# Model-Driven Development

# Episode 4

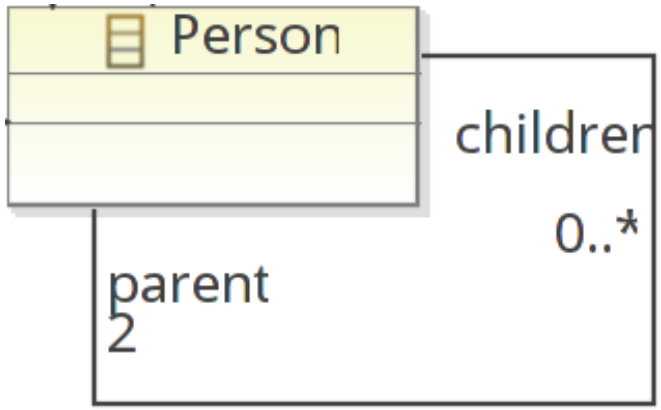
## Object Constraint Language

**Andrzej Wąsowski**

**Thorsten Berger**

IT UNIVERSITY OF COPENHAGEN

PROCESS AND SYSTEM MODELS GROUP


Person
children
0..*
parent
2

- General Characteristics of OCL
- OCL Syntax and Semantics by Example
- Crash Summary of OCL
- Playing with EMF and OCL Practically

# AGENDA

*The Object Constraint Language (OCL) [is] a formal language used to describe expressions on UML models. These expressions typically specify invariant conditions that must hold for the system being modeled or queries over objects described in a model. Note that when the OCL expressions are evaluated, they do not have side effects (i.e., their evaluation cannot alter the state of the corresponding executing system).[OCL specification]*

# OCL Characteristics

- Declarative
- First order predicate logics
- Quantifiers hidden as collection iterators
- Strongly typed
- Used to express invariants over classes
- Used to express pre- and post-conditions for methods

# OCL Characteristics

- Declarative
- First order predicate logics
- Quantifiers hidden as collection iterators
- Strongly typed
- Used to express invariants over classes
- Used to express pre- and post-conditions for methods

# OCL Characteristics

- Declarative
- First order predicate logics
- Quantifiers hidden as collection iterators
- Strongly typed
- Used to express invariants over classes
- Used to express pre- and post-conditions for methods

# OCL Characteristics

- Declarative
- First order predicate logics
- Quantifiers hidden as collection iterators
- Strongly typed
- Used to express invariants over classes
- Used to express pre- and post-conditions for methods

# OCL Characteristics

- Declarative
- First order predicate logics
- Quantifiers hidden as collection iterators
- Strongly typed
- Used to express invariants over classes
- Used to express pre- and post-conditions for methods

# OCL Characteristics

- Declarative
- First order predicate logics
- Quantifiers hidden as collection iterators
- Strongly typed
- Used to express invariants over classes
- Used to express pre- and post-conditions for methods

- Introduction
- **OCL Syntax and Semantics by Example**
- Crash Summary of OCL
- Playing with EMF and OCL Practically

# AGENDA

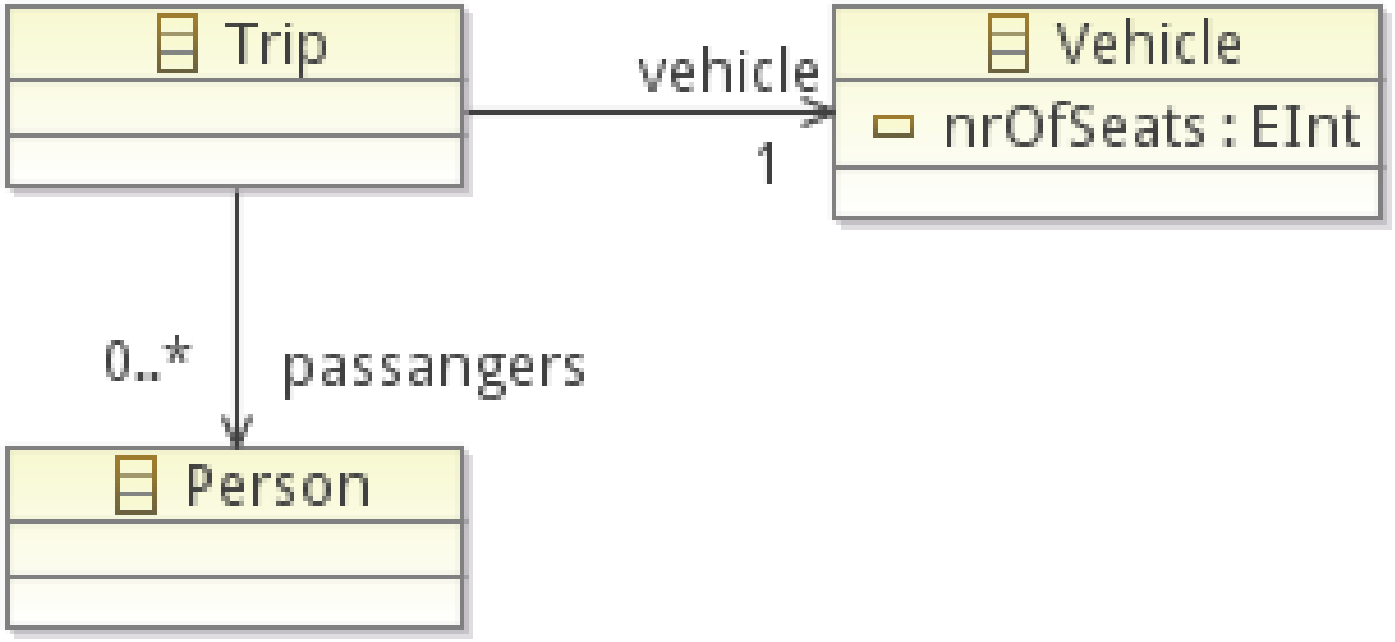
# OCL Syntax

## Invariants

```
context ClassName
inv: OCL-expression
```

# OCL Syntax

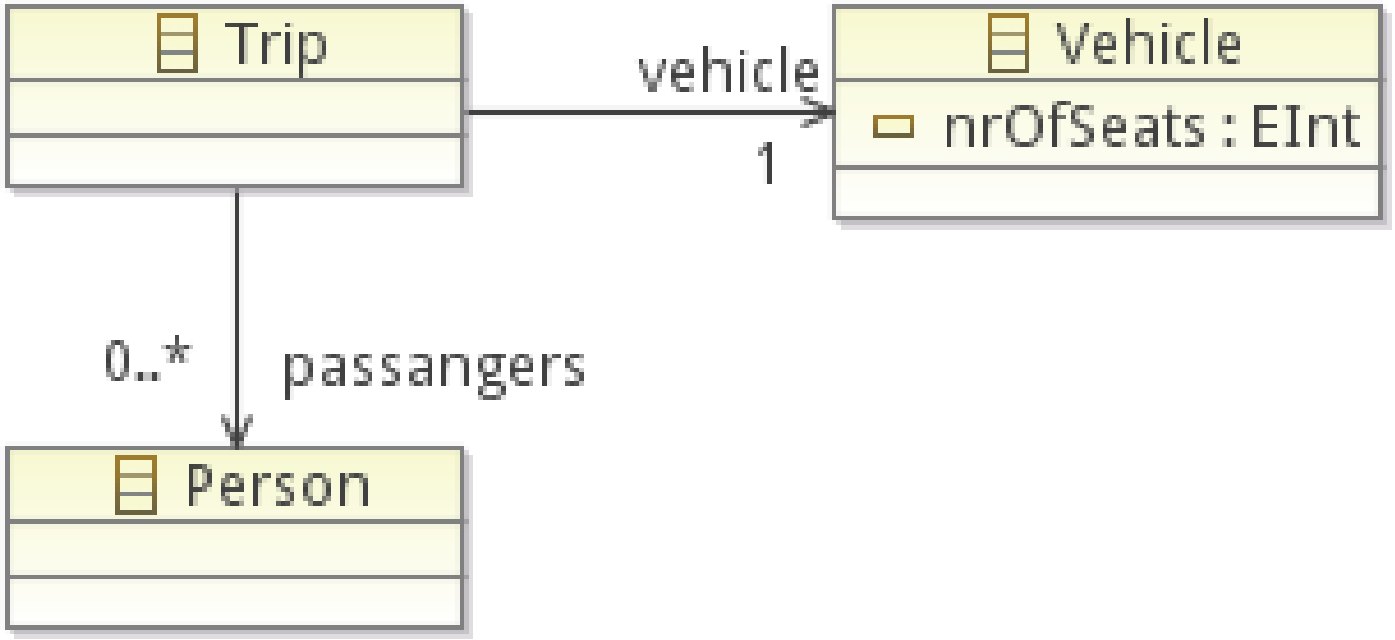
## Sample Constraint




```
context Trip
inv: passengers->size() <= vehicle.nrOfSeats
```

# OCL Syntax

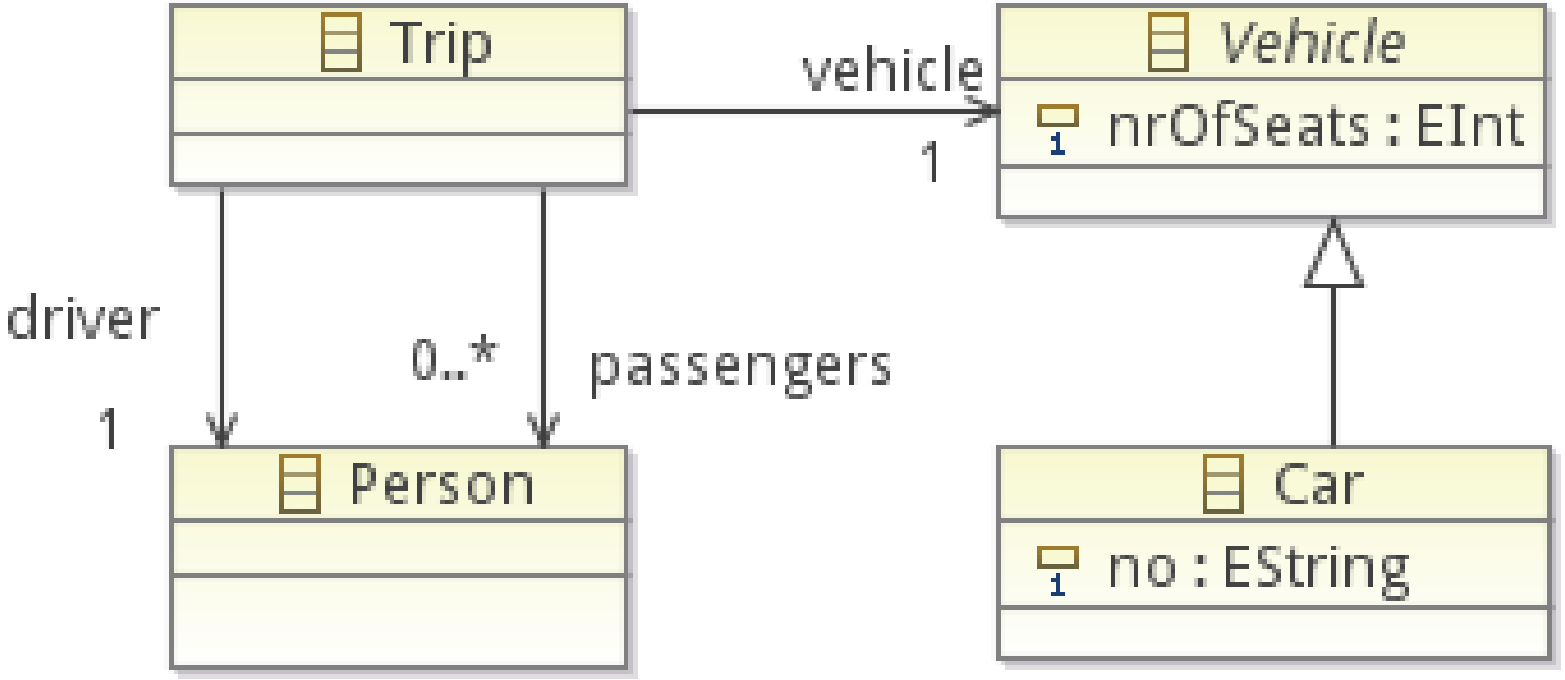
## Sample Constraint




```
context Trip
inv: passengers->size() <= vehicle.nrOfSeats
```

# OCL Syntax

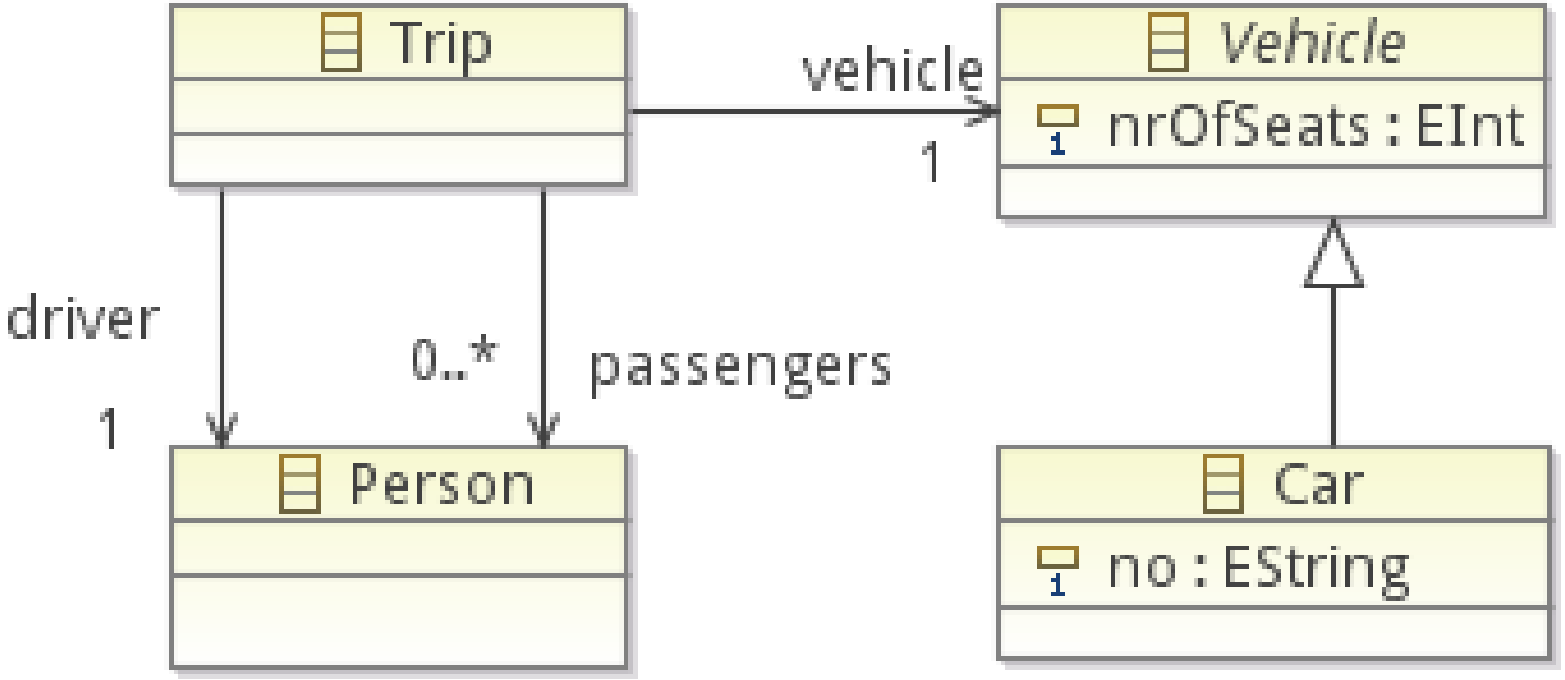
## Set operations




```
context Trip
inv: passengers->includes(driver)
```

# OCL Syntax

## Set operations




```
context Trip
inv: passengers->includes(driver)
```

# OCL Syntax

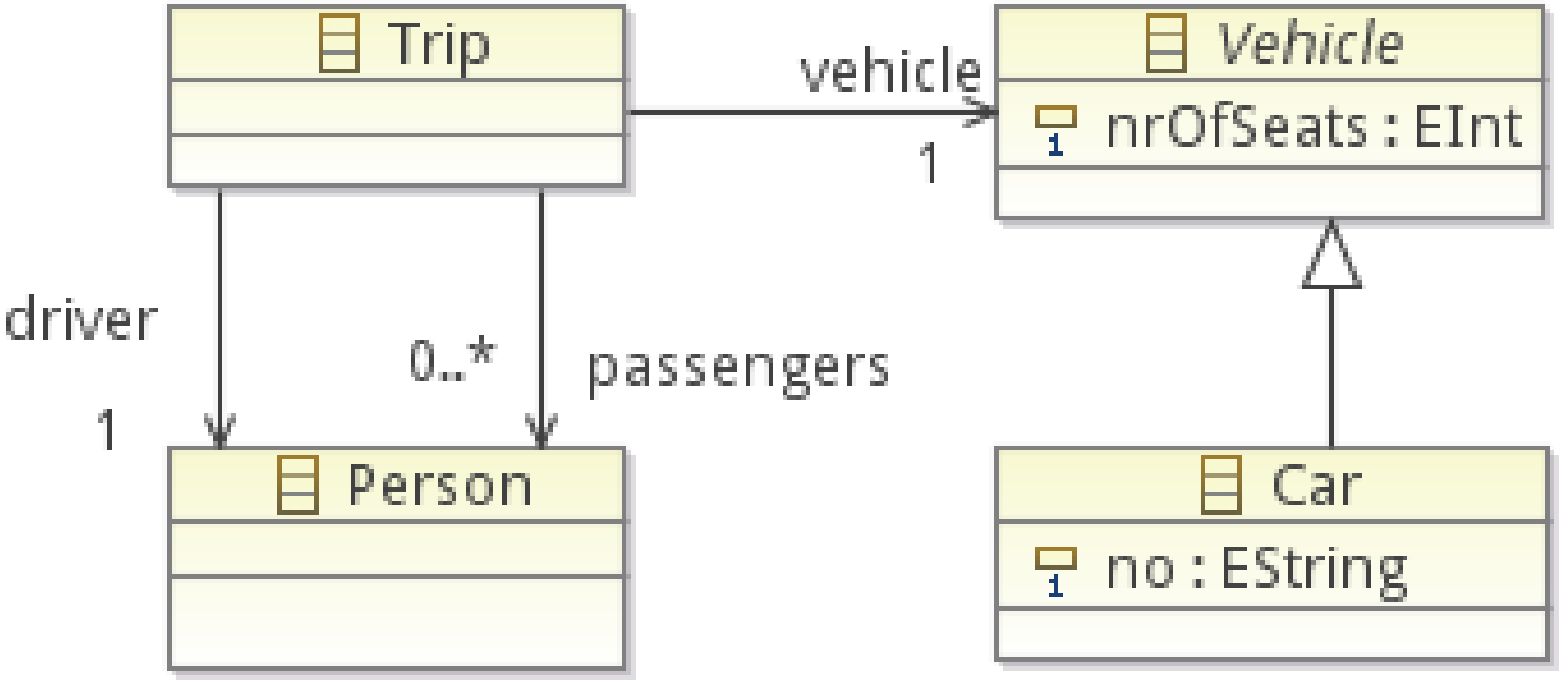
## self

```
context Trip
inv: passengers->includes(driver)

context Trip
inv: self.passengers->includes(self.driver)
```

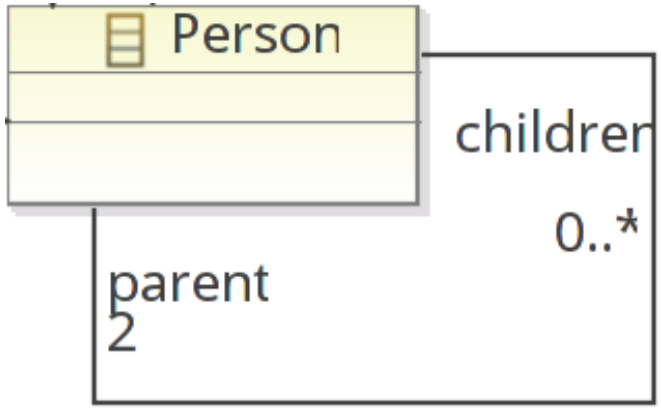
# OCL Syntax

instance quantification, uniqueness constraints




```
Context Car
inv: Car::allInstances()->isUnique(no)
```

# Cyclic Constraint, Commutativity

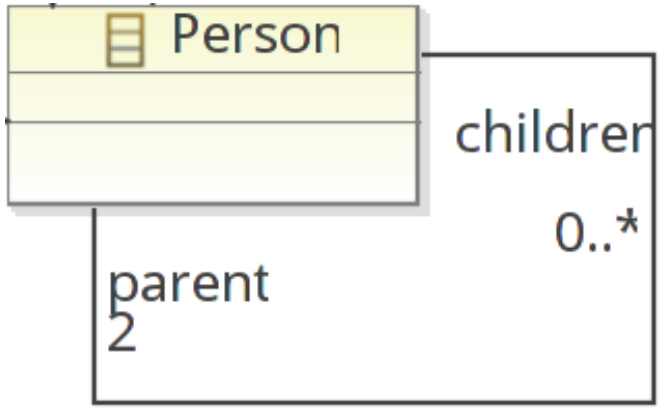



```
context Person
inv: self.children->
                forAll(c | c.parent->includes(self))

inv: self.parent->
                forAll(p | p.children->includes(self))

inv: not self.children->includes (self)
inv: not self.parent->includes (self)
```

# Cyclic Constraint, Commutativity

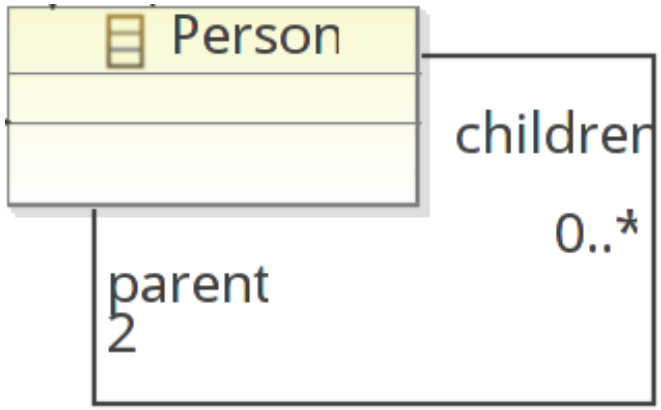



```
context Person
inv: self.children->
                forAll(c | c.parent->includes(self))

inv: self.parent->
                forAll(p | p.children->includes(self))

inv: not self.children->includes (self)
inv: not self.parent->includes (self)
```

# Cyclic Constraint, Commutativity

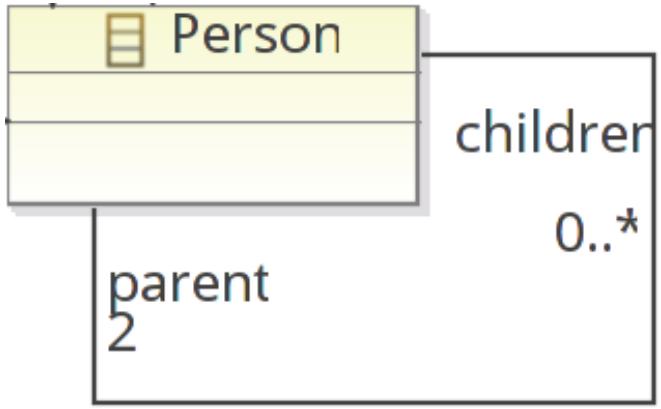



```
context Person
inv: self.children->
                forAll(c | c.parent->includes(self))

inv: self.parent->
                forAll(p | p.children->includes(self))

inv: not self.children->includes (self)
inv: not self.parent->includes (self)
```

# Cyclic Constraint, Commutativity




```
context Person
inv: self.children->
                forAll(c | c.parent->includes(self))

inv: self.parent->
                forAll(p | p.children->includes(self))

inv: not self.children->includes (self)
inv: not self.parent->includes (self)
```

- Introduction
- OCL Syntax and Semantics by Example
- **Crash Summary of OCL**
- OCL as a Domain Specific Language
- Playing with EMF and OCL Practically

# AGENDA

# OCL in a Nutshell

In OCL navigation works like in Java:

```
ClassName.attribute.relation.operation().attribute ...
```

Operation calls are allowed, but then you should be careful that the operations have no side effects.

# OCL in a Nutshell

Invariants can be named, to allow easier references:

```
context Trip
inv driverIsPassenger:  passengers.include(driver)
```

# OCL in a Nutshell

Referring to enumerations

`EnumerationClass::Literal`; for instance `Color::red`

# OCL in a Nutshell

Supported operators include: `implies and or xor not`, `if then else` (this is the same as `? :` in Java); `>= <= > < = <> + - / * a.mod(b) a.div(b)`
`a.abs() a.max(b) a.min(b) a.round() a.floor()`
`string.concat(string)`
`string.size() string.toLower() string.toUpper()`
`string.substring(int,int)`

# OCL in a Nutshell

If navigation arrives at more than one object (via a link with multiplicity exceeding 1), we obtain a collection as the value.

- `Course.students->size ()` — size of the collection class
- `Course.students->select (isGuest())-> size()`
  select the guest students, and count them
- `Course.students->select(isGuest())->isEmpty()`
  no guest students are allowed in the course.
- `Course.students->forAll(age >= 18)`
  this course is only for adult students
- `Course.students->forAll ( s | s.age >= 18)`
  equivalent to the above, but sometimes it is convenient, with a name for the iterated objects
- `Course.students->collect(age)`
  a collection of ages on this course.

# OCL in a Nutshell

If navigation arrives at more than one object (via a link with multiplicity exceeding 1), we obtain a collection as the value.

- `Course.students->size ()` — size of the collection class
- `Course.students->select (isGuest())-> size()`
  select the guest students, and count them
- `Course.students->select(isGuest())->isEmpty()`
  no guest students are allowed in the course.
- `Course.students->forAll(age >= 18)`
  this course is only for adult students
- `Course.students->forAll ( s | s.age >= 18)`
  equivalent to the above, but sometimes it is convenient, with a name for the iterated objects
- `Course.students->collect(age)`
  a collection of ages on this course.

# OCL in a Nutshell

If navigation arrives at more than one object (via a link with multiplicity exceeding 1), we obtain a collection as the value.

- `Course.students->size ()` — size of the collection class
- `Course.students->select (isGuest())-> size()`
  select the guest students, and count them
- `Course.students->select(isGuest())->isEmpty()`
  no guest students are allowed in the course.
- `Course.students->forAll(age >= 18)`
  this course is only for adult students
- `Course.students->forAll ( s | s.age >= 18)`
  equivalent to the above, but sometimes it is convenient, with a name for the iterated objects
- `Course.students->collect(age)`
  a collection of ages on this course.

# OCL in a Nutshell

If navigation arrives at more than one object (via a link with multiplicity exceeding 1), we obtain a collection as the value.

- `Course.students->size ()` — size of the collection class
- `Course.students->select (isGuest())-> size()`
  select the guest students, and count them
- `Course.students->select(isGuest())->isEmpty()`
  no guest students are allowed in the course.
- `Course.students->forAll(age >= 18)`
  this course is only for adult students
- `Course.students->forAll ( s | s.age >= 18)`
  equivalent to the above, but sometimes it is convenient, with a name for the iterated objects
- `Course.students->collect(age)`
  a collection of ages on this course.

# OCL in a Nutshell

If navigation arrives at more than one object (via a link with multiplicity exceeding 1), we obtain a collection as the value.

- `Course.students->size ()` — size of the collection class
- `Course.students->select (isGuest())-> size()`
  select the guest students, and count them
- `Course.students->select(isGuest())->isEmpty()`
  no guest students are allowed in the course.
- `Course.students->forAll(age >= 18)`
  this course is only for adult students
- `Course.students->forAll ( s | s.age >= 18)`
  equivalent to the above, but sometimes it is convenient, with a name for the iterated objects
- `Course.students->collect(age)`
  a collection of ages on this course.

# OCL in a Nutshell

If navigation arrives at more than one object (via a link with multiplicity exceeding 1), we obtain a collection as the value.

- `Course.students->size ()` — size of the collection class
- `Course.students->select (isGuest())-> size()`
  select the guest students, and count them
- `Course.students->select(isGuest())->isEmpty()`
  no guest students are allowed in the course.
- `Course.students->forAll(age >= 18)`
  this course is only for adult students
- `Course.students->forAll ( s | s.age >= 18)`
  equivalent to the above, but sometimes it is convenient, with a name for the iterated objects
- `Course.students->collect(age)`
  a collection of ages on this course.

# More on Collections

- Collection operators are introduced with an arrow (unlike in Java!)
- Other collection operators: `notEmpty`, `includes(object)`, `includesAll(collection)`, `union(collection)`
- Four types: sets, bags, ordered sets, sequences (an ordered bag)
- When you navigate through more than one association with multiplicity greater than 1 you end up with a bag.
- When you navigate just one such association you get a set
- You get an ordered-set or a sequence if any of these associations was marked as `ordered`

# More on Collections

- Collection operators are introduced with an arrow (unlike in Java!)
- Other collection operators:
  `notEmpty`, `includes(object)`, `includesAll(collection)`, `union(collection)`
- Four types: sets, bags, ordered sets, sequences (an ordered bag)
- When you navigate through more than one association with multiplicity greater than 1 you end up with a bag.
- When you navigate just one such association you get a set
- You get an ordered-set or a sequence if any of these associations was marked as `ordered`

# More on Collections

- Collection operators are introduced with an arrow (unlike in Java!)
- Other collection operators:
  `notEmpty`, `includes(object)`, `includesAll(collection)`, `union(collection)`
- Four types: sets, bags, ordered sets, sequences (an ordered bag)
- When you navigate through more than one association with multiplicity greater than 1 you end up with a bag.
- When you navigate just one such association you get a set
- You get an ordered-set or a sequence if any of these associations was marked as `ordered`

# More on Collections

- Collection operators are introduced with an arrow (unlike in Java!)
- Other collection operators:
  `notEmpty`, `includes(object)`, `includesAll(collection)`, `union(collection)`
- Four types: sets, bags, ordered sets, sequences (an ordered bag)
- When you navigate through more than one association with multiplicity greater than 1 you end up with a bag.
- When you navigate just one such association you get a set
- You get an ordered-set or a sequence if any of these associations was marked as `ordered`

# More on Collections

- Collection operators are introduced with an arrow (unlike in Java!)
- Other collection operators:
  `notEmpty`, `includes(object)`, `includesAll(collection)`, `union(collection)`
- Four types: sets, bags, ordered sets, sequences (an ordered bag)
- When you navigate through more than one association with multiplicity greater than 1 you end up with a bag.
- When you navigate just one such association you get a set
- You get an ordered-set or a sequence if any of these associations was marked as `ordered`

# More on Collections

- Collection operators are introduced with an arrow (unlike in Java!)
- Other collection operators:
  `notEmpty`, `includes(object)`, `includesAll(collection)`, `union(collection)`
- Four types: sets, bags, ordered sets, sequences (an ordered bag)
- When you navigate through more than one association with multiplicity greater than 1 you end up with a bag.
- When you navigate just one such association you get a set
- You get an ordered-set or a sequence if any of these associations was marked as `ordered`

# OCL in a Nutshell

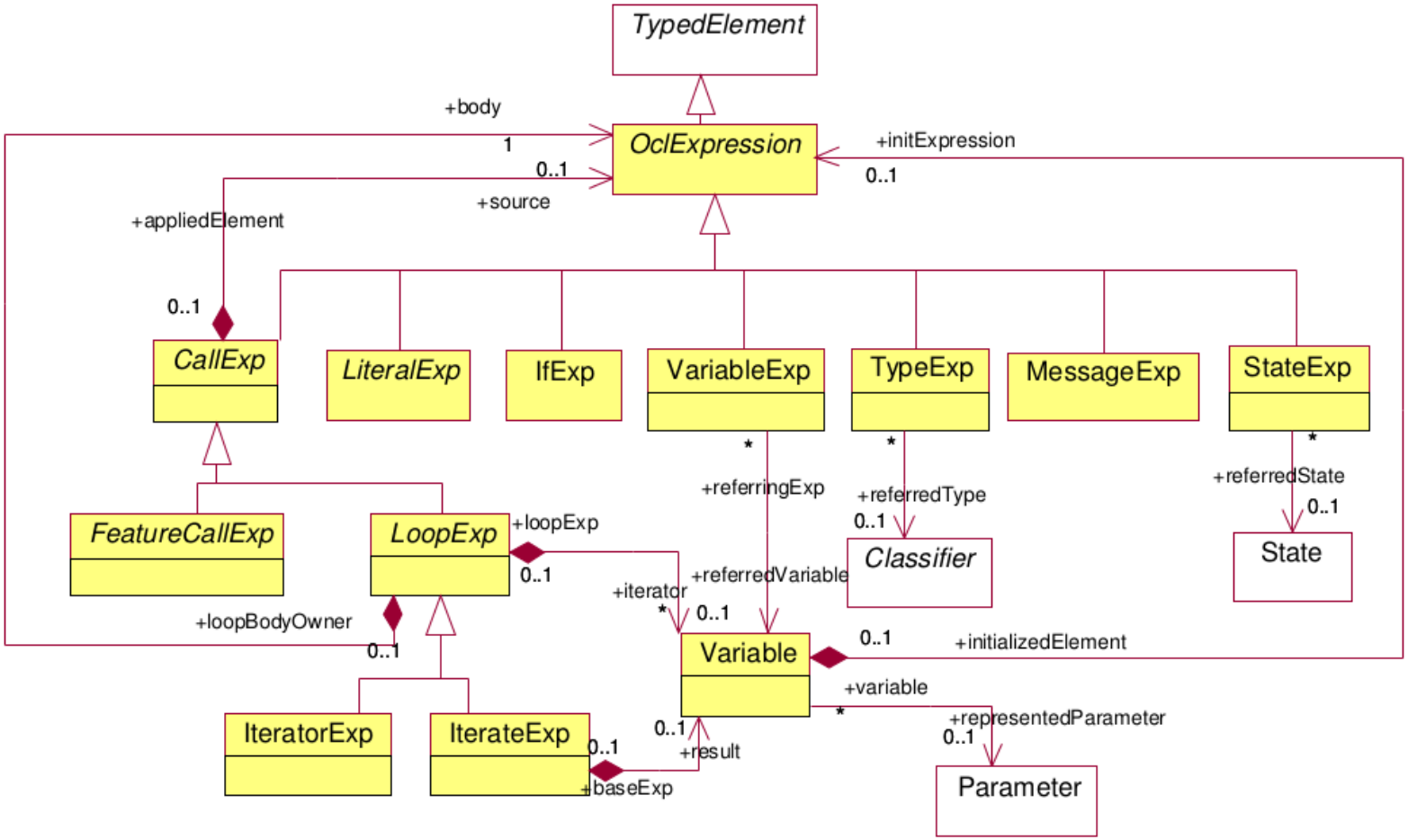
Let expressions:

```
let guests = Course.students-select(isGuest())
in guests->forAll( age >=18 )
                        and guests->forAll( age <= 20 )
```

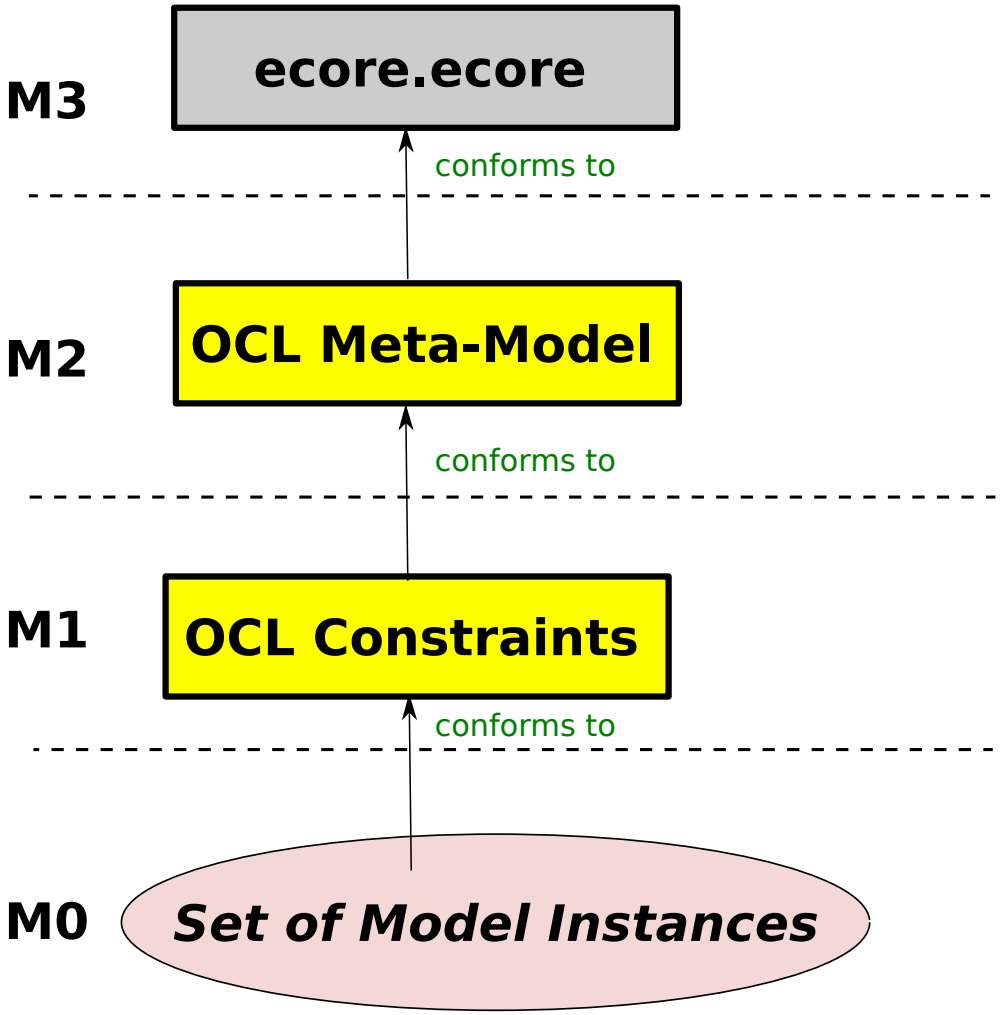
- Introduction
- OCL Syntax and Semantics by Example
- Crash Summary of OCL
- **OCL as a Domain Specific Language**
- Playing with EMF and OCL, Practically

# AGENDA

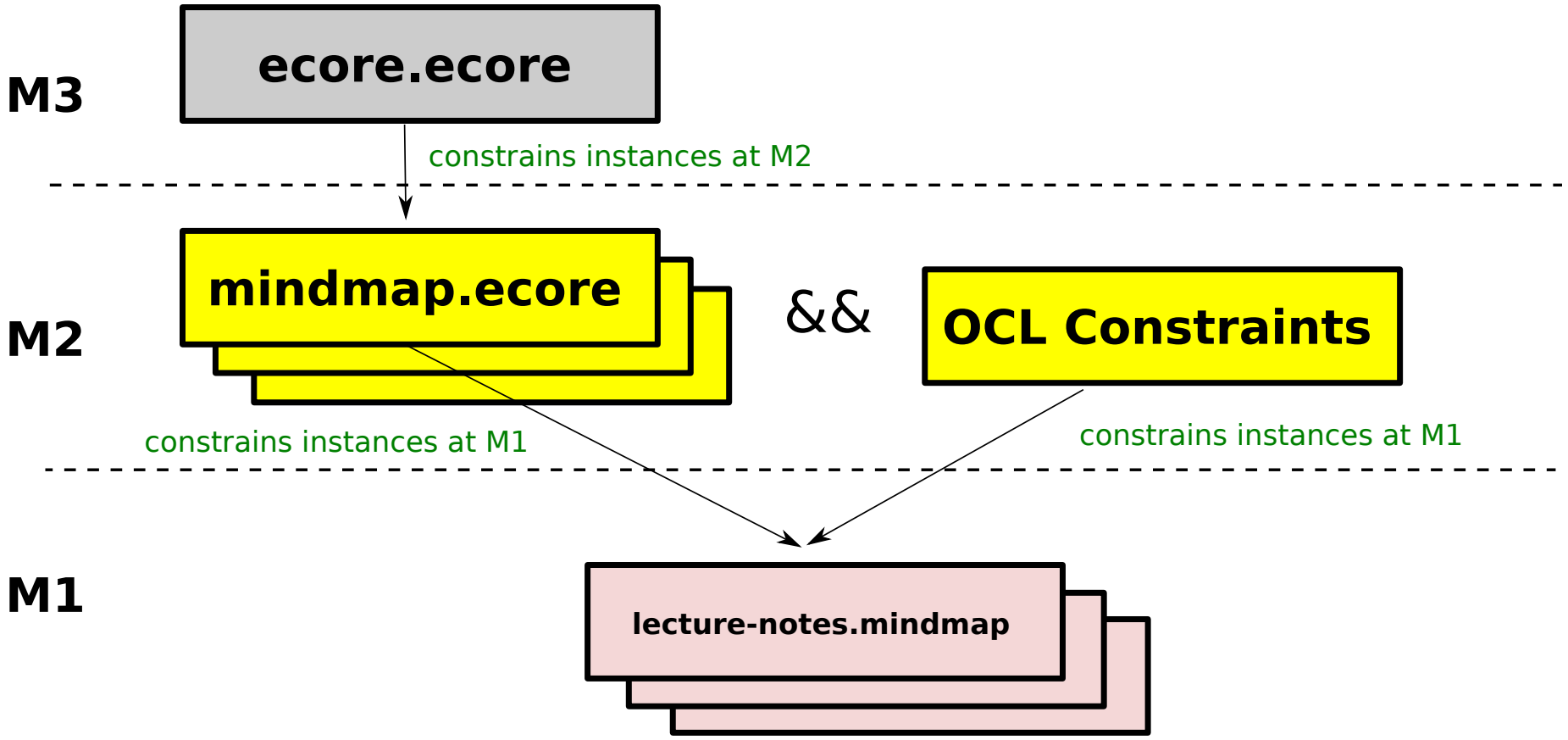
# Core OCL Meta-model



# OCL in the Meta-modeling Hierarchy



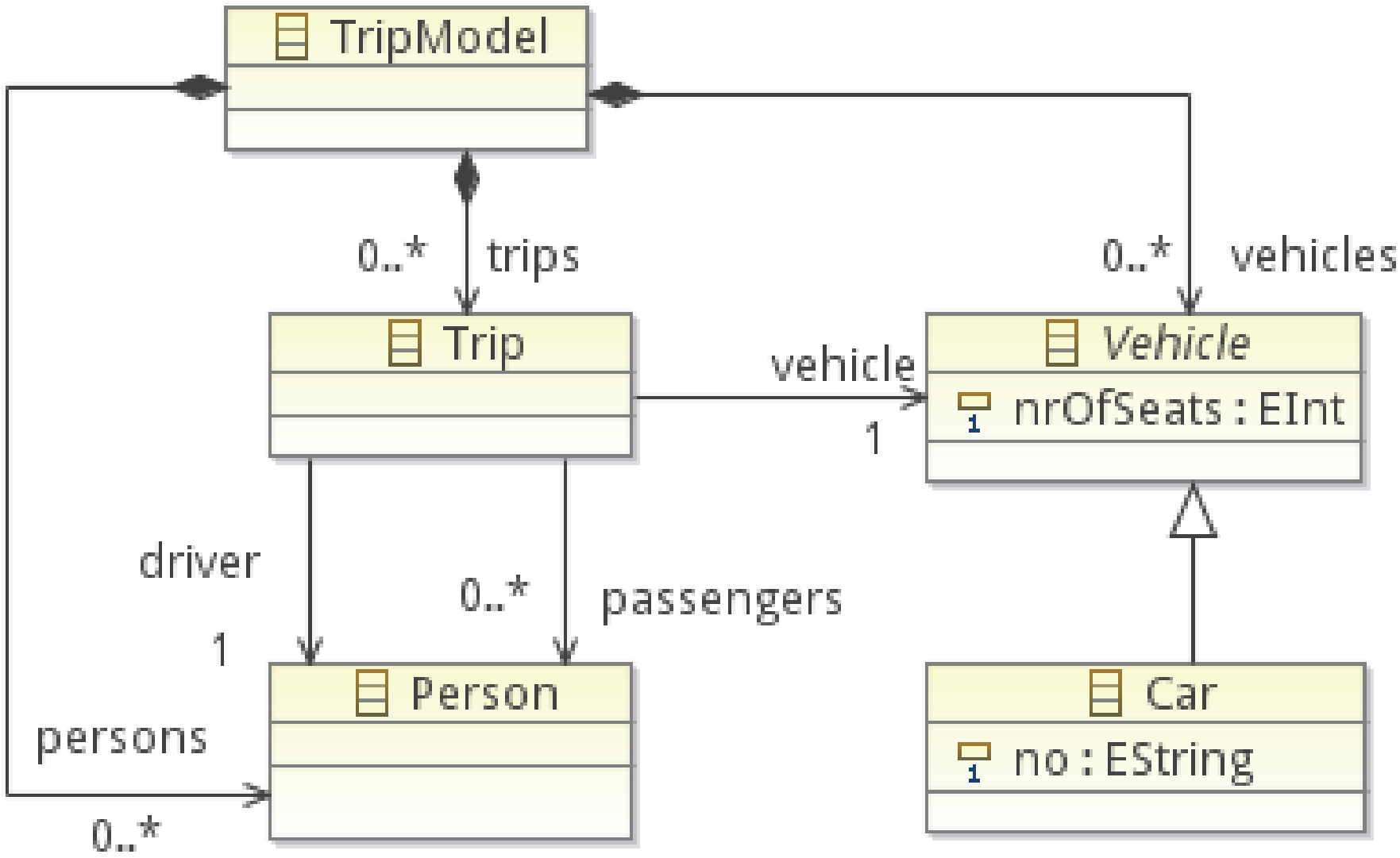
# OCL Semantics and the Hierarchy



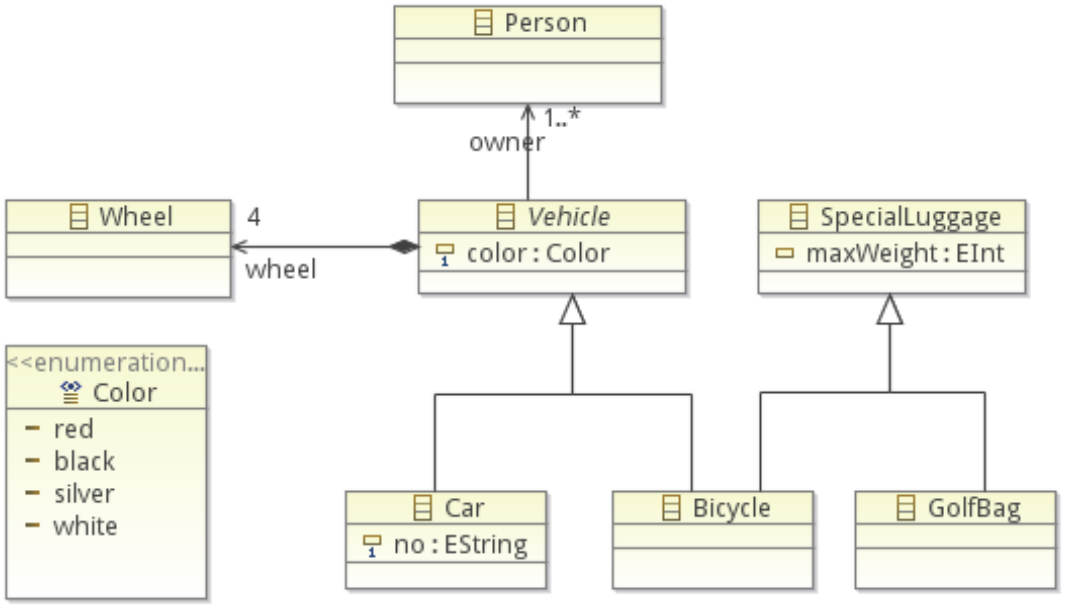
- Introduction
- OCL Syntax and Semantics by Example
- Crash Summary of OCL
- OCL as a Domain Specific Language
- **Playing with EMF and OCL, Practically**

# AGENDA

# Model Class in EMF



# Tree View of An EMF Model




- GeneralizationPackage
  - Car -> Vehicle
    - Vehicle
    - no : EString
  - Vehicle
    - color : Color
    - owner : Person
    - wheel : Wheel
  - Bicycle -> Vehicle, SpecialLuggage
    - Vehicle
    - SpecialLuggage
  - SpecialLuggage
    - maxWeight : EInt
  - GolfBag -> SpecialLuggage
    - SpecialLuggage
  - Color
    - red = 0
    - black = 1
    - silver = 2
    - white = 3
  - Person
    - ERefeence0 : Vehicle
  - Wheel

# We will use OCL in exercises next week

- To play now take or make an ecore model
- Derive a dynamic instance of the model (right click on a class)
- Give a file name in which the instance should be stored.
- Editor opens and you can start adding children to the created node.
- Values for attributes and references can be specified in the properties view (right click)
- 'Validate' in context-menu checks if multiplicity constraints of the meta-model are satisfied by the instance.
- 'OCL console' available in the context menu of instance elements
- In the OCL console, do not write `inv` and contexts
- Context is the selected element in the editor
- Constraints can be shown at M1 and M2

# the end of episode 4

Frank Budinsky, David Steinber, Ed Merks, Raymond Ellersick, and Timothy J. Groose.
*Eclipse Modeling Framework*.
Addison-Wesley, 2004.