants

context Person
inv: self.age \geq 18

context c:Person
inv: c.age \geq 18

Notational Variants

context Person
inv: self.age \geq 18

context c:Person
inv: c.age \geq 18

context Person
inv: age \geq 18

Notational Variants

context Person
inv: self.age \geq 18

context c:Person
inv: c.age \geq 18

context Person
inv: age \geq 18

Notational Variants

context Person
inv: self.age \geq 18

context c:Person
inv: c.age \geq 18

context Person
inv: age \geq 18

Beware: variants using named instances don't work in TogetherCC!

Notational Variants

context Person
inv: self.age \geq 18

context c:Person
inv: c.age \geq 18

context Person
inv: age \geq 18

Beware: variants using named instances don't work in TogetherCC!

context Person - - all constraints are equivalent
inv: fleet \rightarrow collect(v:Vehicle | v.colour) \rightarrow size = 1

Notational Variants

context Person
inv: self.age \geq 18

context c:Person
inv: c.age \geq 18

context Person
inv: age \geq 18

Beware: variants using named instances don't work in TogetherCC!

context Person - - all constraints are equivalent
inv: fleet \rightarrow collect(v:Vehicle | v.colour) \rightarrow size = 1
inv: fleet \rightarrow collect(v | v.colour) \rightarrow size = 1

Notational Variants

context Person
inv: self.age \geq 18

context c:Person
inv: c.age \geq 18

context Person
inv: age \geq 18

Beware: variants using named instances don't work in TogetherCC!

context Person - - all constraints are equivalent
inv: fleet \rightarrow collect(v:Vehicle | v.colour) \rightarrow size = 1
inv: fleet \rightarrow collect(v | v.colour) \rightarrow size = 1
inv: fleet \rightarrow collect(colour) \rightarrow size = 1

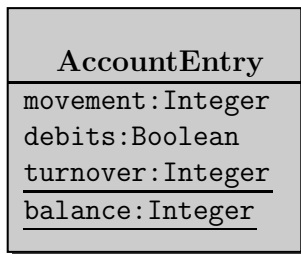
Notational Variants

context	Person	context	c:Person	context	Person
inv:	self.age \geq 18	inv:	c.age \geq 18	inv:	age \geq 18

Beware: variants using named instances don't work in TogetherCC!

context	Person	- - all constraints are equivalent
inv:	fleet	\rightarrow collect(v:Vehicle v.colour) \rightarrow size = 1
inv:	fleet	\rightarrow collect(v v.colour) \rightarrow size = 1
inv:	fleet	\rightarrow collect(colour) \rightarrow size = 1
inv:	fleet.colour	\rightarrow size = 1 - - shorthand only valid for 'collect'

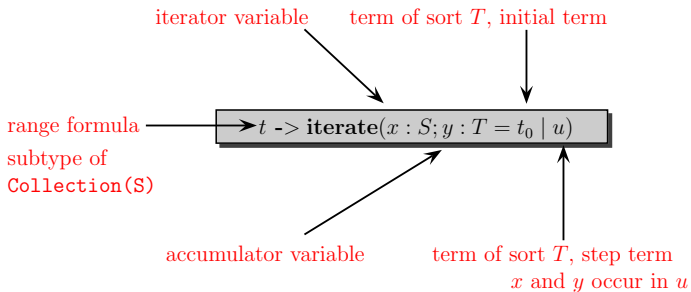
allInstances and iterate



context AccountEntry

inv: AccountEntry.allInstances \rightarrow
iterate(a:AccountEntry ; m:Integer=0 | m+a.movement) =
AccountEntry.turnover

iterate Syntax



iterate Semantics

$$t \rightarrow \text{iterate}(x:S; y:T=t0 \mid u)$$

iterate Semantics

$$t \rightarrow \text{iterate}(x:S; y:T=t0 \mid u)$$

Java pseudocode:

```
S x;  
T y = t0;  
for (Iterator it = t.iterator(); it.hasNext();) {  
    x = it.next();  
    y = u(x,y);  
}
```

iterate Semantics

$$t \rightarrow \text{iterate}(x:S; y:T=t0 \mid u)$$

Java pseudocode:

```
S x;  
T y = t0;  
for (Iterator it = t.iterator(); it.hasNext();) {  
    x = it.next();  
    y = u(x,y);  
}
```

Type of x and y can be inferred from t and u

Optional in OCL, but needed for KeY's OCL parser

Need to use **Boolean** and **int**

iterate Semantics

$$t \rightarrow \text{iterate}(x:S; y:T=t0 \mid u)$$

Java pseudocode:

```
S x;  
T y = t0;  
for (Iterator it = t.iterator(); it.hasNext();) {  
    x = it.next();  
    y = u(x,y);  
}
```

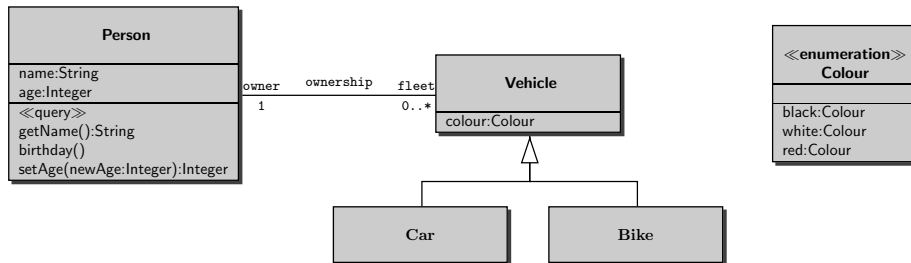
Type of x and y can be inferred from t and u

Optional in OCL, but needed for KeY's OCL parser

Need to use **Boolean** and **int**

iterate can be used to simulate the other collection operations

Referring to Previous Values



context Person::birthday()
pre: age \geq 0
post: age = age@pre + 1

Junctors

Of course there are logical junctors in OCL. The following are boolean OCL expressions, if e_1 and e_2 are boolean OCL expressions:

- not e_1
- e_1 and e_2
- e_1 or e_2
- e_1 implies e_2 (logical implication)
- $e_1 = e_2$ (logical equivalence)
- if e_1 then e_2 else e_3 endif
equivalent to
(e_1 implies e_2) and (not e_1 implies e_3)

Parentheses are allowed to change precedence.

OCL in TogetherCC/KeY

KeY provides an integrated OCL parser / type checker for TogetherCC

Use the fields *preconditions*, *postconditions*, and *invariants* in the properties pane of operations and classes (resp.)

Snapshots and OCL Constraints

- OCL constraints evaluated relative to a snapshot \mathcal{I}

Snapshots and OCL Constraints

- OCL constraints evaluated relative to a snapshot \mathcal{I}
- OCL expressions have type Boolean \Rightarrow they are true or false wrt \mathcal{I}

Snapshots and OCL Constraints

- OCL constraints evaluated relative to a snapshot \mathcal{I}
- OCL expressions have type Boolean \Rightarrow they are true or false wrt \mathcal{I}
- OCL constraints restrict legal snapshots of UML diagram

Possibility to express intended semantics of diagram

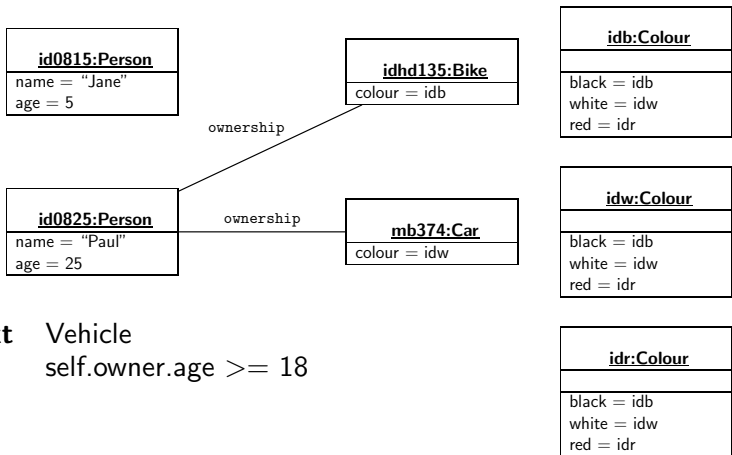
Snapshots and OCL Constraints

- OCL constraints evaluated relative to a snapshot \mathcal{I}
- OCL expressions have type Boolean \Rightarrow they are true or false wrt \mathcal{I}
- OCL constraints restrict legal snapshots of UML diagram

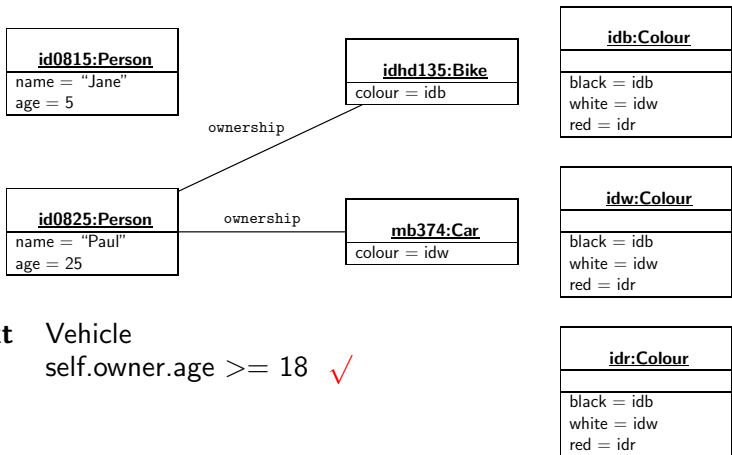
Possibility to express intended semantics of diagram

- Could define formal semantics of OCL in terms of snapshots
In KeY: semantics by translation to first order logic

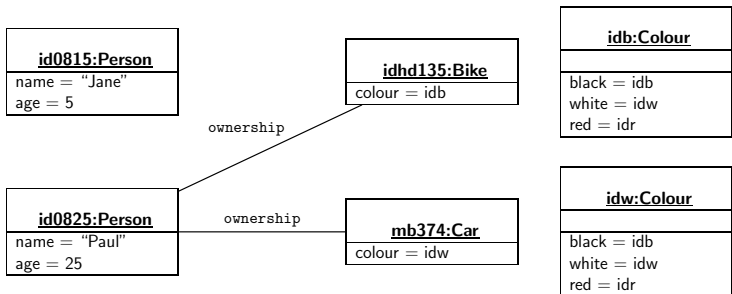
Snapshots and OCL Constraints



Snapshots and OCL Constraints



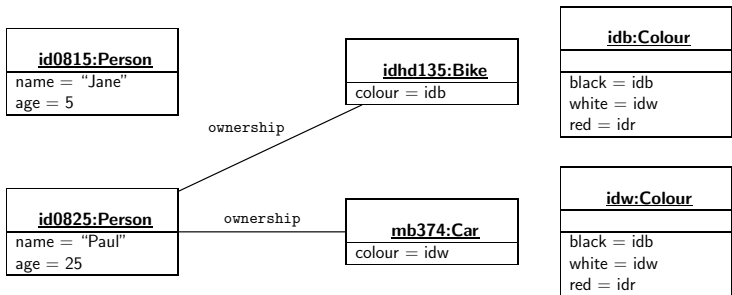
Snapshots and OCL Constraints



context Vehicle
inv: self.owner.age ≥ 18 ✓

context Person
inv: fleet \rightarrow forAll(colour = colour.black())

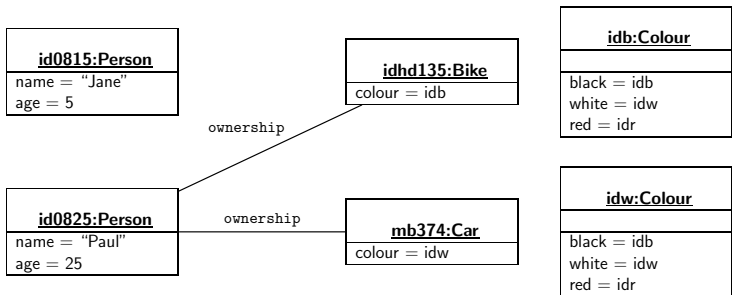
Snapshots and OCL Constraints



context Vehicle
inv: self.owner.age ≥ 18 ✓

context Person
inv: fleet \rightarrow forAll(colour = colour.black()) ☒

Snapshots and OCL Constraints

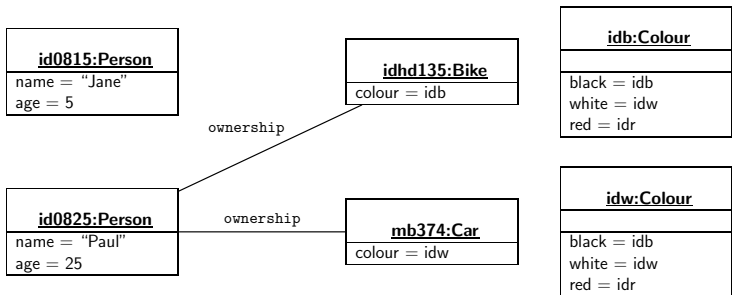


context Vehicle
inv: self.owner.age ≥ 18 ✓

context Person
inv: fleet \rightarrow forAll(colour = colour.black()) ☒

context Person
inv: fleet \rightarrow select(colour = colour.black()) \rightarrow size ≤ 3

Snapshots and OCL Constraints

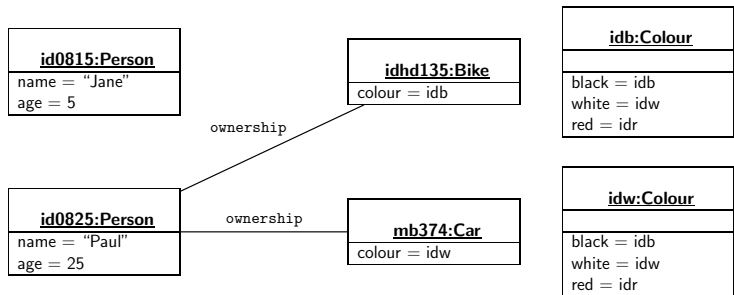


context Vehicle
inv: self.owner.age \geq 18 ✓

context Person
inv: fleet \rightarrow forAll(colour = colour.black()) ☒

context Person
inv: fleet \rightarrow select(colour = colour.black()) \rightarrow size \leq 3 ✓

Snapshots and OCL Constraints



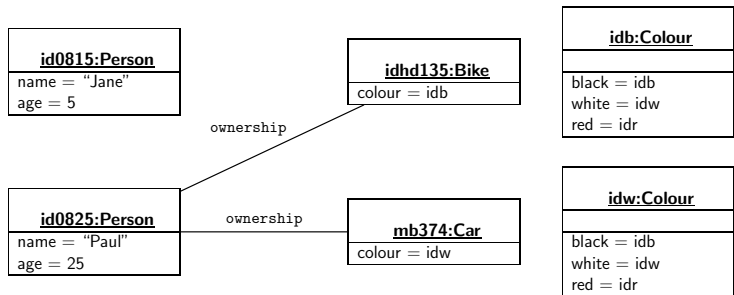
context Vehicle
inv: self.owner.age ≥ 18 ✓

context Person
inv: fleet \rightarrow forAll(colour = colour.black()) ☒

context Person
inv: fleet \rightarrow select(colour = colour.black()) \rightarrow size ≤ 3 ✓

inv: Car.allInstances \rightarrow exists(colour = colour.red())

Snapshots and OCL Constraints



context Vehicle

inv: self.owner.age ≥ 18 ✓

context Person

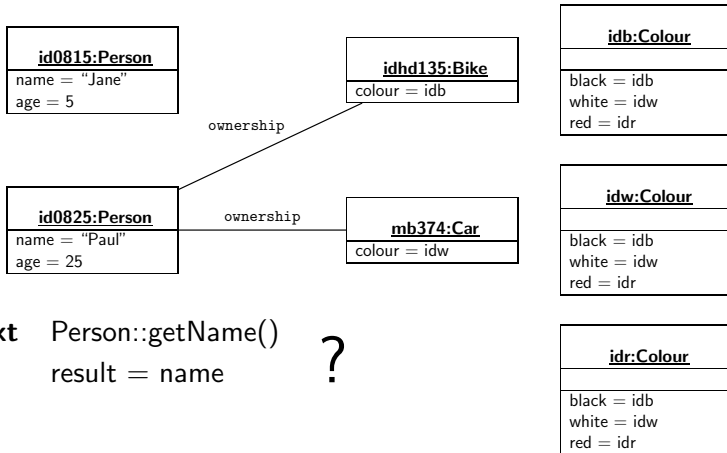
inv: fleet \rightarrow forAll(colour = colour.black()) ☒

context Person

inv: fleet \rightarrow select(colour = colour.black()) \rightarrow size ≤ 3 ✓

inv: Car.allInstances \rightarrow exists(colour = colour.red()) ☒

Snapshots and OCL Constraints



Summary

- UML class diagrams (and OCL constraints) can be given a rigorous semantics using *snapshots*.
- There are two kinds of OCL constraints: invariants and pre-/postconditions, distinguishable by context.
- OCL expressions can talk about collections. Important constructs: size, select, collect, forAll, exist, includes, iterate
- In postcondition, @pre allows to refer to values before method invocation.