

Model-Based Software Engineering

Lecture 04 – OCL and Concrete Syntax

Prof. Dr. Joel Greenyer



April 26, 2016



Acknowledgment

- The slides of this lecture are inspired by lecture slides from
 - *Ekkart Kindler*: Course on Advanced Topics in Software Engineering, DTU Compute, 2015.
 - <http://www2.imm.dtu.dk/courses/02265/f15/schedule.shtml>
 - *Ina Schäfer, Christoph Seidl*: Modellbasierte Softwareentwicklung, TU Braunschweig, 2015.
 - *Steffen Becker*: Model-Driven Software Development, Universität Paderborn, 2013
 - The Eclipse Open Model CourseWare (OMCW) Project:
 - <https://eclipse.org/gmt/omcw/>

```

classDiagram
    class ETypedElement {
        ordered : boolean = true
        unique : boolean = true
        lowerBound : int
        upperBound : int = 1
        many : boolean
        required : boolean
    }
    class EClassifier {
        instanceClassName : String
        instanceClass : EJavaClass
        defaultValue : EJavaObject
        isInstance(object : EJavaObject) : boolean
        getClassifierID() : int
    }
    class EPackage {
        nsURI : String
        nsPrefix : String
        getEClassifier(name : String) : EClassifier
    }
    class EClass {
        abstract : boolean
        interface : boolean
        isSuperTypeOf(someClass : EClass) : boolean
        getEStructuralFeature(featureID : int) : EStructuralFeature
        getEStructuralFeature(featureName : String) : EStructuralFeature
    }
    class EDataType {
        serializable : boolean = true
    }
    class EStructuralFeature {
        changeable : boolean = true
        volatile : boolean
        transient : boolean
        defaultValueLiteral : String
        defaultValue : EJavaObject
        unsettable : boolean
        derived : boolean
        getFeatureID() : int
        getContainerClass() : EJavaClass
    }
    class EReference {
        containment : boolean
        container : boolean
        resolveProxies : boolean = true
    }
    class EAttribute {
        iD : boolean
    }
    class EEnum {
        getEEnumLiteral(name : String) : EEnumLiteral
        getEEnumLiteral(value : int) : EEnumLiteral
    }
    class EEnumLiteral {
        value : int
        instance : EEnumerator
    }
    class EOperation {
    }
    class EParameter {
    }

    ETypedElement <|-- EOperation
    ETypedElement <|-- EParameter
    ETypedElement --> EClassifier : +eType 0..1
    EClassifier <|-- EPackage
    EClassifier <|-- EClass
    EClassifier <|-- EDataType
    EPackage --> EClassifier : +ePackage
    EPackage --> EClassifier : +eSubpackages 0..*
    EPackage --> EClassifier : +eSuperPackage
    EClass --> EClassifier : +eClassifiers 0..*
    EClass --> EClass : +eOperations
    EClass --> EClass : +eParameters 0..*
    EClass --> EClass : +eContainingClass 0..*
    EClass --> EStructuralFeature : +eStructuralFeatures 0..*
    EClass --> EClass : +eSuperTypes 0..*
    EClass --> EClass : +eContainingClass 0..*
    EStructuralFeature <|-- EReference
    EStructuralFeature <|-- EAttribute
    EReference --> EReference : +eOpposite 0..1
    EEnum --> EDataType
    EEnum --> EEnumLiteral : +eLiterals 0..*
    EEnum --> EEnum : +eEnum
    
```

The diagram illustrates the Ecore metamodel classes and their relationships. The classes are organized into several groups: **Base and Classifier Classes** (ETypedElement, EClassifier, EPackage, EClass, EDataType), **Structural Feature Classes** (EStructuralFeature, EReference, EAttribute), and **Enumeration Classes** (EEnum, EEnumLiteral). Relationships include inheritance (e.g., EClass inherits from EClassifier), associations (e.g., EPackage to EClassifier), and composition (e.g., EClass to EStructuralFeature). Multiplicities and directionality are indicated on the association lines.

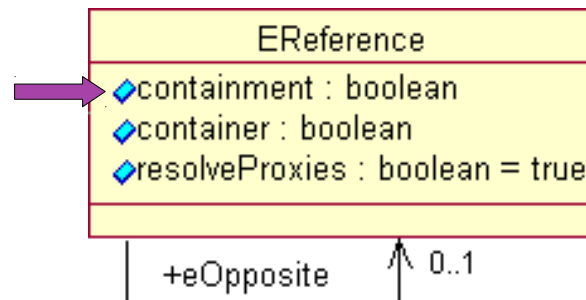
Specialties of EReferences

in the last lecture...

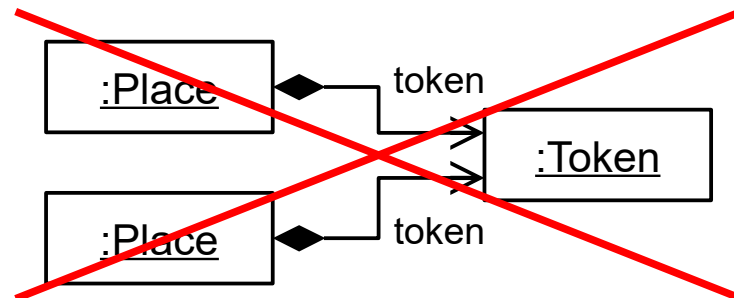
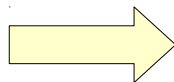
- **Containment:**

- An object can only be contained in *at most one* other object at a time

- it can be target of at most one containment link at a time



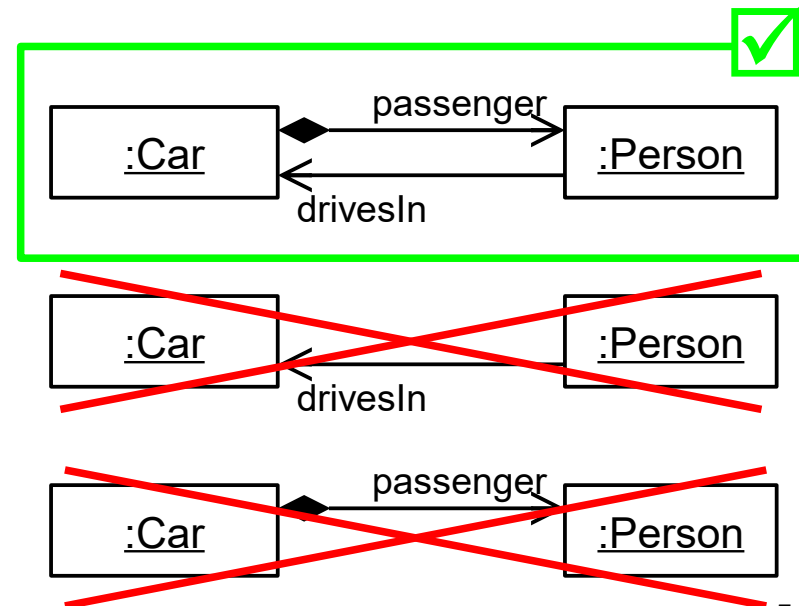
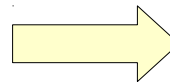
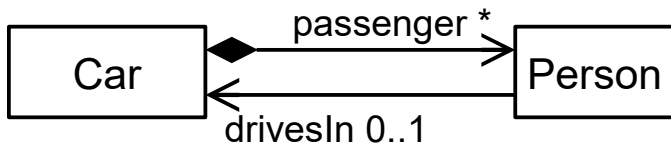
- **Example:**



Specialties of EReferences

in the last lecture...

- **eOpposite:**
 - Two EReferences in opposite directions between two EClasses can be “opposites”
 - Thereby forming a bidirectional relationship
 - At the object level, there must be bidirectional links
- Example:



in the last lecture...

- Separation of Interfaces and Implementation

```
public class PlaceImpl extends NodeImpl implements Place {

    protected static final int INTIAL_MARKING_EDEFAULT = 0;
    protected int intialMarking = INTIAL_MARKING_EDEFAULT;

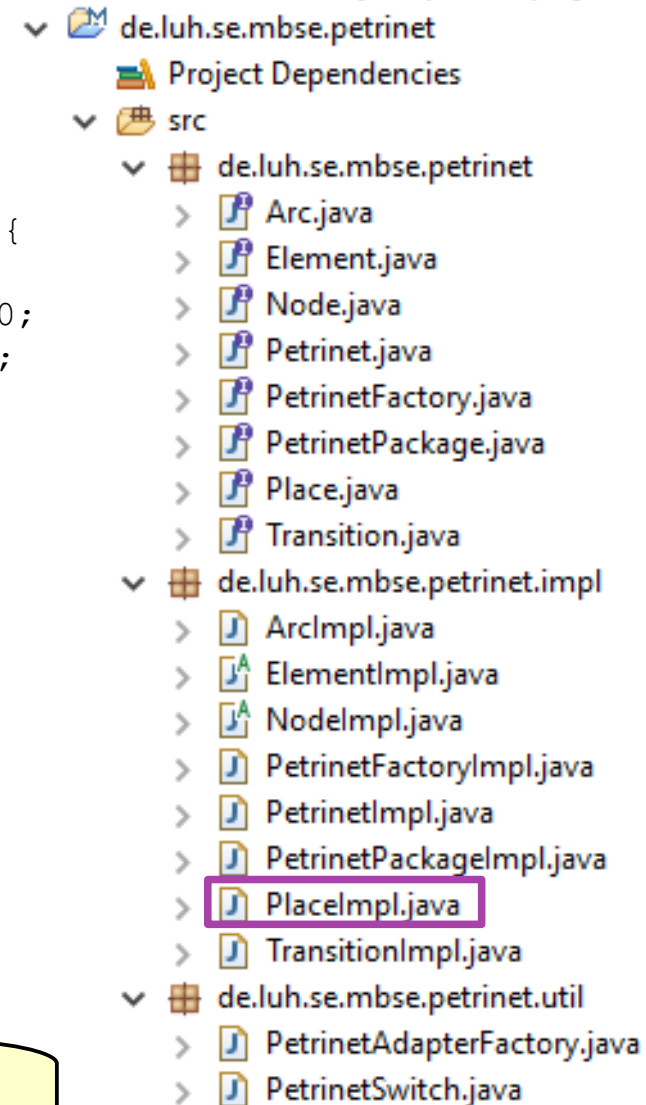
    public int getIntialMarking() {
        return intialMarking;
    }

    public void setIntialMarking(int newIntialMarking) {
        int oldIntialMarking = intialMarking;
        intialMarking = newIntialMarking;
        if (eNotificationRequired())
            eNotify(new ENotificationImpl(this,
                Notification.SET,
                PetrinetPackage.PLACE__INTIAL_MARKING,
                oldIntialMarking,
                intialMarking));
    }

    ...

} //PlaceImpl
```

Notification mechanism
(observer pattern) built in



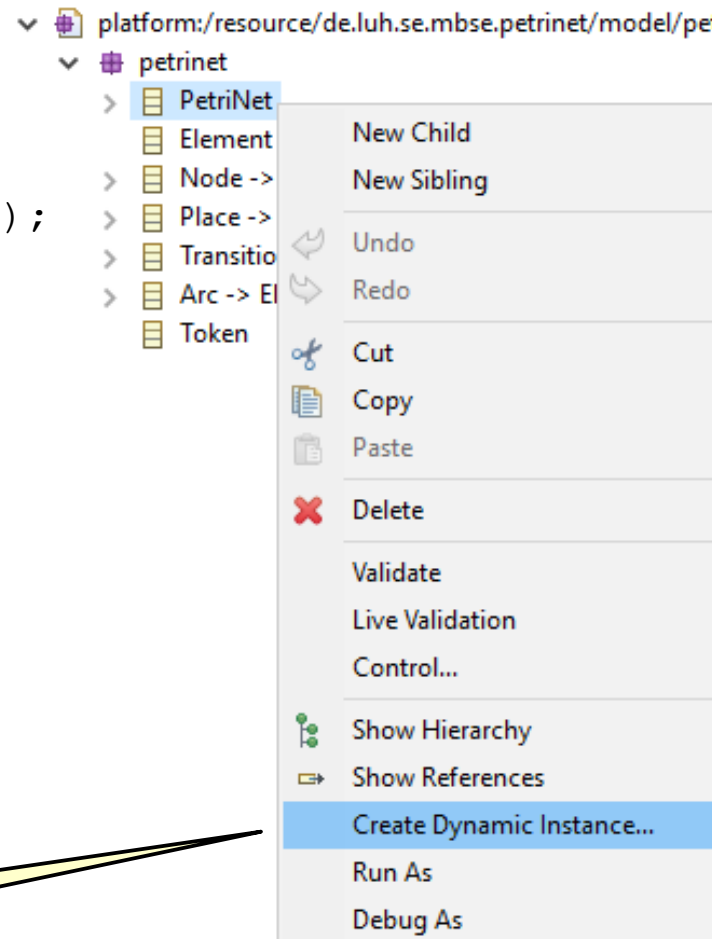
- The generated code allows us to create instances of our Ecore models
 - for example (factory method):

```
public Petrinet createPetrinet() {  
    PetrinetImpl petrinet = new PetrinetImpl();  
    return petrinet;  
}
```

- The generated code allows us to create instances of our Ecore models
 - for example (factory method):

```
public Petrinet createPetrinet() {
    PetrinetImpl petrinet = new PetrinetImpl();
    return petrinet;
}
```

- But EMF also supports working with **dynamic instances**
- EMF **interprets** the metamodels to allow us to work on instance models without code generation:



creating a dynamic object
via the UI

Dynamic EMF

- creating models and instances dynamically via the API:

- creating models and instances dynamically via the API:

```
// create package  
EPackage petrinetPackage = EcoreFactory.eINSTANCE.createEPackage();
```

- creating models and instances dynamically via the API:

```
// create package
EPackage petrinetPackage = EcoreFactory.eINSTANCE.createEPackage();

//create Place class
EClass placeClass = EcoreFactory.eINSTANCE.createEClass();
placeClass.setName("Place");
petrinetPackage.getEClassifiers().add(placeClass);
```

- creating models and instances dynamically via the API:

```
// create package
EPackage petrinetPackage = EcoreFactory.eINSTANCE.createEPackage();

//create Place class
EClass placeClass = EcoreFactory.eINSTANCE.createEClass();
placeClass.setName("Place");
petrinetPackage.getEClassifiers().add(placeClass);

//create initialMarkings attribute and add it to the Place class
EAttribute initialMarkingsAttribute
    = EcoreFactory.eINSTANCE.createEAttribute();
initialMarkingsAttribute.setName("initialMarkings");
initialMarkingsAttribute.setEType(EcorePackage.eINSTANCE.getEInt());
placeClass.getEAttributes().add(initialMarkingsAttribute);
```

- creating models and instances dynamically via the API:

```
// create package
EPackage petrinetPackage = EcoreFactory.eINSTANCE.createEPackage();

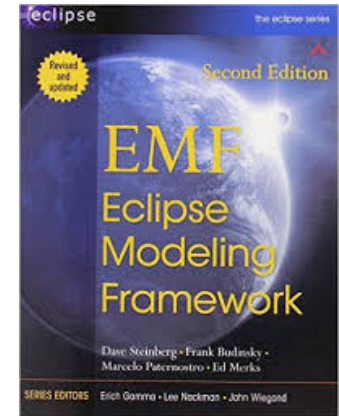
//create Place class
EClass placeClass = EcoreFactory.eINSTANCE.createEClass();
placeClass.setName("Place");
petrinetPackage.getEClassifiers().add(placeClass);

//create initialMarkings attribute and add it to the Place class
EAttribute initialMarkingsAttribute
    = EcoreFactory.eINSTANCE.createEAttribute();
initialMarkingsAttribute.setName("initialMarkings");
initialMarkingsAttribute.setEType(EcorePackage.eINSTANCE.getEInt());
placeClass.getEAttributes().add(initialMarkingsAttribute);

//create dynamic instance of Place class
EFactory petrinetFactory = petrinetPackage.getEFactoryInstance();
EObject place = petrinetFactory.create(placeClass);
place.eSet(initialMarkingsAttribute, 2);
```

EMF Resources

- D. Steinberg, F. Budinski, M. Paternostro, E. Merks: EMF: Eclipse Modeling Framework, Addison Wesley, 2nd edition, 2008.



- Online resources
 - <http://www.vogella.com/tutorials/EclipseEMF/article.html>
 - <http://eclipsesource.com/blogs/tutorials/emf-tutorial/>
 - There are many more online resources...

Model-Based Software Engineering

Lecture 04 – OCL and Concrete Syntax

Prof. Dr. Joel Greenyer

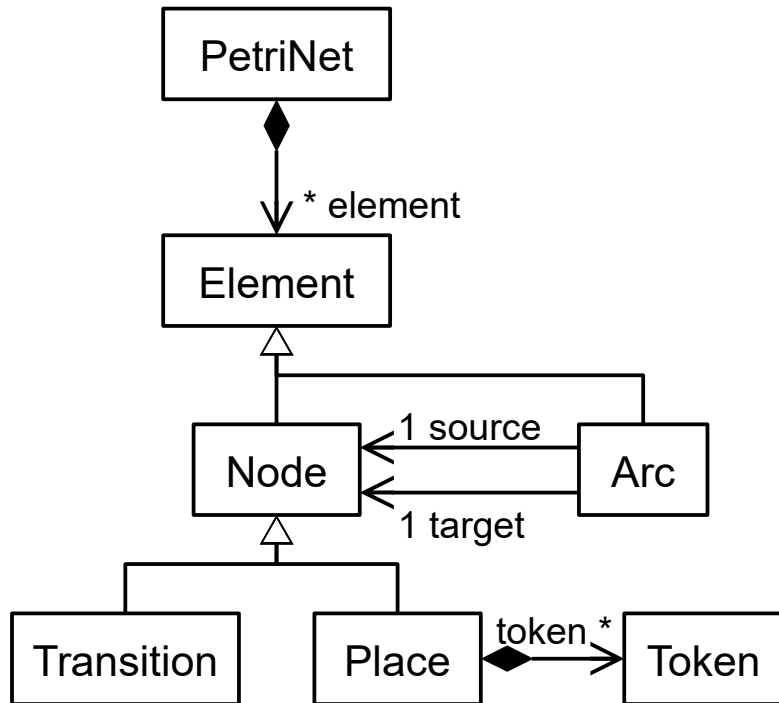


April 26, 2016

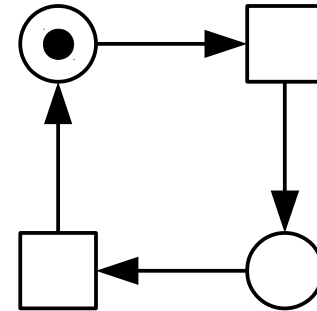
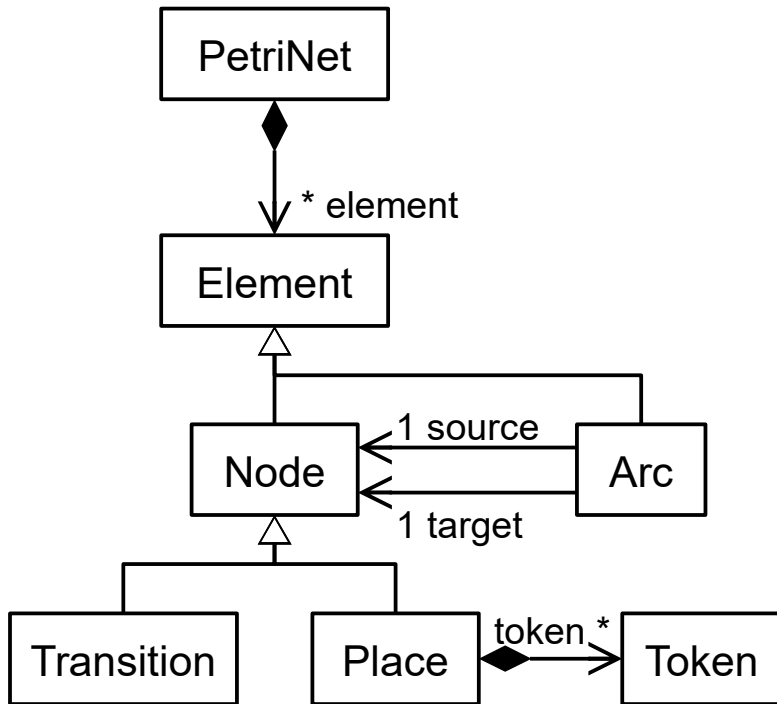


3.1. Introduction to OCL

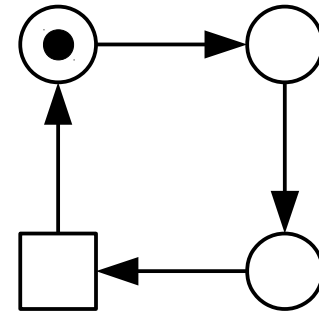
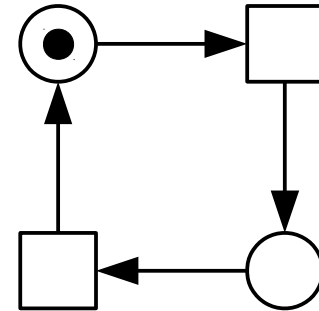
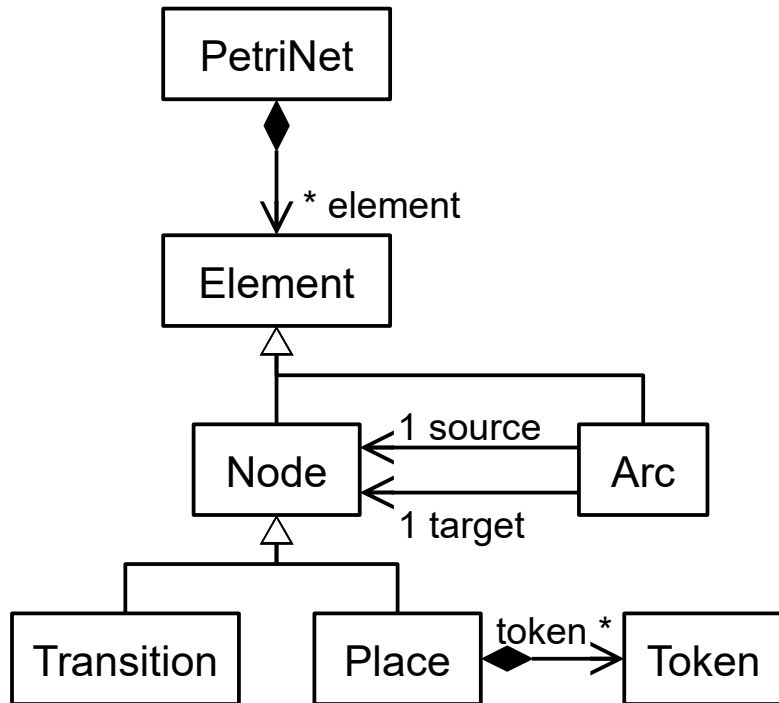
What's missing?



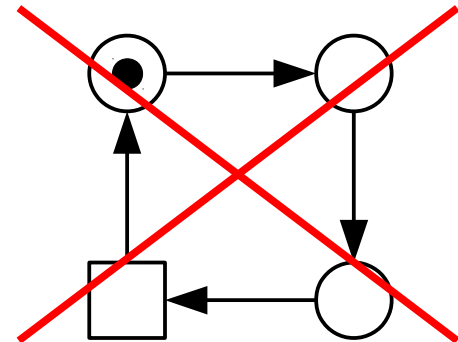
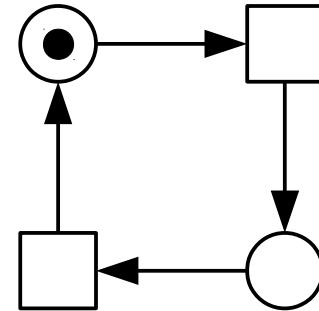
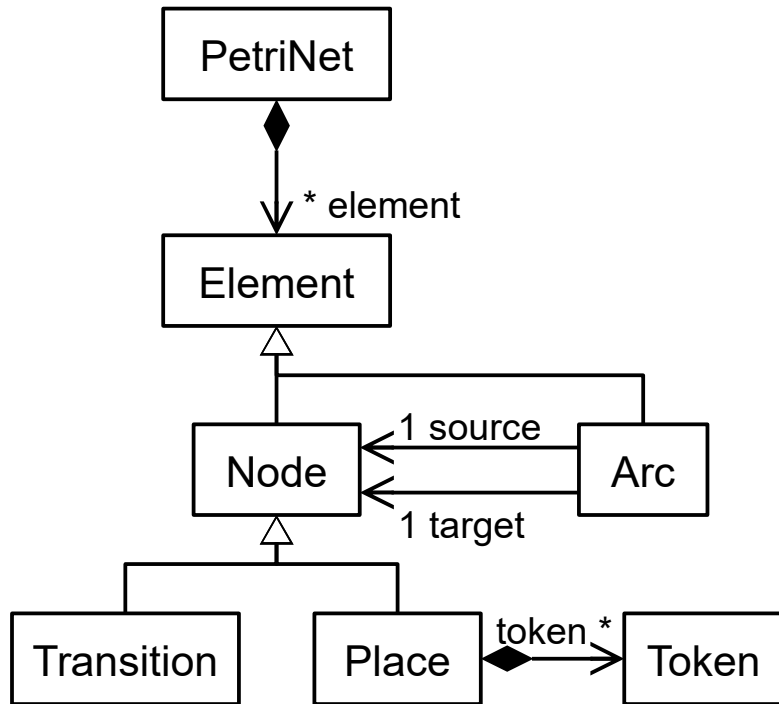
What's missing?



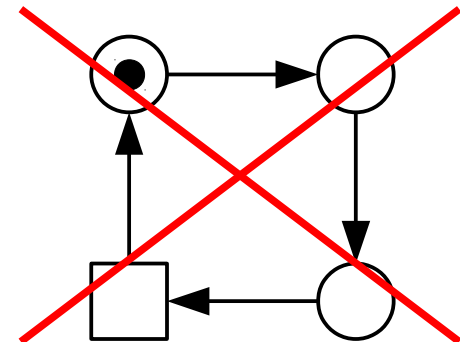
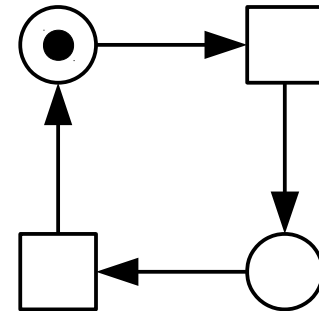
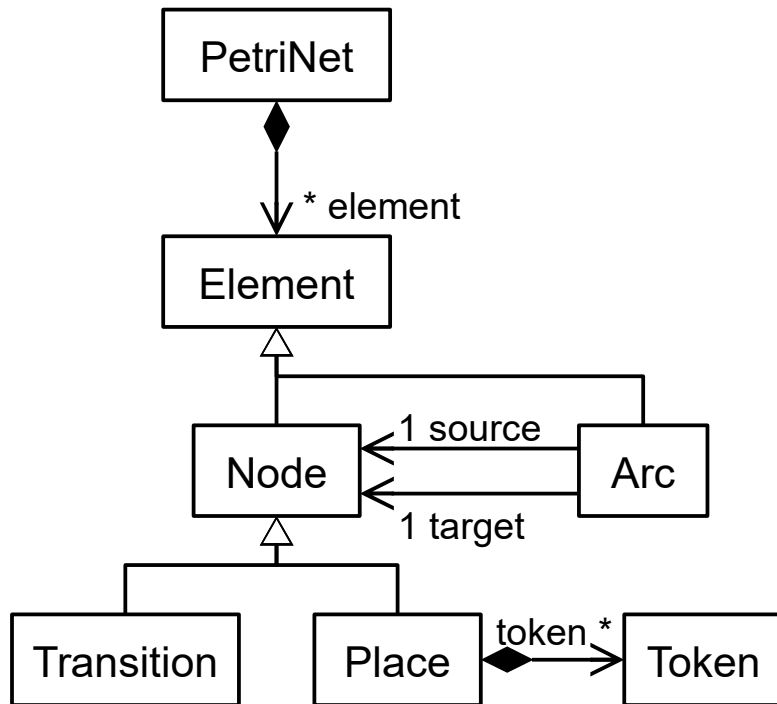
What's missing?



What's missing?

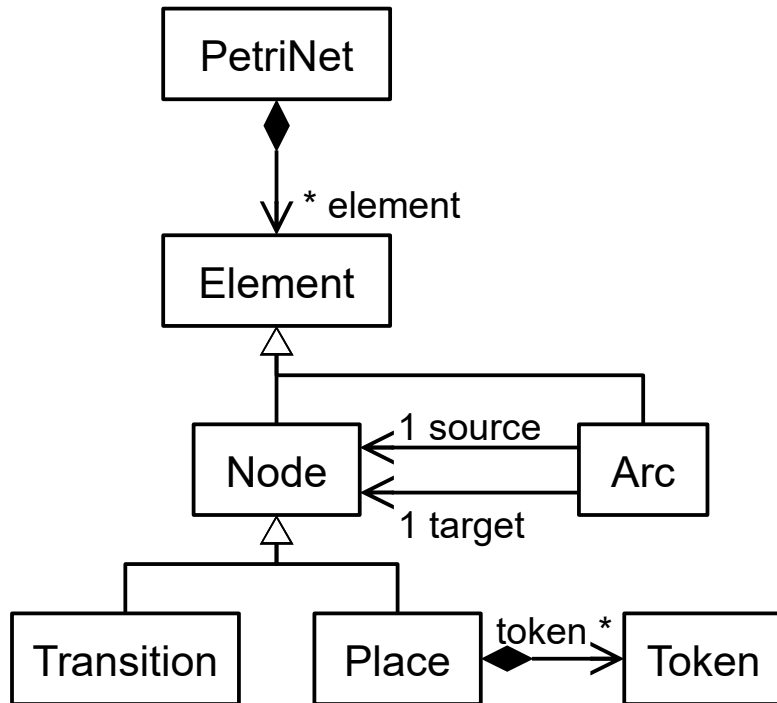


What's missing?

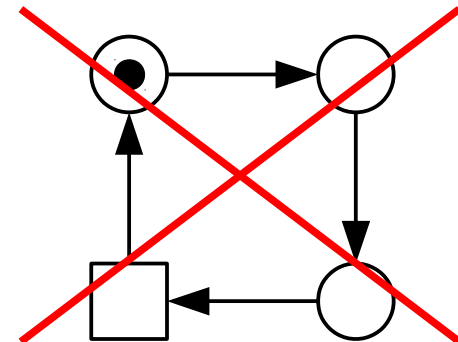
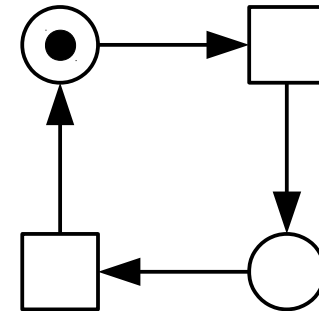


That is not a valid Petri net!: An Arc must only connect Places to Transitions or Transitions to Places

What's missing?

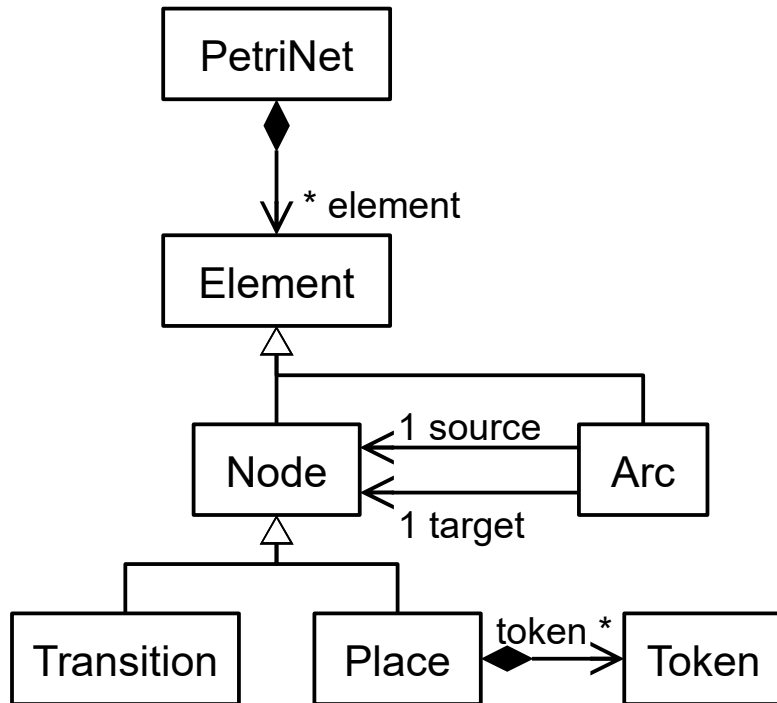


Obviously something is missing in the Petri net metamodel!

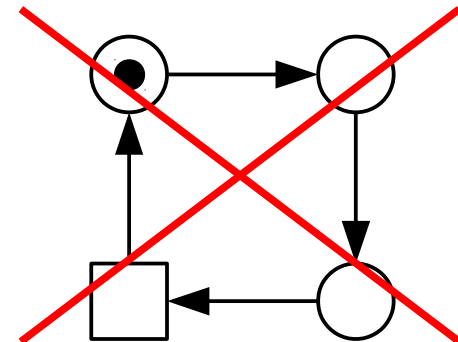
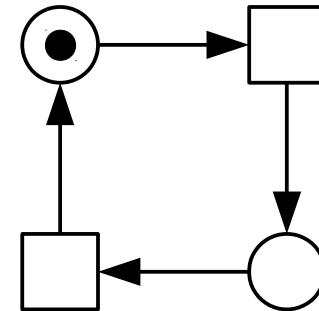


That is not a valid Petri net! An Arc must only connect Places to Transitions or Transitions for Places

What's missing?

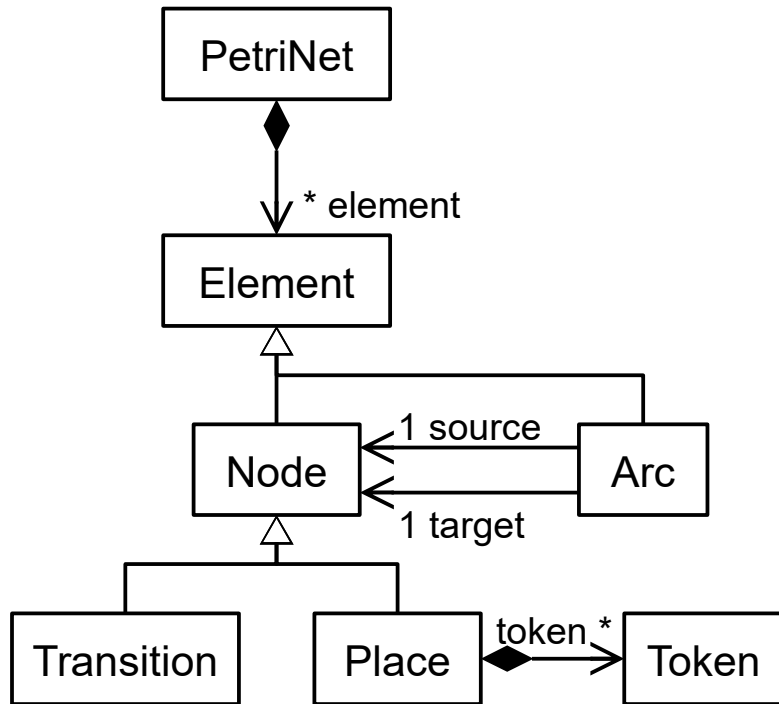


Obviously something is missing in the Petri net metamodel!

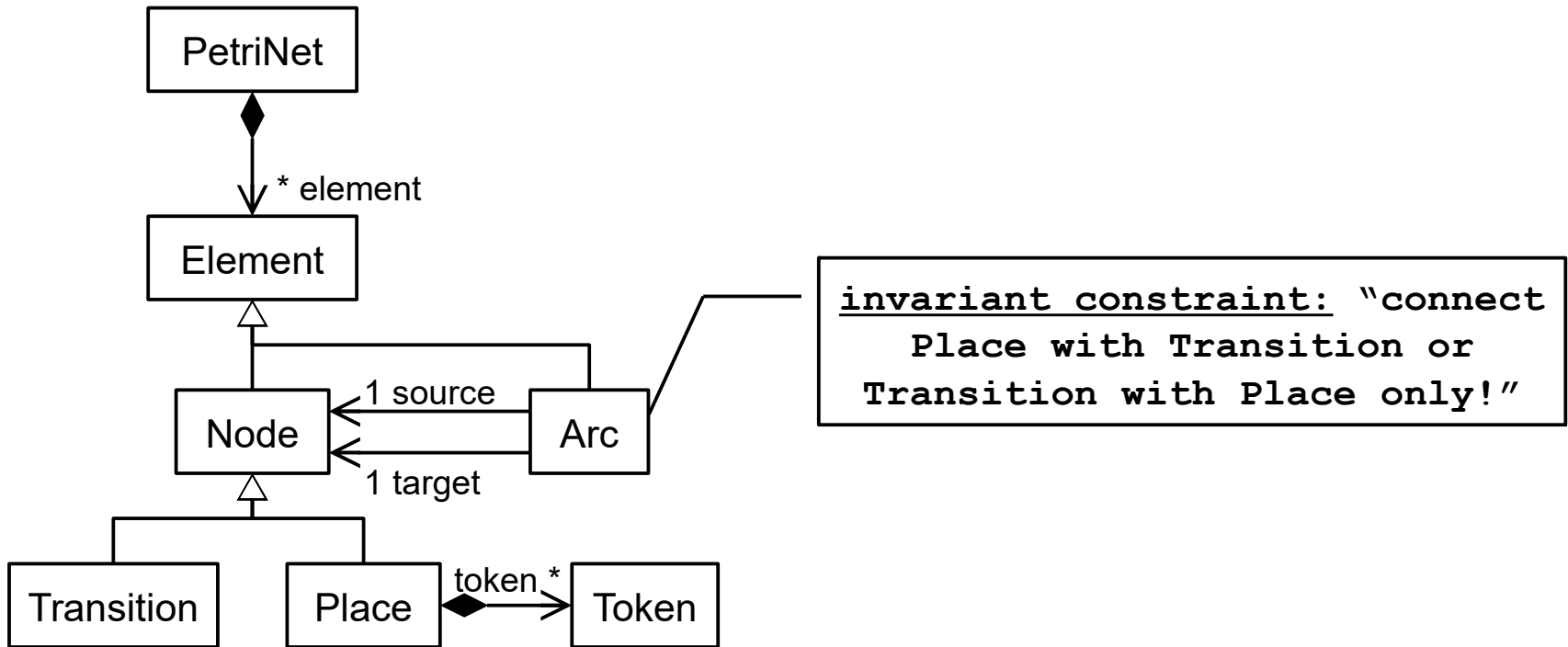


That is not a valid Petri net!: An Arc must only connect Places to Transitions or Transitions for Places

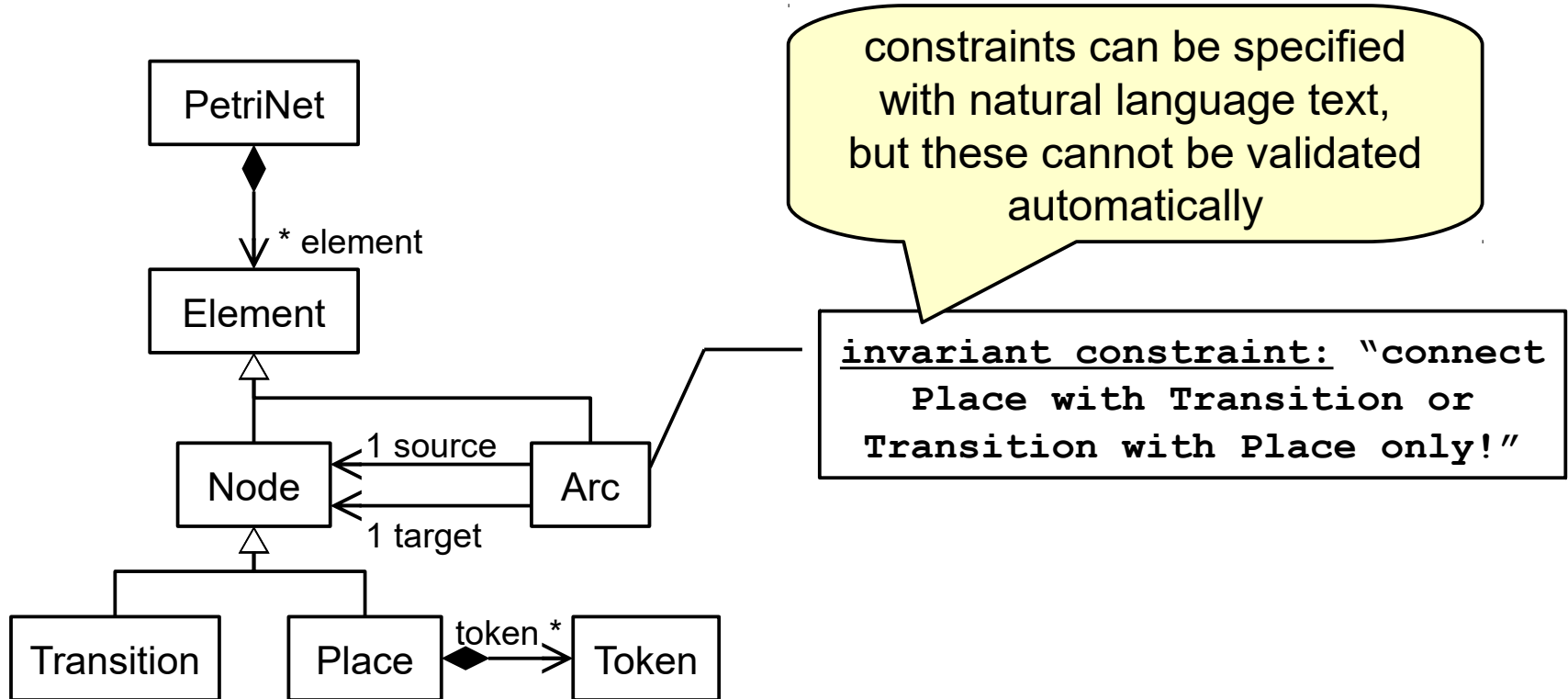
What's missing?



What's missing?



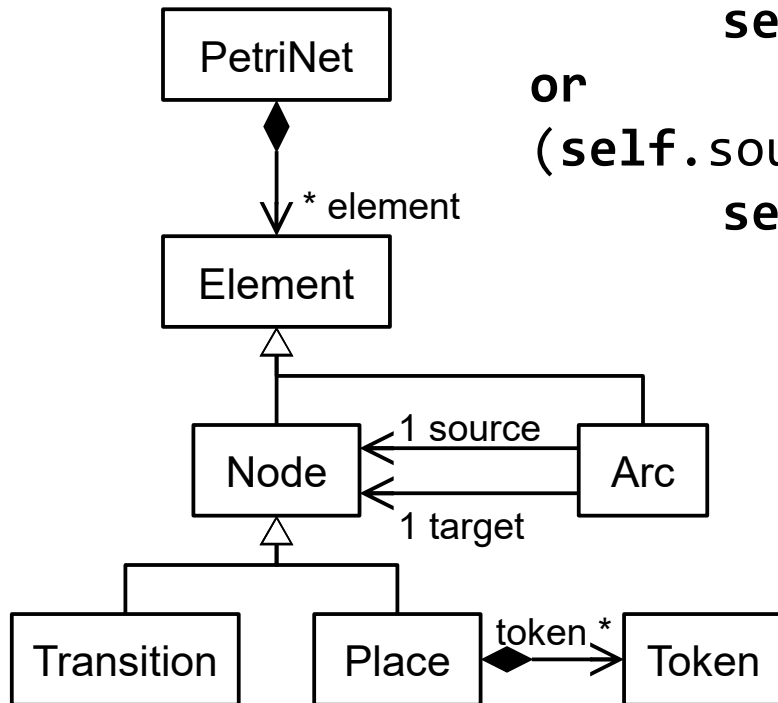
What's missing?



- Invariant constraint written in the **Object Constraint Language (OCL)**:

```

context Arc
inv "No Arcs Between Nodes Of The Same Kind":
((self.source.ocIsKindOf(Place) and
  self.target.ocIsKindOf(Transition))
or
 (self.source.ocIsKindOf(Transition) and
  self.target.ocIsKindOf(Place) ) );
  
```



OCCL – Example

- The **Object Constraint Language (OCL)** is a formal textual language that allows us to specify **constraints** and **queries** on models with a MOF-style metamodel (UML, MOF, ...)
 - OMG standard: <http://www.omg.org/spec/OCL/>

OCCL – Example

- The **Object Constraint Language (OCL)** is a formal textual language that allows us to specify **constraints** and **queries** on models with a MOF-style metamodel (UML, MOF, ...)
 - OMG standard: <http://www.omg.org/spec/OCL/>
- The OCL language and an interpreter are also implemented for EMF

OCCL – Example

- The **Object Constraint Language (OCL)** is a formal textual language that allows us to specify **constraints** and **queries** on models with a MOF-style metamodel (UML, MOF, ...)
 - **OMG standard:** <http://www.omg.org/spec/OCL/>
- The OCL language and an interpreter are also implemented for EMF
- OCL is used in many other standards to express constraints: MOF, UML, QVT, ...

- The **Object Constraint Language (OCL)** has been developed to achieve the following goals:

- The **Object Constraint Language (OCL)** has been developed to achieve the following goals:
 - to be formal, precise, unambiguous

- The **Object Constraint Language (OCL)** has been developed to achieve the following goals:
 - to be formal, precise, unambiguous
 - to be applicable for a large number of users (business or system modeler, programmers)

- The **Object Constraint Language (OCL)** has been developed to achieve the following goals:
 - to be formal, precise, unambiguous
 - to be applicable for a large number of users (business or system modeler, programmers)
 - to be a constraint and query language, not a programming language

- The **Object Constraint Language (OCL)** has been developed to achieve the following goals:
 - to be formal, precise, unambiguous
 - to be applicable for a large number of users (business or system modeler, programmers)
 - to be a constraint and query language, not a programming language
 - to be tool supported

- OCL constraints and queries have **no side-effects**

- OCL constraints and queries have **no side-effects**
- The **evaluation** of an OCL expression **returns a value**

- OCL constraints and queries have **no side-effects**
- The **evaluation** of an OCL expression **returns a value**
 - multiple **types** are supported: we get to them shortly

- OCL constraints and queries have **no side-effects**
- The **evaluation** of an OCL expression **returns a value**
 - multiple **types** are supported: we get to them shortly
 - When an **invariant constraint: Boolean**

- OCL constraints and queries have **no side-effects**
- The **evaluation** of an OCL expression **returns a value**
 - multiple **types** are supported: we get to them shortly
 - When an **invariant constraint: Boolean**
- OCL is **not a programming language**

- OCL constraints and queries have **no side-effects**
- The **evaluation** of an OCL expression **returns a value**
 - multiple **types** are supported: we get to them shortly
 - When an **invariant constraint: Boolean**
- OCL is **not a programming language**
 - no program logic or flow control

- OCL constraints and queries have **no side-effects**
- The **evaluation** of an OCL expression **returns a value**
 - multiple **types** are supported: we get to them shortly
 - When an **invariant constraint: Boolean**
- OCL is **not a programming language**
 - no program logic or flow control
 - no invocation of processes or activation of non-query operations

- OCL constraints and queries have **no side-effects**
- The **evaluation** of an OCL expression **returns a value**
 - multiple **types** are supported: we get to them shortly
 - When an **invariant constraint: Boolean**
- OCL is **not a programming language**
 - no program logic or flow control
 - no invocation of processes or activation of non-query operations
- OCL is a **typed language**

- OCL constraints and queries have **no side-effects**
- The **evaluation** of an OCL expression **returns a value**
 - multiple **types** are supported: we get to them shortly
 - When an **invariant constraint: Boolean**
- OCL is **not a programming language**
 - no program logic or flow control
 - no invocation of processes or activation of non-query operations
- OCL is a **typed language**
 - Each classifier in the model represents a distinct OCL type

- OCL constraints and queries have **no side-effects**
- The **evaluation** of an OCL expression **returns a value**
 - multiple **types** are supported: we get to them shortly
 - When an **invariant constraint: Boolean**
- OCL is **not a programming language**
 - no program logic or flow control
 - no invocation of processes or activation of non-query operations
- OCL is a **typed language**
 - Each classifier in the model represents a distinct OCL type
 - we can define variables typed over classifiers in the model

- OCL constraints and queries have **no side-effects**
- The **evaluation** of an OCL expression **returns a value**
 - multiple **types** are supported: we get to them shortly
 - When an **invariant constraint: Boolean**
- OCL is **not a programming language**
 - no program logic or flow control
 - no invocation of processes or activation of non-query operations
- OCL is a **typed language**
 - Each classifier in the model represents a distinct OCL type
 - we can define variables typed over classifiers in the model
 - Includes a set of predefined types

- OCL can be used

- OCL can be used
 - as a **query language**

- OCL can be used
 - as a **query language**
 - to **specify invariants** on classes and types in a class model

- OCL can be used
 - as a **query language**
 - to **specify invariants** on classes and types in a class model
 - to describe **pre- and post conditions** on operations

- OCL can be used
 - as a **query language**
 - to **specify invariants** on classes and types in a class model
 - to describe **pre- and post conditions** on operations
 - to describe **guards** (in UML behavior models)

- OCL can be used
 - as a **query language**
 - to **specify invariants** on classes and types in a class model
 - to describe **pre- and post conditions** on operations
 - to describe **guards** (in UML behavior models)
 - to specify **derivation rules** for **derived features** (attributes or references/associations)

- Each OCL expression is related to an object, the instance of a class

- Each OCL expression is related to an object, the instance of a class
 - A **context declaration** is used to determine the class

- Each OCL expression is related to an object, the instance of a class
 - A **context declaration** is used to determine the class
- **self** refers to the contextual instance

- Each OCL expression is related to an object, the instance of a class
 - A **context declaration** is used to determine the class
- **self** refers to the contextual instance
- Example:

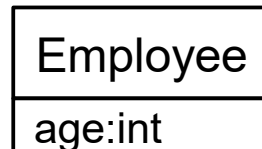
- Each OCL expression is related to an object, the instance of a class
 - A **context declaration** is used to determine the class
- **self** refers to the contextual instance
- Example:

Employee
age:int

```
context Employee
inv: self.age >= 19
```

- Each OCL expression is related to an object, the instance of a class
 - A **context declaration** is used to determine the class
- **self** refers to the contextual instance

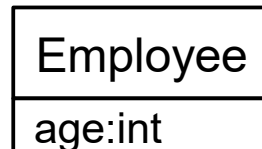
- Example:



context: (an instance of) Employee

context Employee
inv: **self**.age >= 19

- Each OCL expression is related to an object, the instance of a class
 - A **context declaration** is used to determine the class
- **self** refers to the contextual instance
- Example:

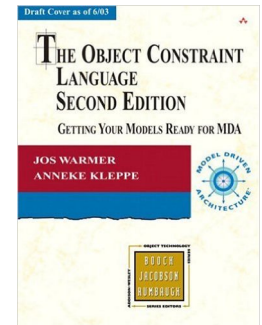


context: (an instance of) Employee

context Employee
inv: **self**.age >= 19

inv: an invariant constraint; must be true for all instances of the context class (here: for all Employee instances)

- Jordi Cabot, Martin Gogolla: Object Constraint Language (OCL): A Definitive Guide, in Formal Methods for Model-Driven Engineering, Volume 7320 of Lecture Notes in Computer Science, pp 58-90, 2012.
 - http://link.springer.com/chapter/10.1007%2F978-3-642-30982-3_3
 - <http://modeling-languages.com/wp-content/uploads/2012/03/OCLChapter.pdf>
- Jos Warmer, Anneke Kleppe: The Object Constraint Language: Getting Your Models Ready for MDA, Addison-Wesley Professional; 2nd edition, 2003.
- Christian Hein, A presentation of OCL 2, Open Model CourseWare, 2006
 - <https://eclipse.org/gmt/omcw/resources/chapter01/downloads/OCL2.Fraunhofer.ppt>



3.2. OCL types

- OCL is a typed language

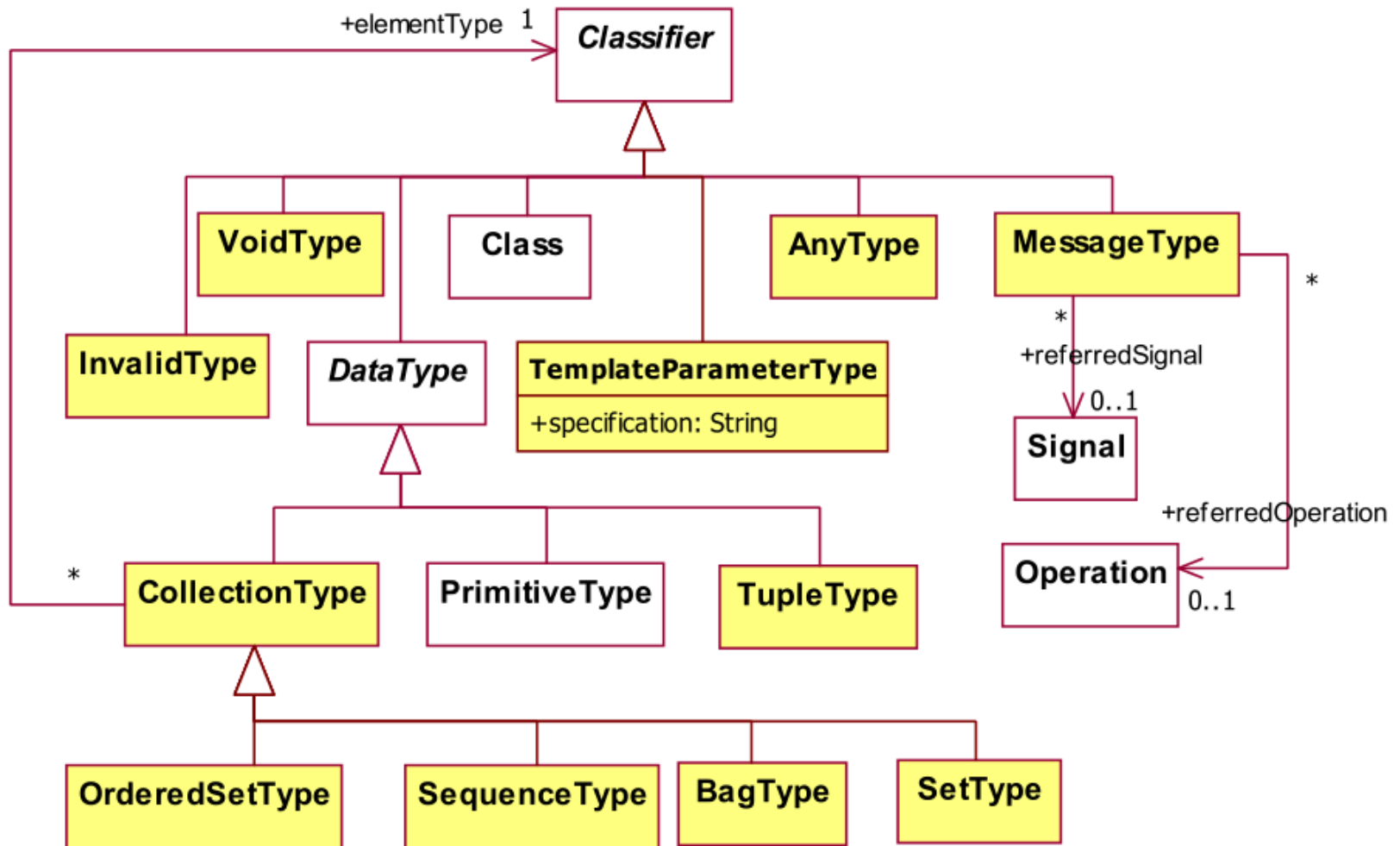
- OCL is a typed language
 - queries evaluate to values of certain types

- OCL is a typed language
 - queries evaluate to values of certain types
 - we can work with variables of certain types

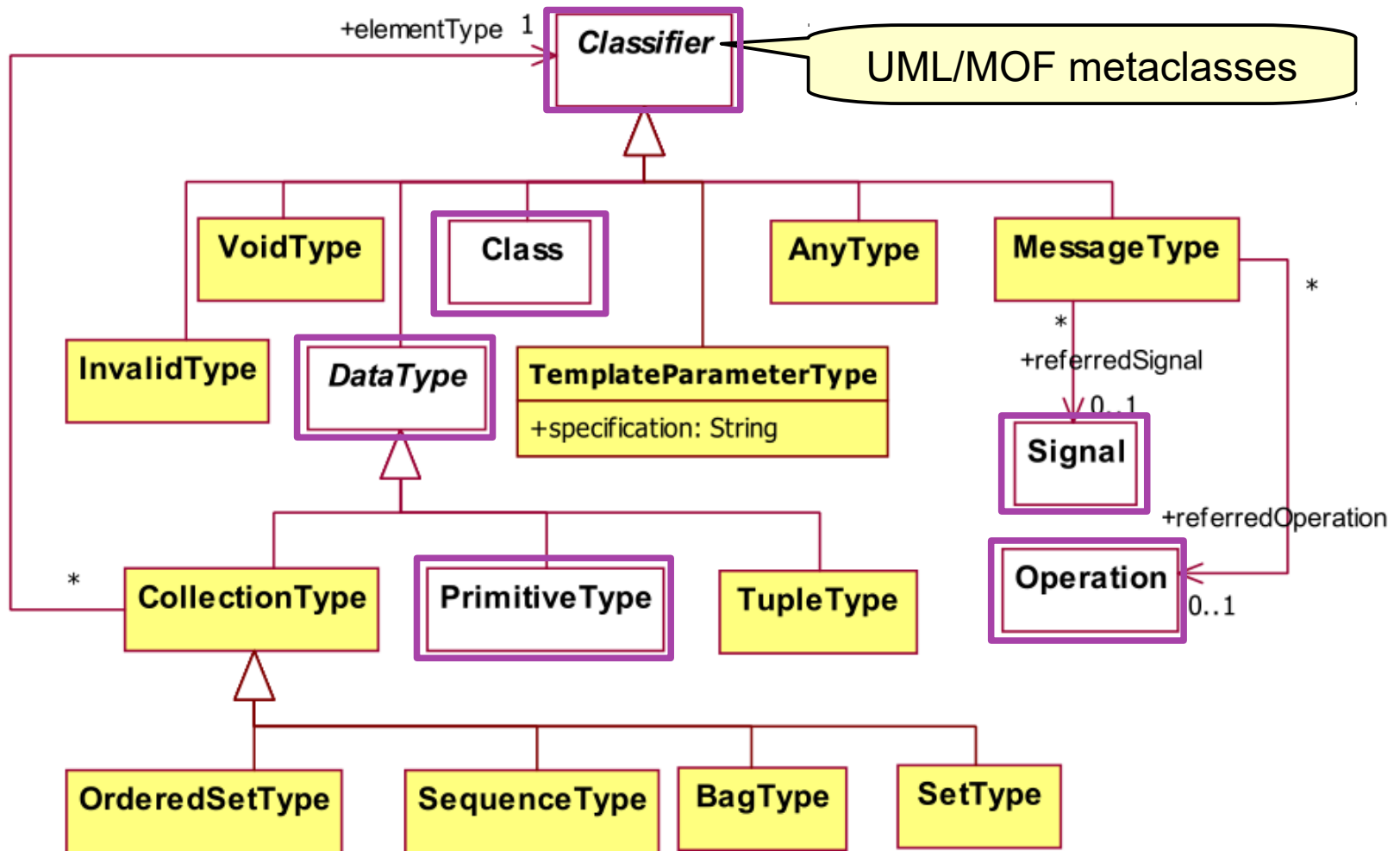
- OCL is a typed language
 - queries evaluate to values of certain types
 - we can work with variables of certain types
 - different types offer different functions

- OCL is a typed language
 - queries evaluate to values of certain types
 - we can work with variables of certain types
 - different types offer different functions
 - for example `collection->forAll(...)`

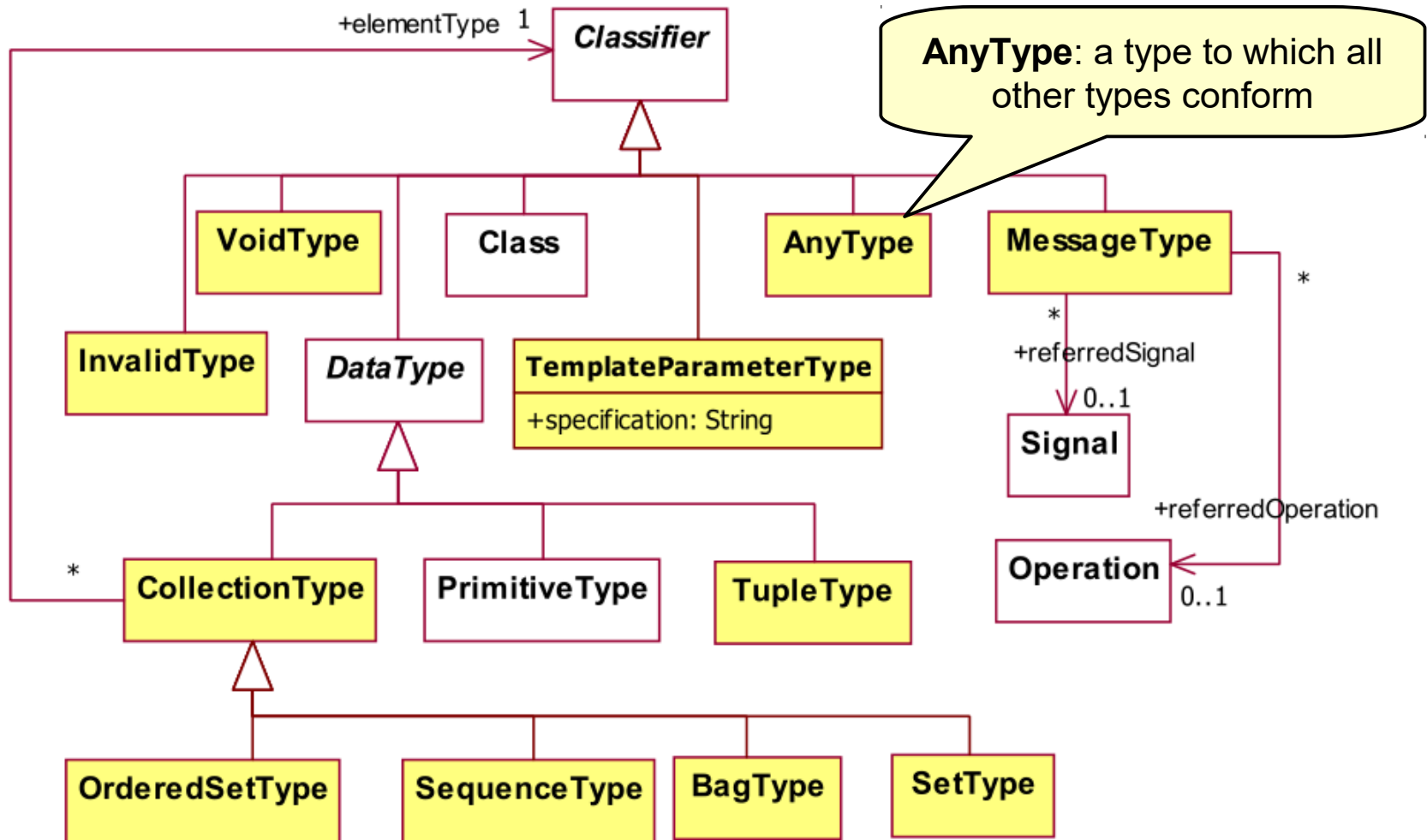
OCL Types Metamodel



OCL Types Metamodel



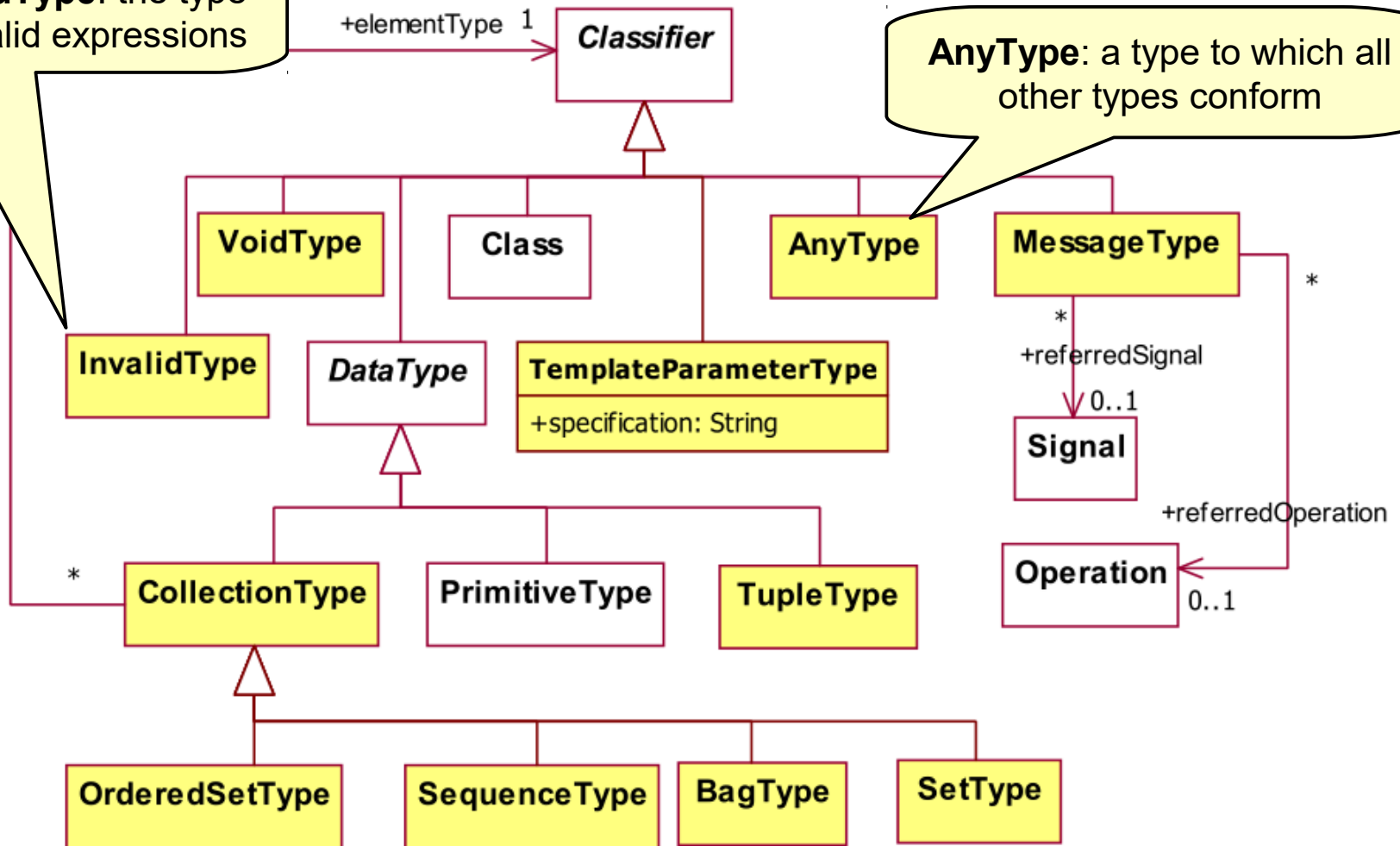
OCL Types Metamodel



OCL Types Metamodel

InvalidType: the type of invalid expressions

AnyType: a type to which all other types conform

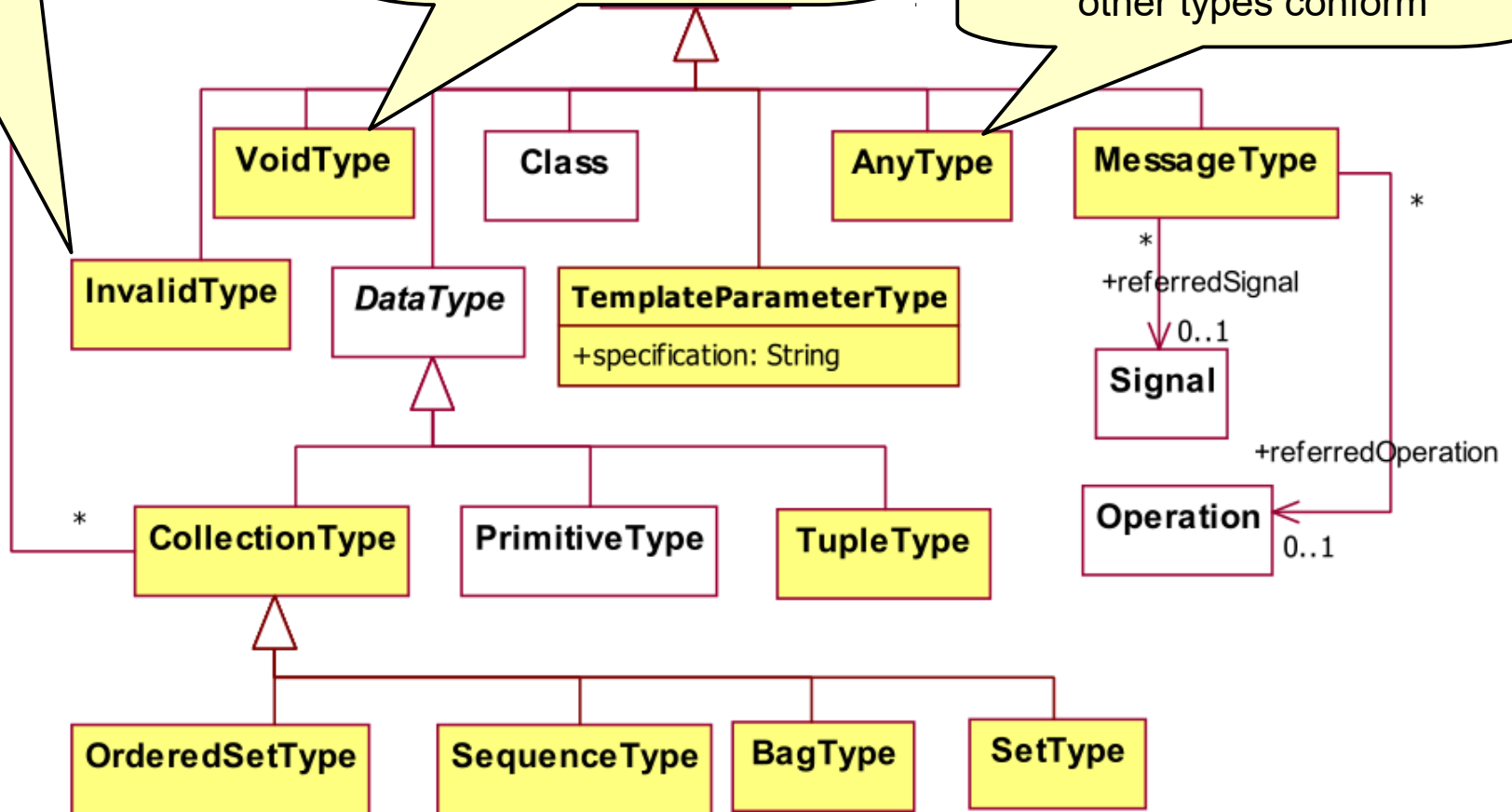


OCIL Types Metamodel

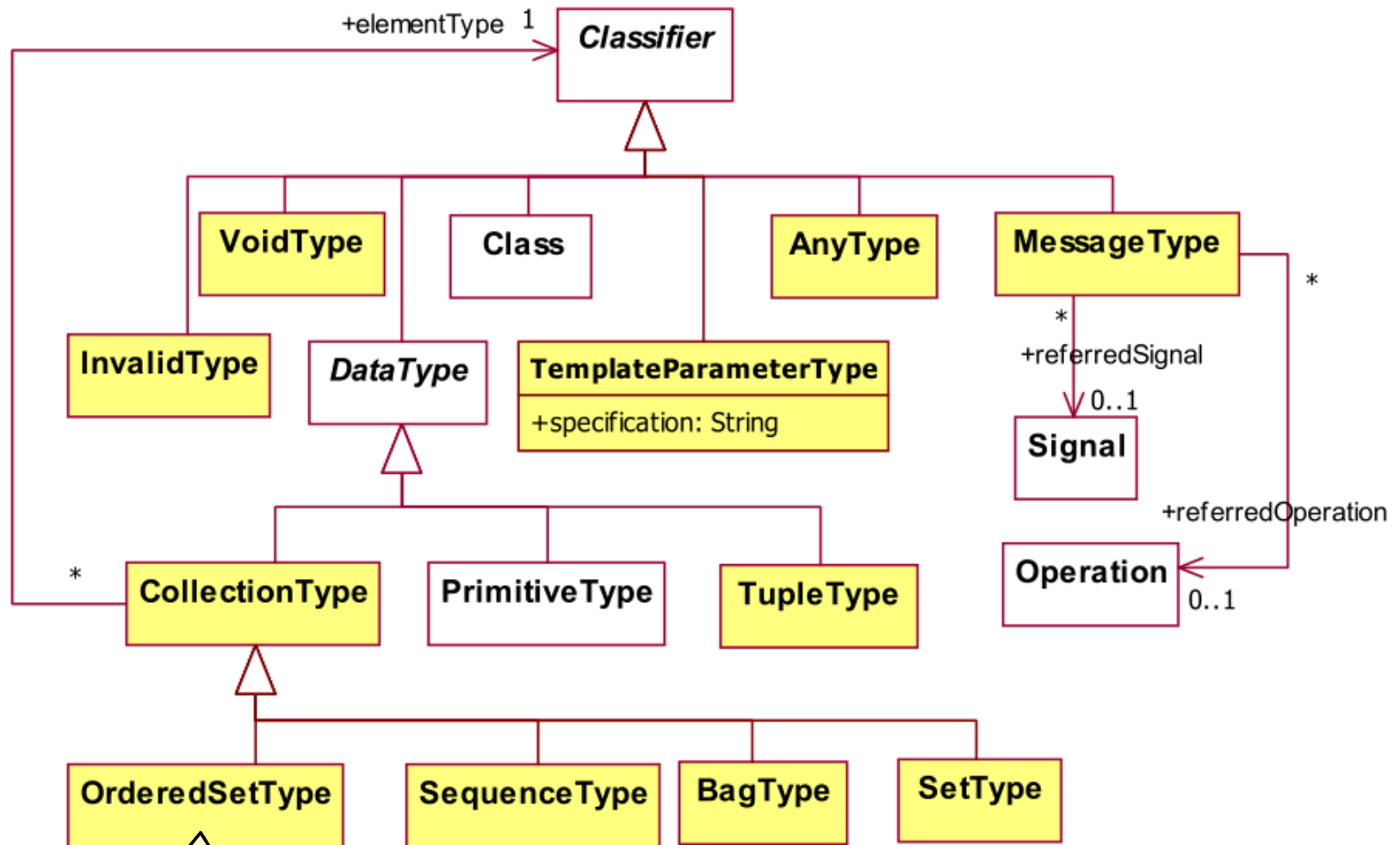
InvalidType: the type of invalid expressions

VoidType: a type to which all other types conform except InvalidType

AnyType: a type to which all other types conform

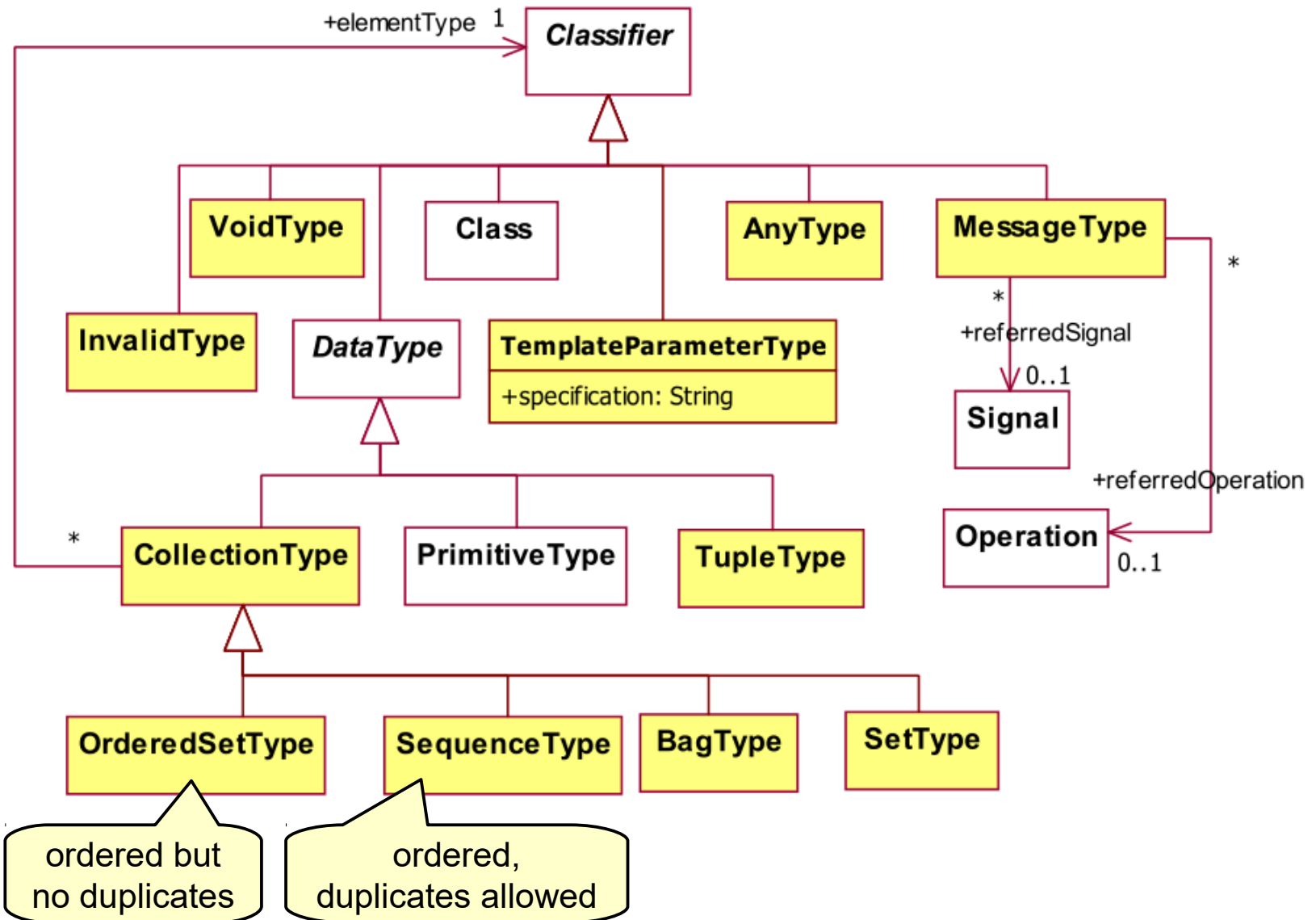


OCl Types Metamodel

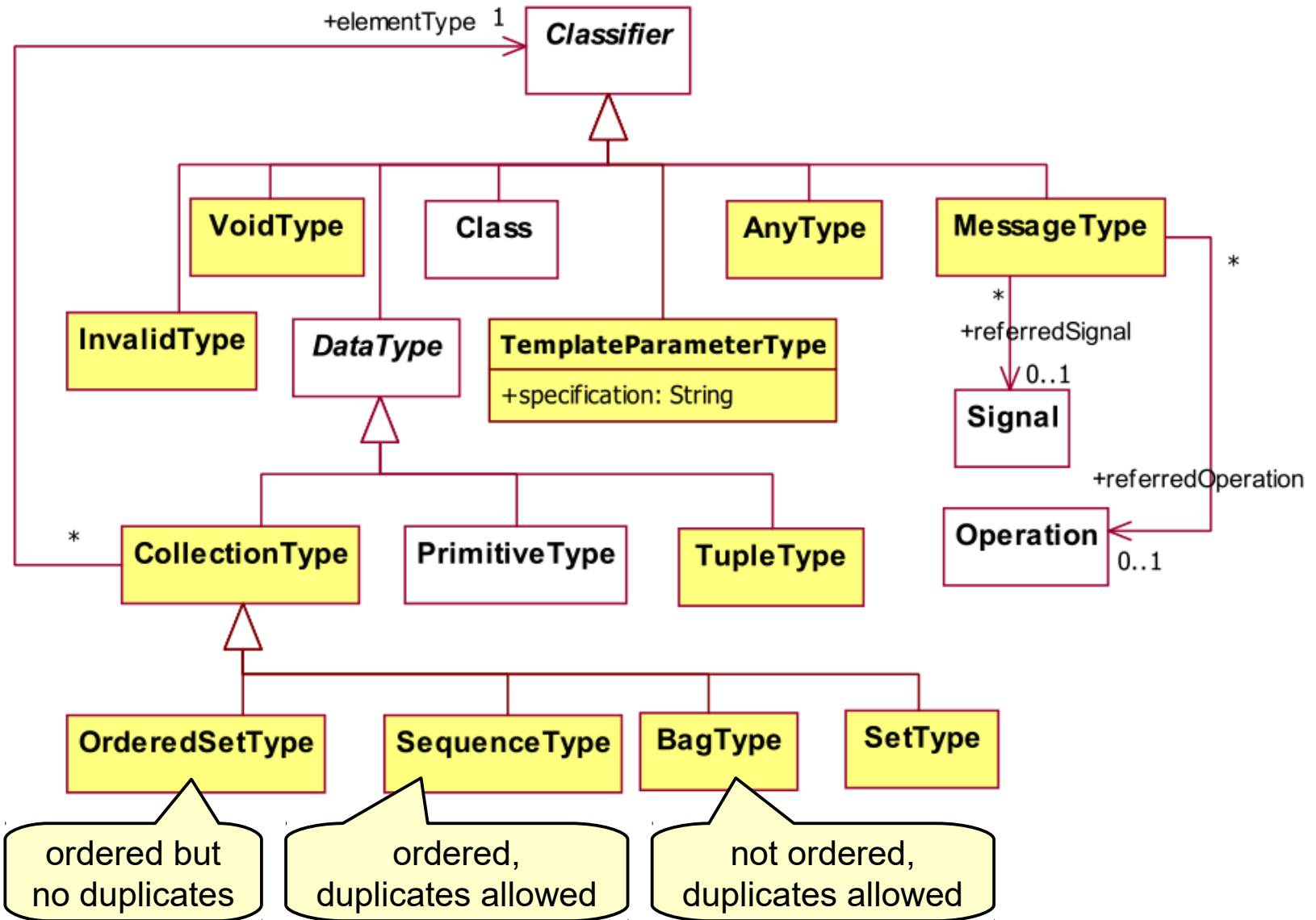


ordered but
no duplicates

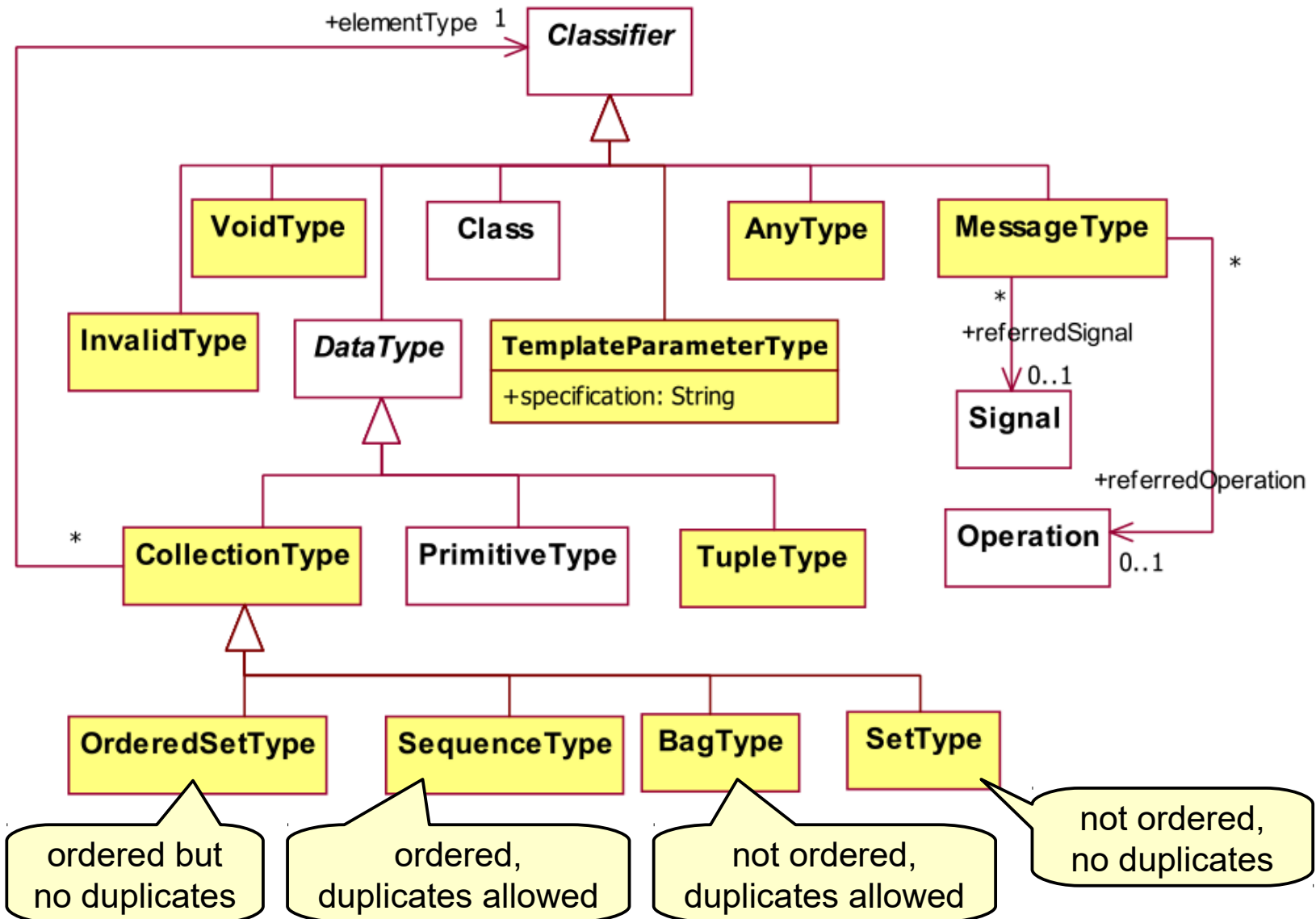
OCaml Types Metamodel



OCaml Types Metamodel

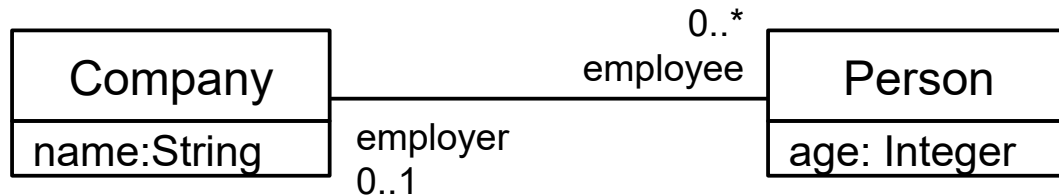


OCaml Types Metamodel



Operations on Collection Types

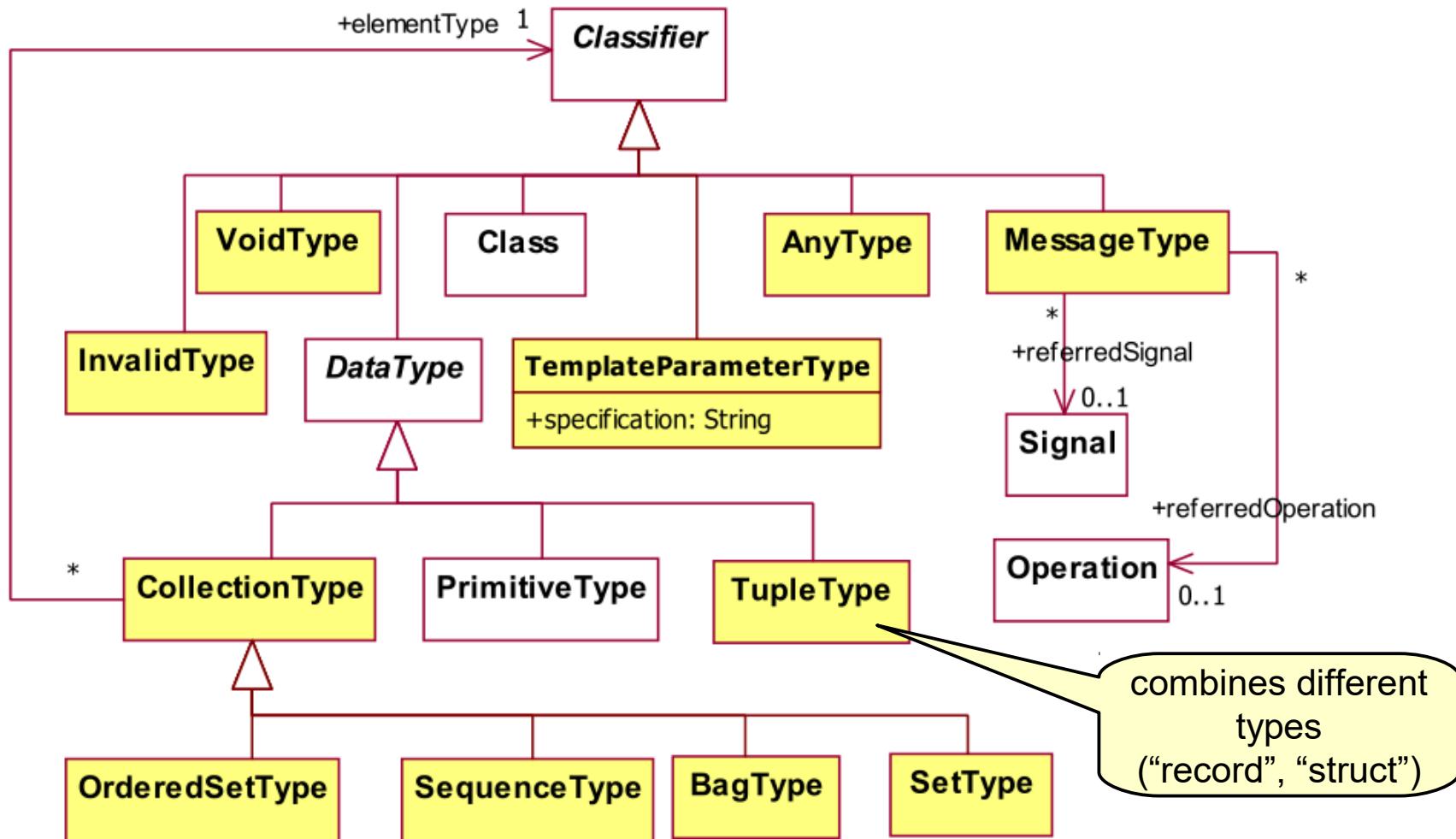
- For example:



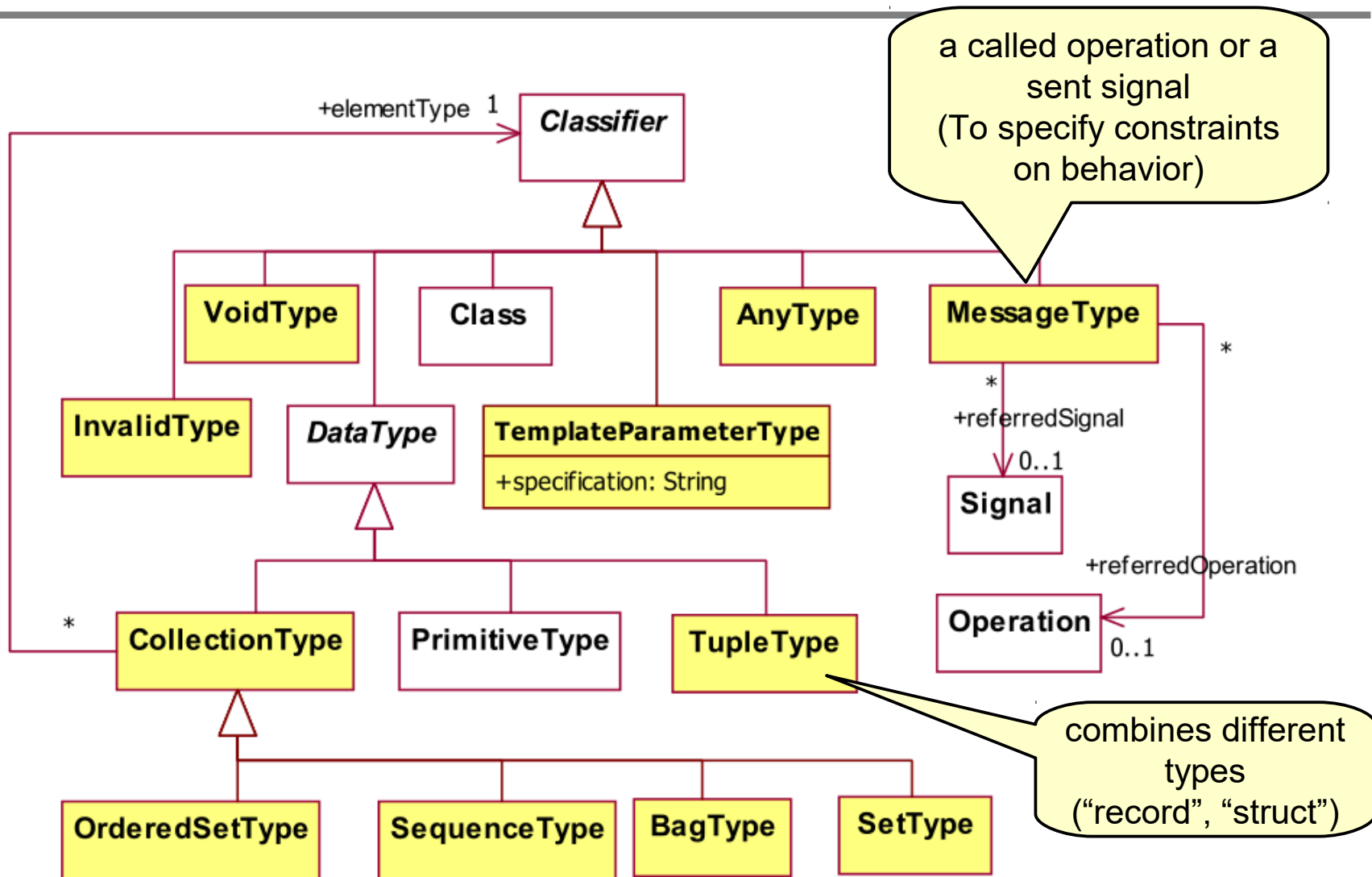
```

context Company
inv: self.employee->forAll( age <= 65 )
inv: self.employee->forAll( p | p.age <= 65 )
inv: self.employee->forAll( p : Person | p.age <= 65 )
  
```

OCCL Types Metamodel

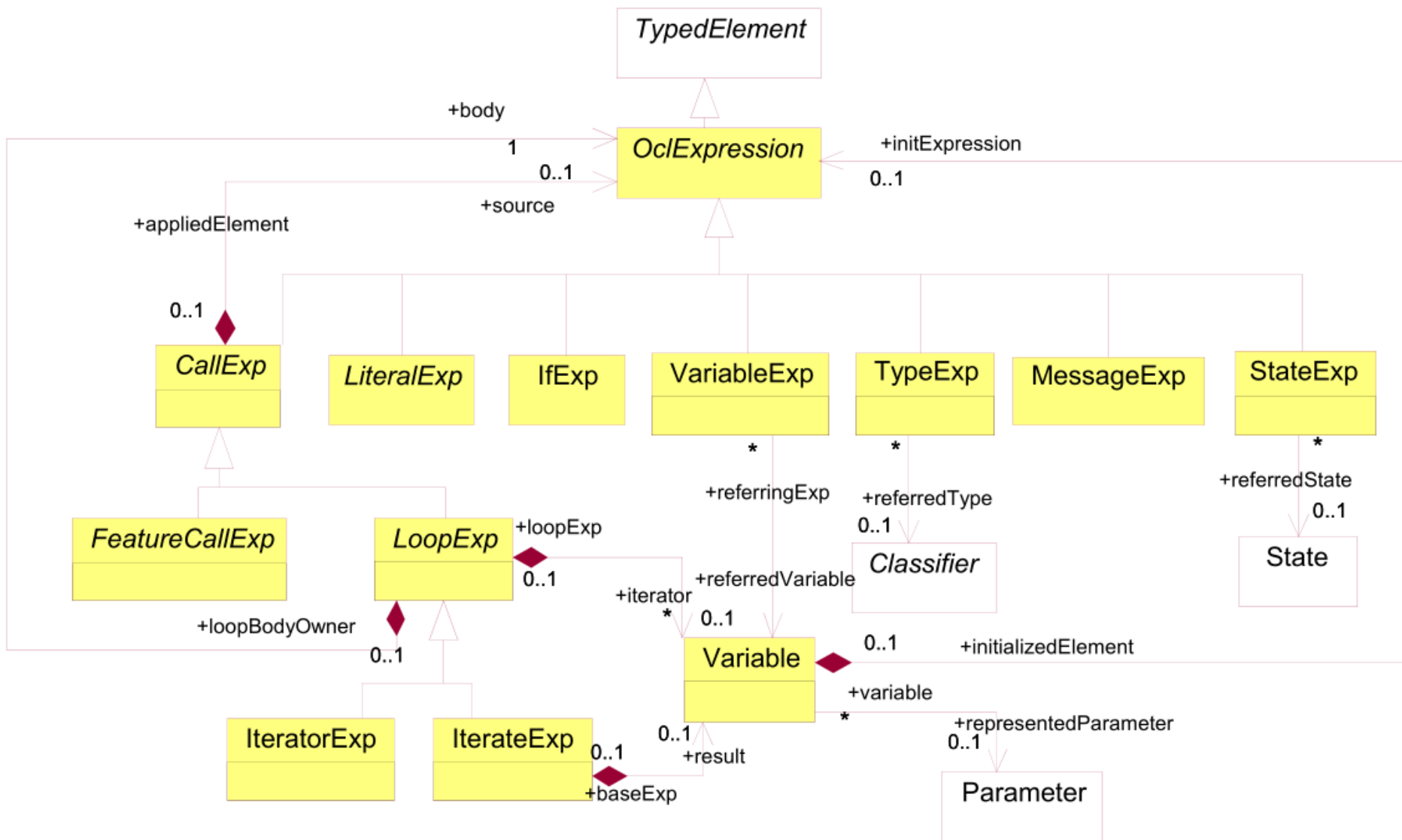


OCCL Types Metamodel

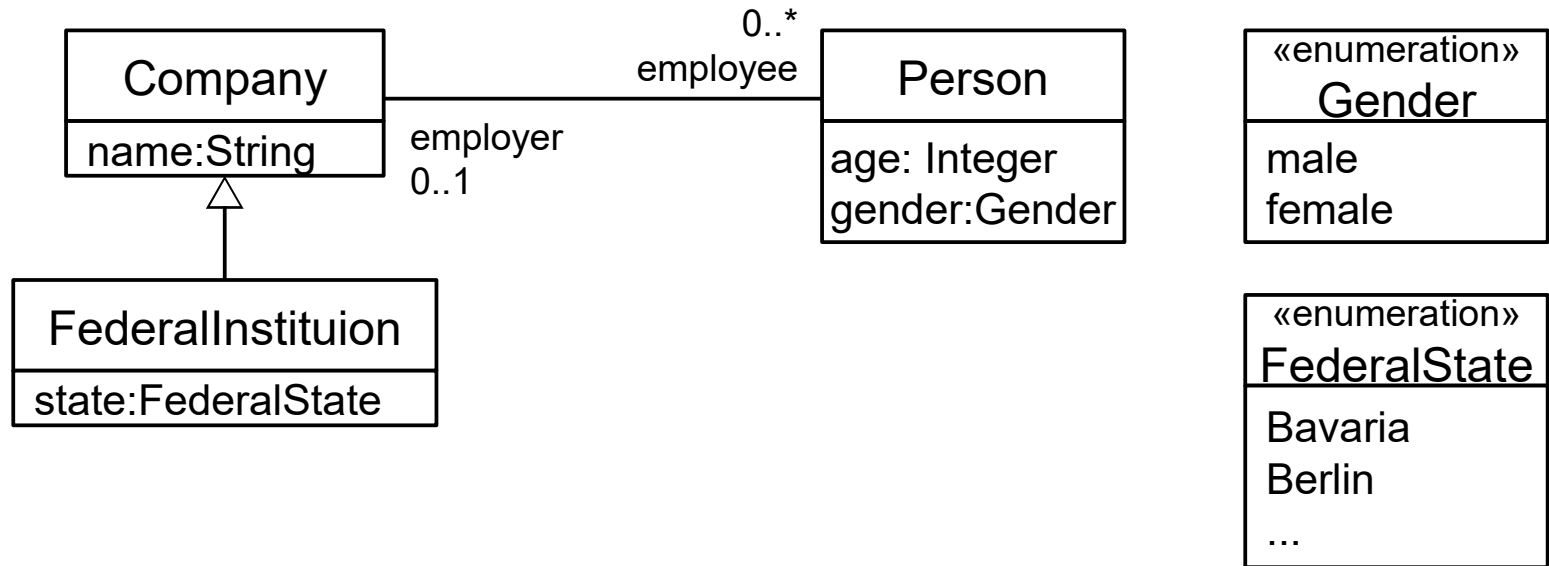


3.2. OCL expressions

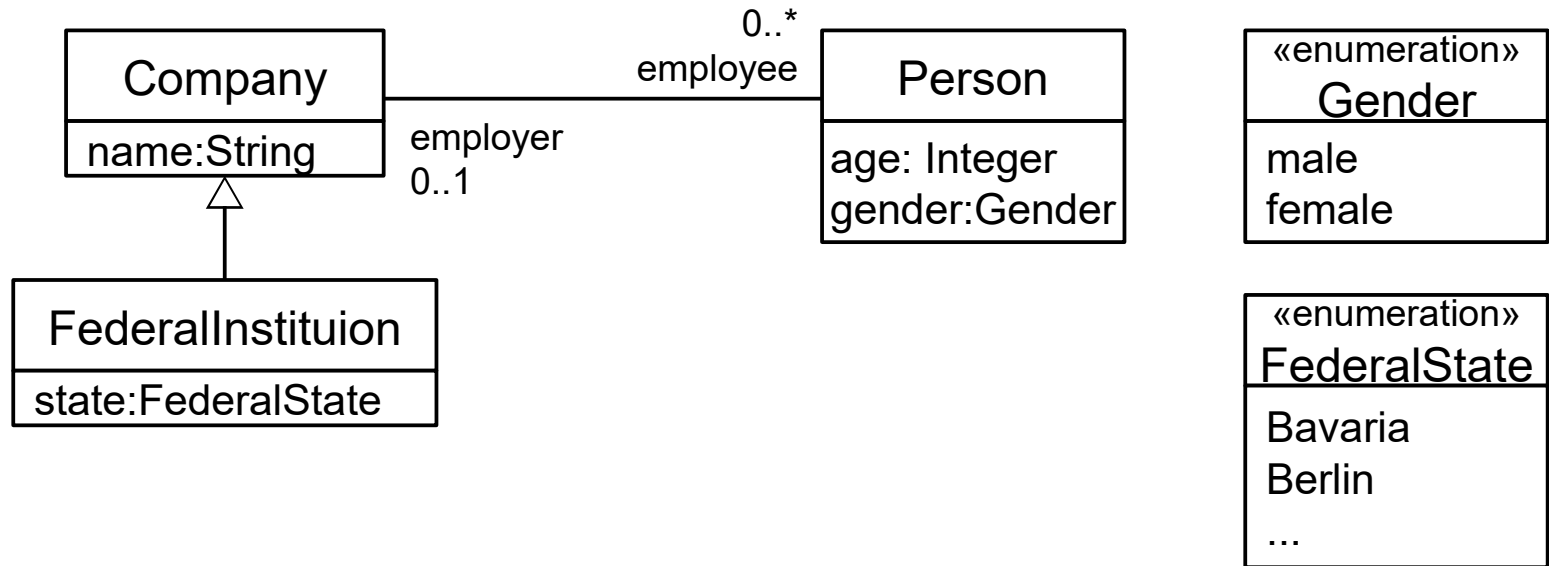
OCL Expressions



Accessing Objects and Properties



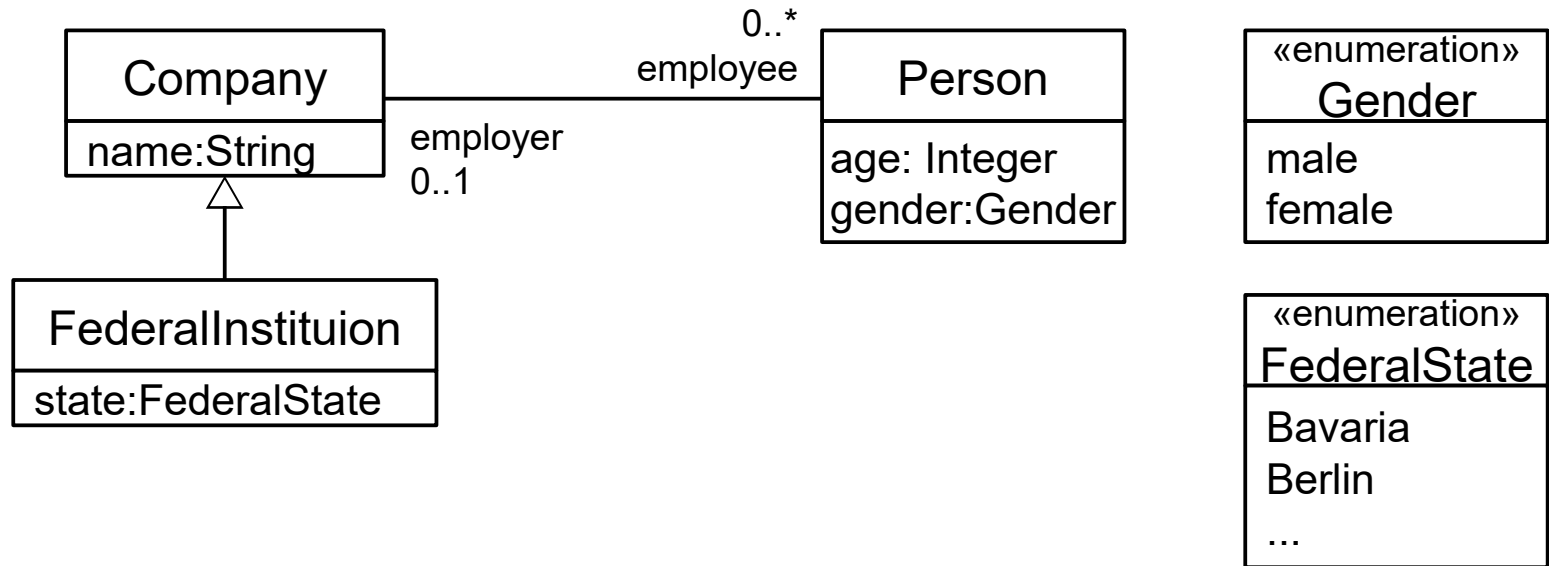
Accessing Objects and Properties



```

context Person
inv:  self.age > 18
  
```

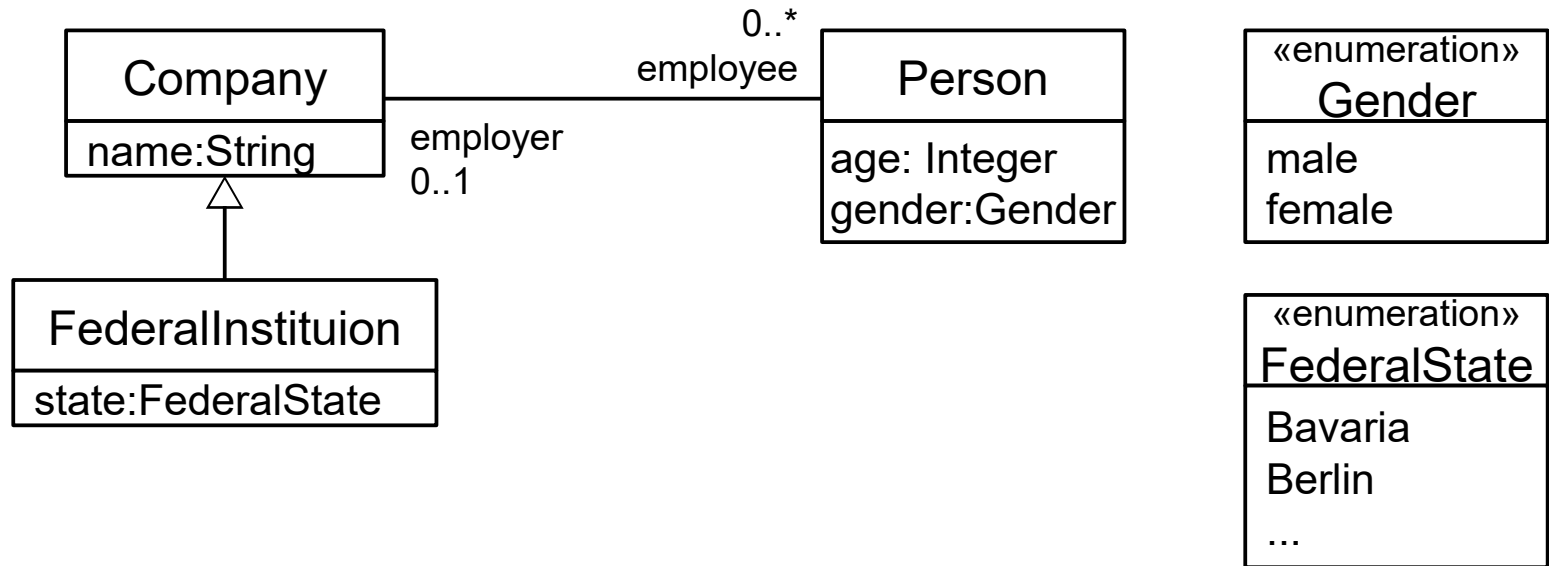
Accessing Objects and Properties



```

context Person
inv:  self.age > 18
inv:  self.gender <> Gender::male
  
```

Accessing Objects and Properties



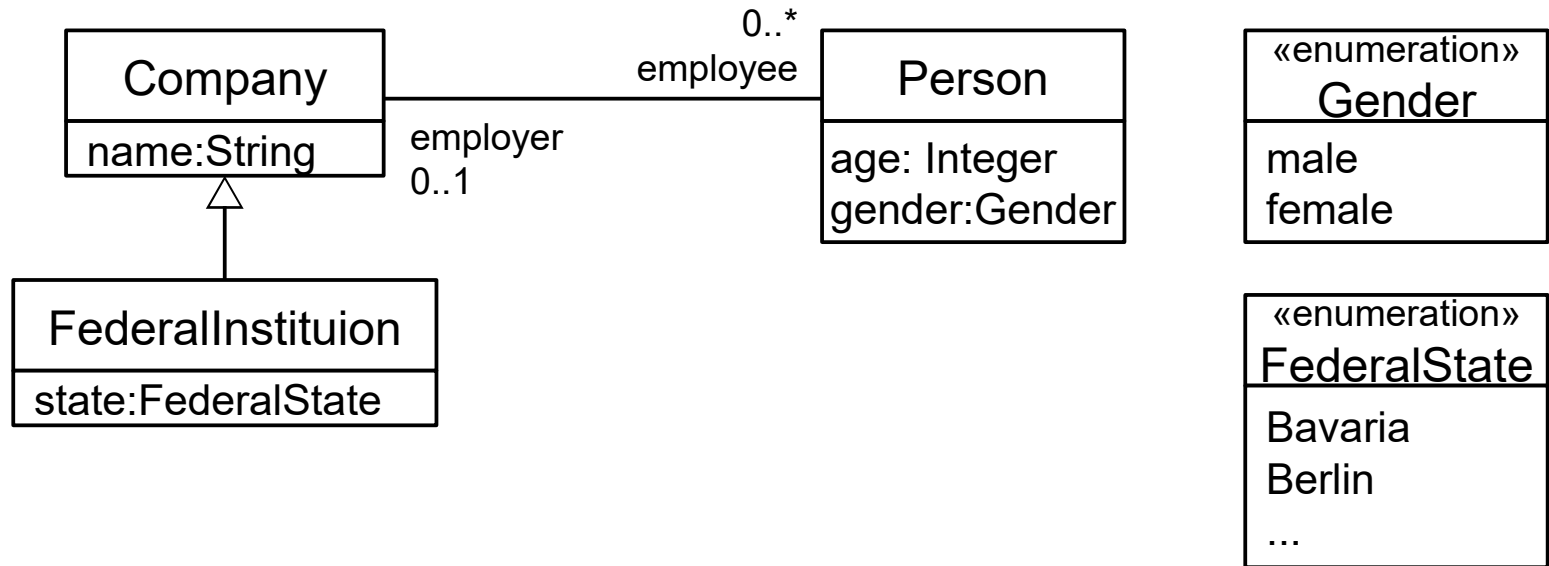
context Person

inv: `self.age > 18`

inv: `self.gender <> Gender::male`

inv: `self.employer.oclAsType(FederalInstitution).state = FederalState::LowerSaxony`

Accessing Objects and Properties



context Person

inv: `self.age > 18`

inv: `self.gender <> Gender::male`

inv: `self.employer.oclAsType(FederalInstitution).state = FederalState::LowerSaxony`

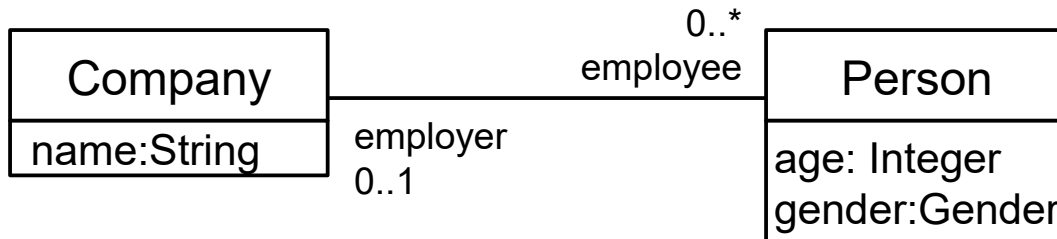


Collection Operations: Select and Reject

- **select** and **reject** are operations on collections to specify subsets
 - **select**: filters elements conforming to a condition
 - **reject**: excludes elements conforming to a condition
 - result type: same as original

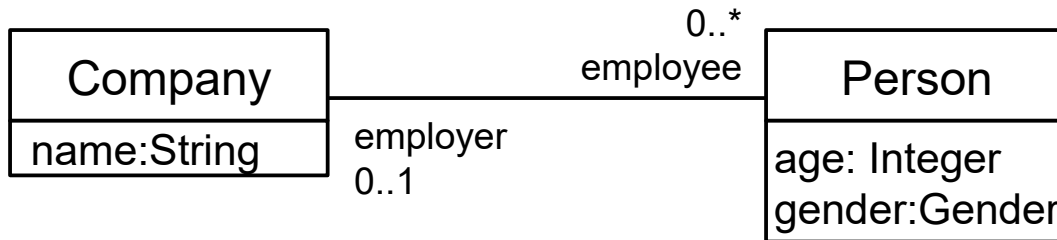
Collection Operations: Select and Reject

- **select** and **reject** are operations on collections to specify subsets
 - **select**: filters elements conforming to a condition
 - **reject**: excludes elements conforming to a condition
 - result type: same as original



Collection Operations: Select and Reject

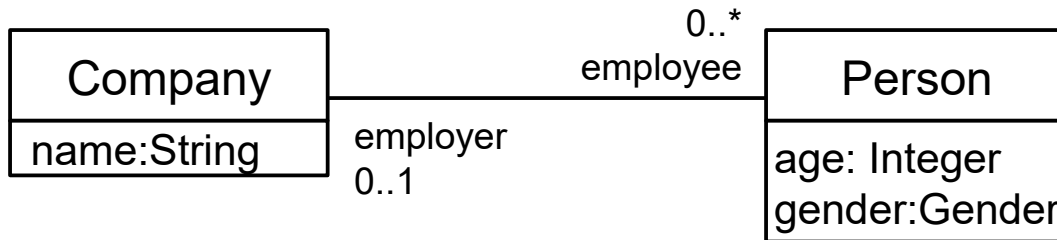
- **select** and **reject** are operations on collections to specify subsets
 - **select**: filters elements conforming to a condition
 - **reject**: excludes elements conforming to a condition
 - result type: same as original



context Company

Collection Operations: Select and Reject

- **select** and **reject** are operations on collections to specify subsets
 - **select**: filters elements conforming to a condition
 - **reject**: excludes elements conforming to a condition
 - result type: same as original

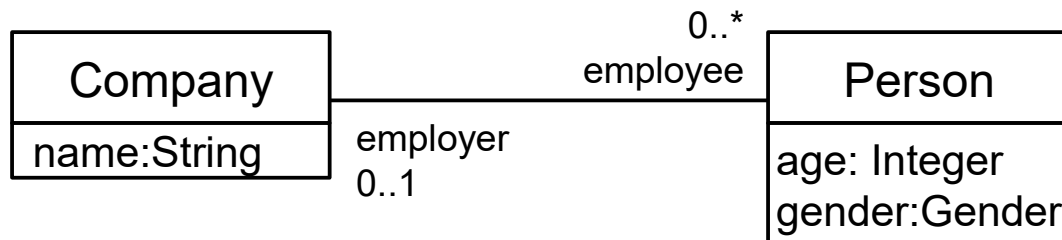


context Company

inv: `self.employee->select(age > 65)->isEmpty()`

Collection Operations: Collect

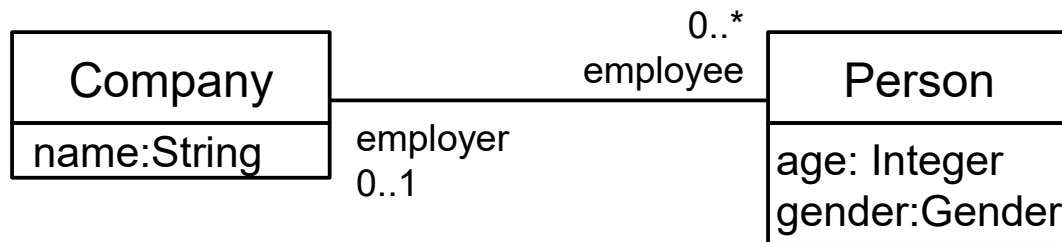
- **collect** operations specify a collection derived from some other collection
 - result type: Bag



returns a bag of integers, for example [32, 55, 43, 32, 27]

Collection Operations: Collect

- **collect** operations specify a collection derived from some other collection
 - result type: Bag

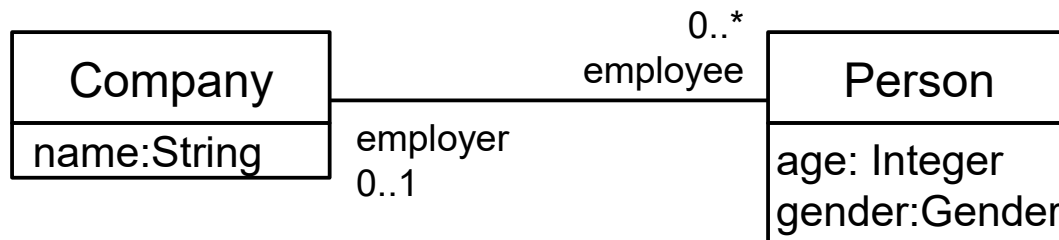


context Company

returns a bag of integers, for example [32, 55, 43, 32, 27]

Collection Operations: Collect

- **collect** operations specify a collection derived from some other collection
 - result type: Bag



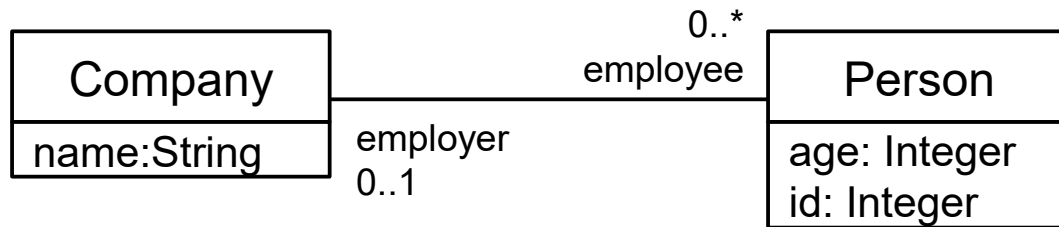
```

context Company
self.employee->collect(age)
    
```

returns a bag of integers, for example [32, 55, 43, 32, 27]

Collection Operations: ForAll

- A **forAll** operation specifies a condition that must hold for all objects in a collection
 - result type: Boolean

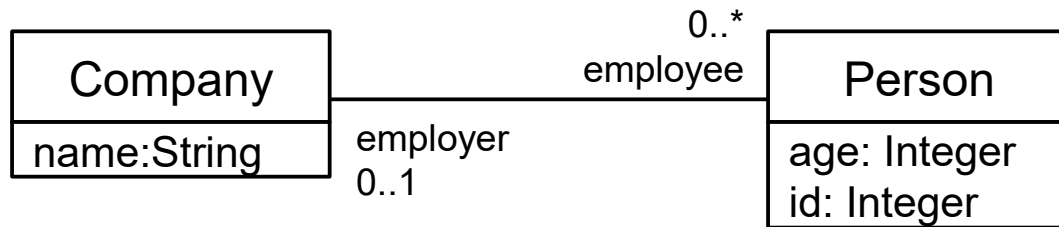


context Company

inv: `self.employee->forAll(age <= 65)`

Collection Operations: ForAll

- A **forAll** operation specifies a condition that must hold for all objects in a collection
 - result type: Boolean



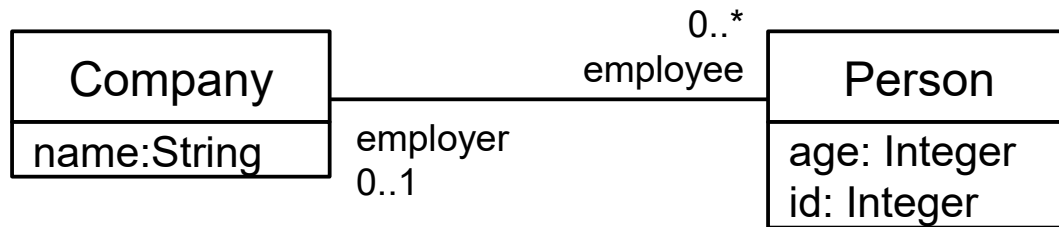
context Company

inv: `self.employee->forAll(age <= 65)`

inv: `self.employee->forAll(p | p.age <= 65)`

Collection Operations: ForAll

- A **forAll** operation specifies a condition that must hold for all objects in a collection
 - result type: Boolean



context Company

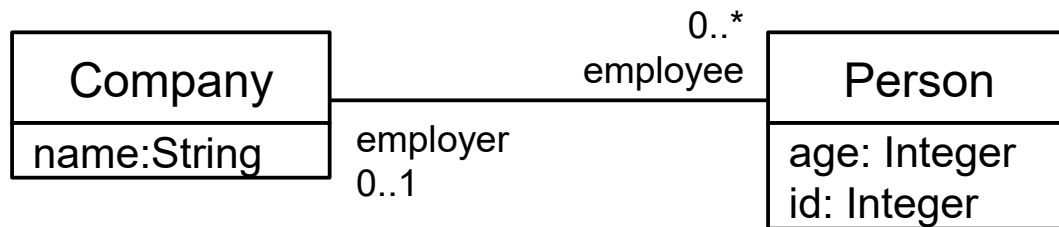
inv: `self.employee->forAll(age <= 65)`

inv: `self.employee->forAll(p | p.age <= 65)`

inv: `self.employee->forAll(p : Person | p.age <= 65)`

Collection Operations: ForAll

- A **forAll** operation specifies a condition that must hold for all objects in a collection
 - result type: Boolean

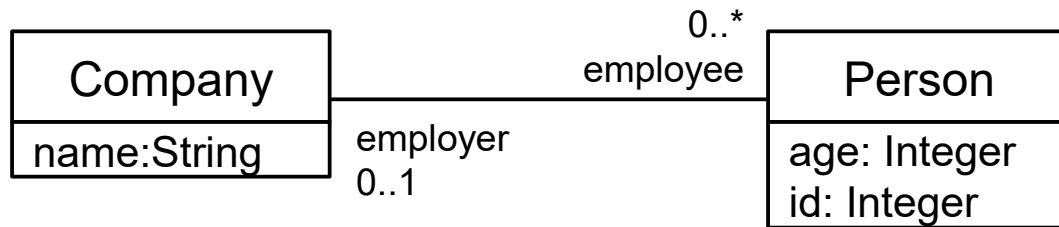


```

context Company
inv:  self.employee->forAll( age <= 65 )
inv:  self.employee->forAll( p | p.age <= 65 )
inv:  self.employee->forAll( p : Person | p.age <= 65 )
inv:  self.employee->forAll( p1 |
    self.employee->forAll( p2 |
        p1 <> p2 implies p1.id <> p2.id ))
    
```

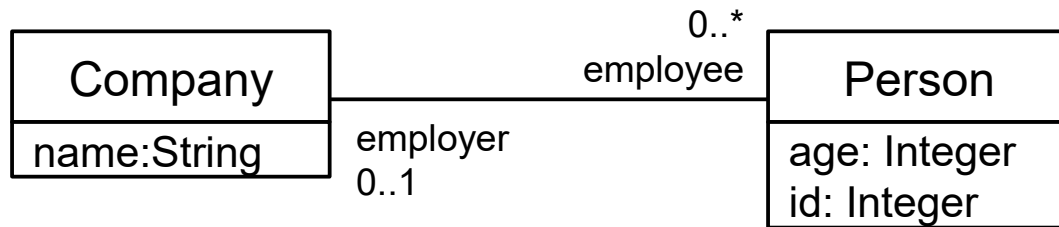

Collection Operations: Exists

- An **exists** operation specifies a condition that must hold for at least one object in a collection
 - result type: Boolean



Collection Operations: Exists

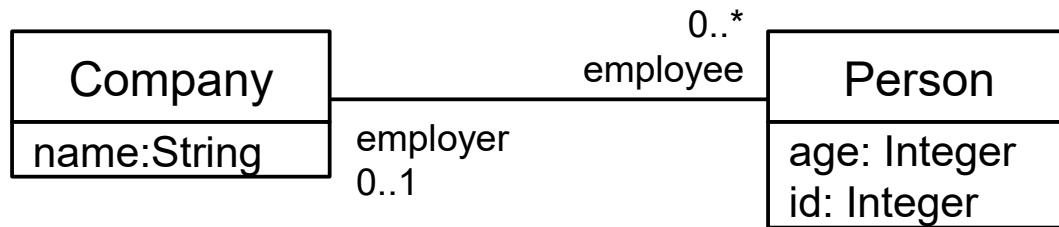
- An **exists** operation specifies a condition that must hold for at least one object in a collection
 - result type: Boolean



context Company

Collection Operations: Exists

- An **exists** operation specifies a condition that must hold for at least one object in a collection
 - result type: Boolean

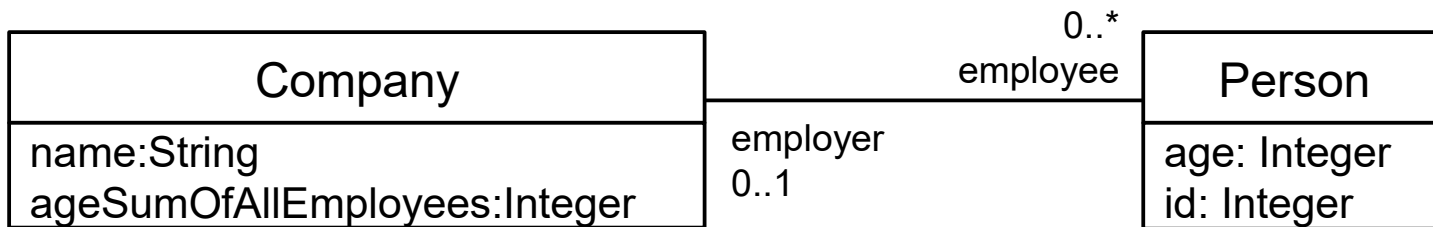


context Company

inv: `self.employee->exists(age <= 65)`

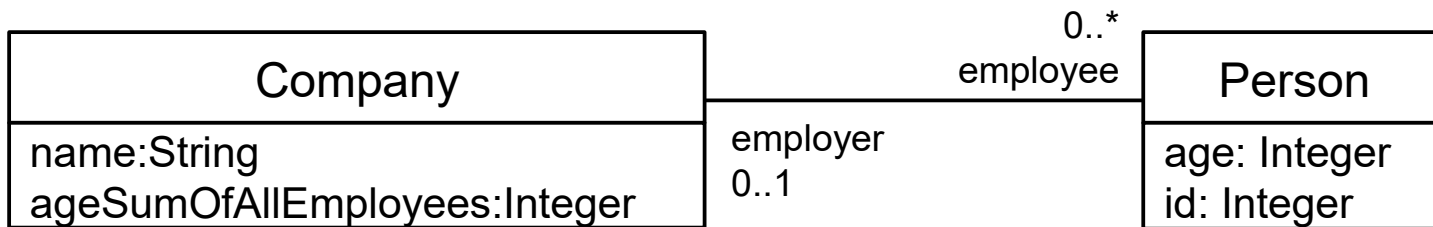
Collection Operations: Iterate

- An **iterate** operation iterates over objects in a collection and accumulates a value of in a certain return type



Collection Operations: Iterate

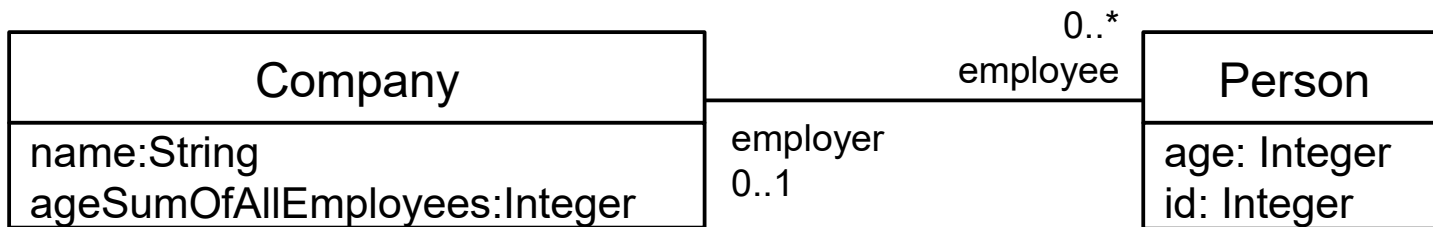
- An **iterate** operation iterates over objects in a collection and accumulates a value of in a certain return type



context `Company.ageSumOfAllEmployees:Integer`

Collection Operations: Iterate

- An **iterate** operation iterates over objects in a collection and accumulates a value of in a certain return type

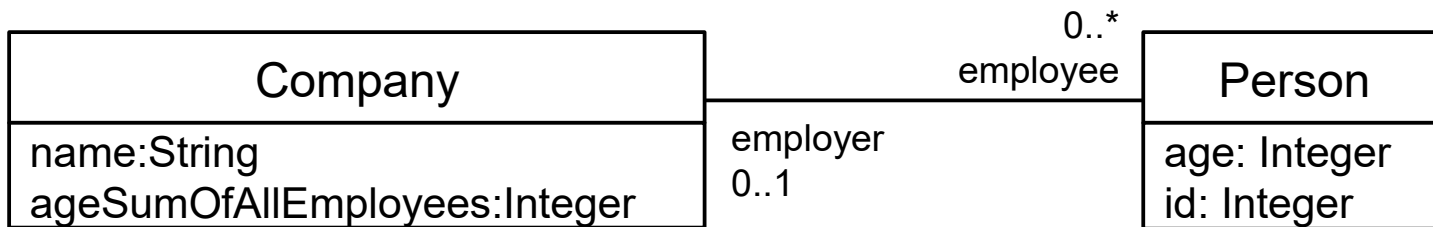


```

context Company.ageSumOfAllEmployees:Integer
body: self.employee->iterate(    p:Person ;
  
```

Collection Operations: Iterate

- An **iterate** operation iterates over objects in a collection and accumulates a value of in a certain return type

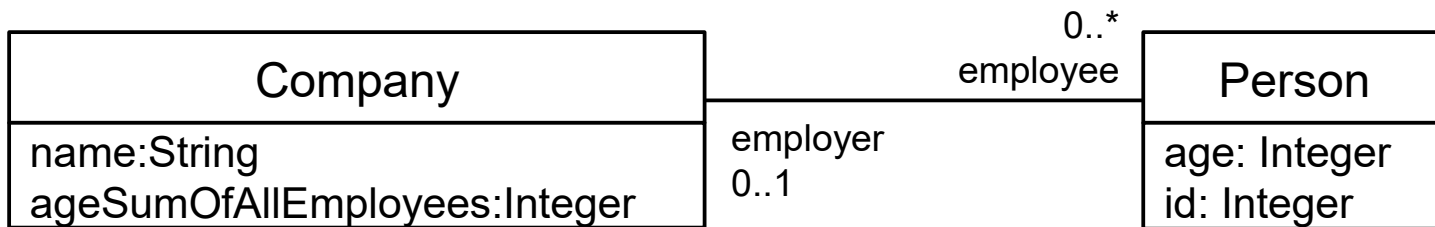


```

context Company.ageSumOfAllEmployees:Integer
body: self.employee->iterate(    p:Person ;
                                sum:Integer = 0 |
  
```

Collection Operations: Iterate

- An **iterate** operation iterates over objects in a collection and accumulates a value of in a certain return type

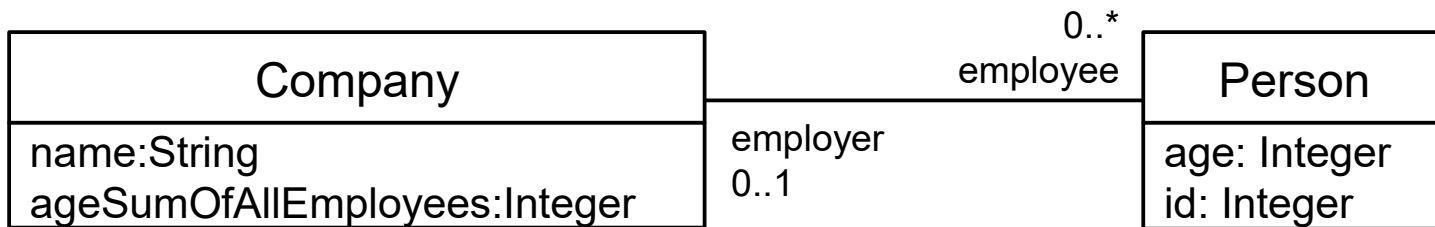


```

context Company.ageSumOfAllEmployees:Integer
body: self.employee->iterate(    p:Person ;
                                sum:Integer = 0 |
                                sum + p.age)
  
```


Collection Operations: Iterate

- An **iterate** operation iterates over objects in a collection and accumulates a value of in a certain return type



```

context Company.ageSumOfAllEmployees:Integer
body: self.employee->iterate(    p:Person ;
                                sum:Integer = 0 |
                                sum + p.age)
  
```

defines the value of the
derived attribute

Further OCL Operations

- **self.oclIsTypeOf(t:OclType):Boolean**
 - returns true if the type of self and t are the same

Further OCL Operations

- **self.oclIsTypeOf**(t:OclType):Boolean
 - returns true if the type of self and t are the same
- **self.oclIsKindOf**(t:OclType):Boolean
 - returns true if the type of self and t are the same or if t is a supertype of the type of self.

Further OCL Operations

- **self.oclIsTypeOf**(t:OclType):Boolean
 - returns true if the type of self and t are the same
- **self.oclIsKindOf**(t:OclType):Boolean
 - returns true if the type of self and t are the same or if t is a supertype of the type of self.
- **self.oclAsType**(t:OclType):T
 - “cast” operator, returns self as an object of type T.

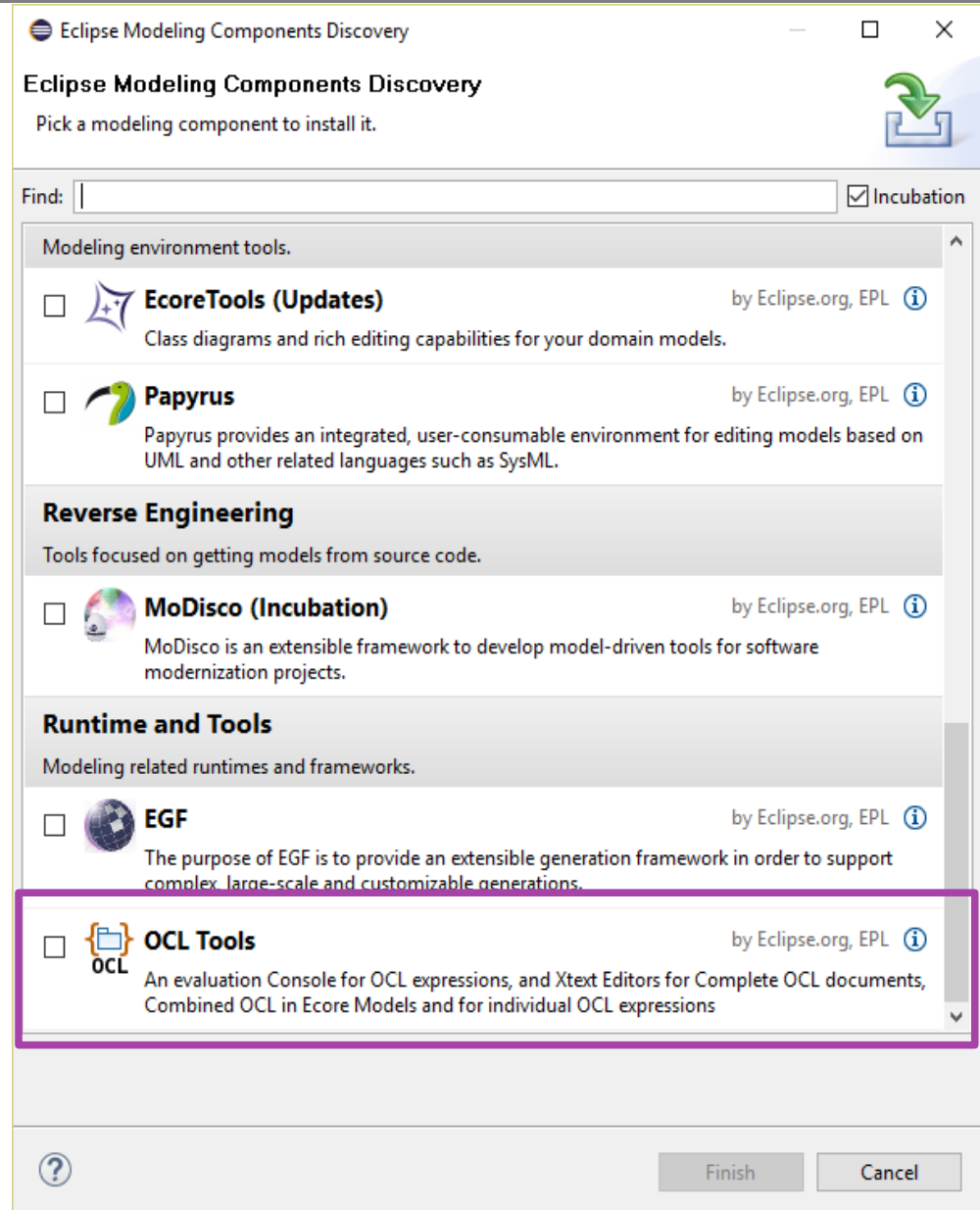
Further OCL Operations

- **self.oclIsTypeOf(t:OclType):Boolean**
 - returns true if the type of self and t are the same
- **self.oclIsKindOf(t:OclType):Boolean**
 - returns true if the type of self and t are the same or if t is a supertype of the type of self.
- **self.oclAsType(t:OclType):T**
 - “cast” operator, returns self as an object of type T.
- **allInstances()**
 - Operation on classes, interfaces, or enumerations
 - returns all instances of the type

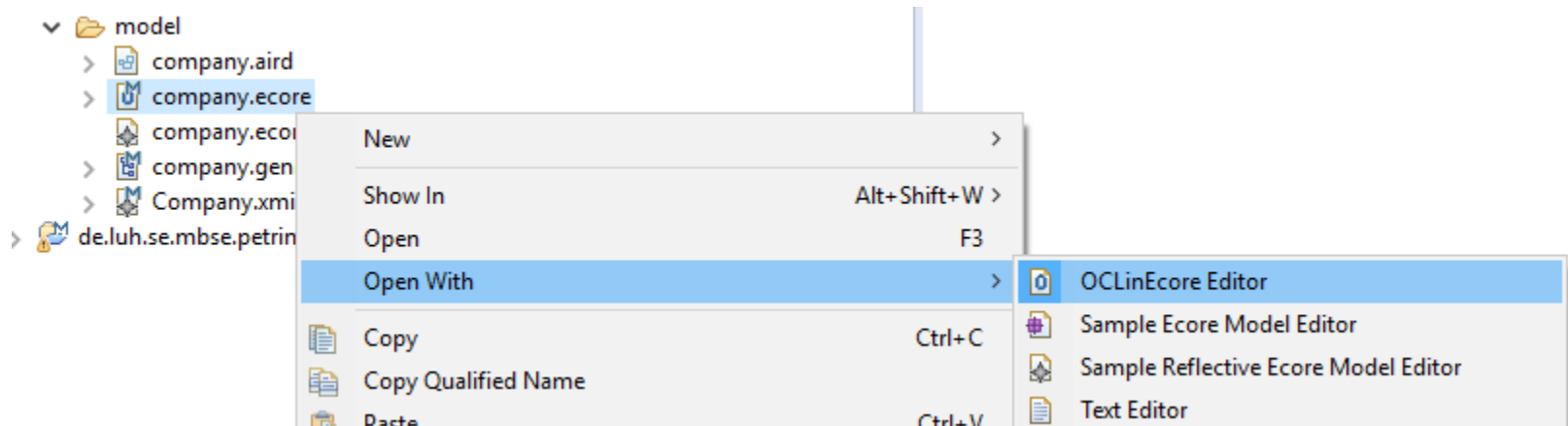
3.3. OCL in Ecore

Eclipse OCL Tools

- Installation:



- You can open .ecore files with the OCLEcore editor



OClinEcore Editor

Example: Company

company.ecore

```

1 import ecore : 'http://www.eclipse.org/emf/2002/Ecore' ;
2
3 package company : company = 'http://www.example.org/company'
4 {
5     class Company extends NamedElement
6     {
7         property department : Department[*] { ordered composes };
8     }
9     class Department extends NamedElement
10    {
11        property employee : Person[*] { ordered composes };
12        attribute ageSumOfEmployees : ecore::EInt[?] { derived readonly transient volatile }
13        {
14            initial: self.employee->iterate(p; sum:Integer = 0 | sum + p.age);
15        }
16    }
17    class NamedElement
18    {
19        attribute name : String[?];
20    }
21    class Person extends NamedElement
22    {
23        attribute age : ecore::EInt[?];
24        invariant AllEmployeesMustBeAdults: self.age >= 18;
25    }
26 }

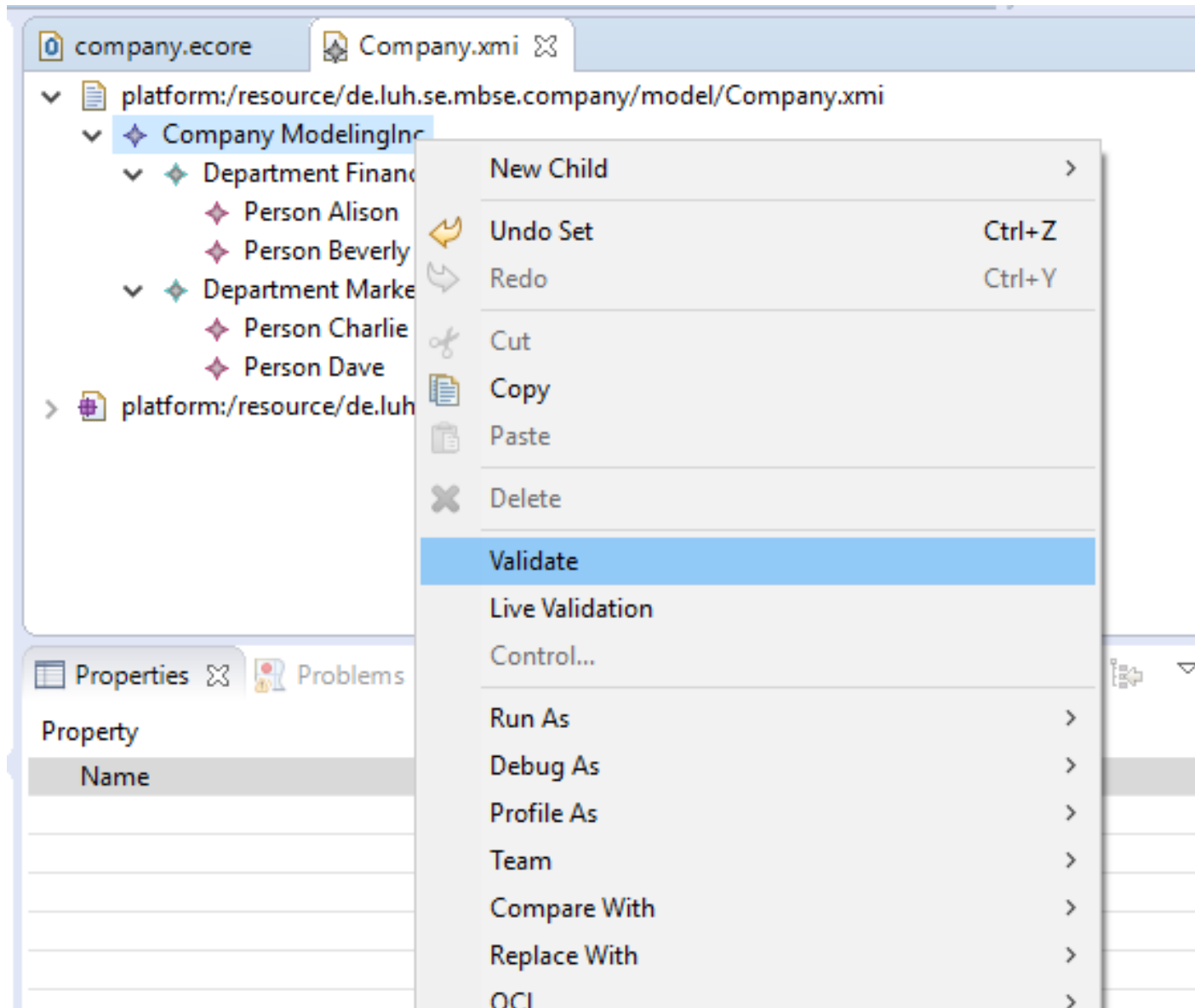
```

derived attribute

invariant

OClinEcore Editor

Example: Company



OClinEcore Editor

Example: Company

The screenshot displays the OCLinEcore Editor interface. The left sidebar shows the 'Model Explorer' with a tree structure of the project 'de.luh.se.mbse.company'. The main workspace shows the 'Company.xmi' file, which contains a 'Company ModelingInc' entity with two departments: 'Department Finance' (containing 'Person Alison' and 'Person Beverly') and 'Department Marketing'.

A 'Validation Problems' dialog box is open, indicating a validation error. The message states: 'Problems encountered during validation' and 'Reason: Diagnosis of Company ModelingInc'. The specific error is: 'The 'AllEmployeesMustBeAdults' constraint is violated on 'Person Dave''. The dialog box includes 'OK' and '<< Details' buttons.

Below the dialog box, the 'Property' tab is visible, showing a table with columns 'Name' and 'Value'. The table is currently empty.

At the bottom of the editor, the status bar indicates 'Object: Company ModelingInc'.

OCLinEcore Editor

Example: Company

The screenshot displays the OCLinEcore Editor interface. On the left, the **Model Explorer** shows a project structure for `de.luh.se.mbse.company`, including `Project Dependencies`, `src`, `JRE System Library [JavaSE-1]`, `Plug-in Dependencies`, `META-INF`, and a `model` folder containing `company.aird`, `company.ecore`, `company.ecore.oclas`, `company.genmodel`, and `Company.xmi`. The `Company.xmi` file is selected.

The main editor area shows the `company.ecore` file with a tree view of the model structure:

- `platform:/resource/de.luh.se.mbse.company/model/Company.xmi`
 - `Company ModelingInc`
 - `Department Finance`
 - `Person Alison`
 - `Person Beverly`
 - `Department Marketing`
 - `Person Charlie`
 - `Person Dave` (highlighted)
- `platform:/resource/de.luh.se.mbse.company/model/company.ecore`

At the bottom, the **Properties** view shows the properties of the selected `Person Dave` object:

Property	Value
Age	17
Name	Dave

A yellow callout bubble points to the `Age` property value `17`, stating: "validation shows invalid value".

OCLEcore Editor

Example: Company

The screenshot displays the OCLEcore Editor interface. The top toolbar includes icons for file operations and a 'Quick Access' search bar. The left sidebar contains a 'Model Explorer' with a search filter and a tree view of the project structure. The main workspace shows a hierarchical view of the 'company.ecore' project, with 'Department Marketing' selected. The bottom panel features a 'Properties' table and a 'Problems' section.

Model Explorer (Left Sidebar):

- type filter text
- de.luh.se.mbse.company
 - Project Dependencies
 - src
 - JRE System Library [JavaSE-1
 - Plug-in Dependencies
 - META-INF
 - model
 - company.aird
 - company.ecore
 - company.ecore.oclas
 - company.genmodel
 - Company.xmi
- de.luh.se.mbse.petrinet

Main Workspace (Company.xmi):

- platform:/resource/de.luh.se.mbse.company/model/Company.xmi
 - Company ModelingInc
 - Department Finance
 - Person Alison
 - Person Beverly
 - Department Marketing
 - Person Charlie
 - Person Dave
 - platform:/resource/de.luh.se.mbse.company/model/company.ecore

Properties Table (Bottom Panel):

Property	Value
Age Sum Of Employees	61
Name	Marketing

A yellow callout bubble points to the 'Marketing' value in the 'Name' row of the Properties table, containing the text: "interpretation of OCL derived value specifications on dynamic instance model".

Outline is not available.

OCLinEcore Editor

Example: Petri net

petrinet.ecore

```

1 import ecore : 'http://www.eclipse.org/emf/2002/Ecore' ;
2
3 package petrinet : petrinet = 'http://www.example.org/petrinet'
4 {
5     class PetriNet
6     {
7         property element : Element[*] { ordered composes };
8     }
9     abstract class Element;
10    abstract class Node extends Element
11    {
12        attribute name : String[?];
13    }
14    class Place extends Node
15    {
16        attribute initialMarkings : ecore::EInt[?];
17    }
18    class Transition extends Node;
19    class Arc extends Element
20    {
21        property source : Node[1];
22        property target : Node[1];
23        invariant NoArcsBetweenNodesOfTheSameKind:
24            ((self.source.ocIsKindOf(Place) and
25              self.target.ocIsKindOf(Transition))
26             or
27             (self.source.ocIsKindOf(Transition) and
28              self.target.ocIsKindOf(Place) ) );
29    }
30

```

- Formal, textual language for specifying queries and constraints on models with a MOF/UML metamodel
- Typed language
- No “programming”, no side-effects
- Tool support for EMF
- Used in other languages
 - we will see it again!