The Object Constraint Language

Jacques Robin

What is OCL? Definition and Role

- A textual specification language to adorn UML and MOF diagrams and make them far more semantically precise and detailed
- OCL2 integral part of the UML2 standard
- OCL complements UML2 diagrams to make UML2:
  - A domain ontology language that is self-sufficient at the knowledge level to completely specify both structure and behaviors
  - A complete input for the automated generation of a formal specification at the formalization level to be verified by theorem provers
  - A complete input for the automated generation of source code at the implementation level to be executed by a deployment platform
- OCL forms the basis of model transformation languages
  - such as Atlas Transformation Language (ATL) or Query-View-Transform (QVT)
  - which declaratively specify through rewrite transformation rules the automated generation of formal specifications and implementations from a knowledge level ontology
  - OCL expressions are used in the left-hand and right-hand sides of such rules
  - To specify objects to match in the source ontology of the transformation
  - To specify objects to create in the target formal specification or code of the transformation

What is OCL? Characteristics

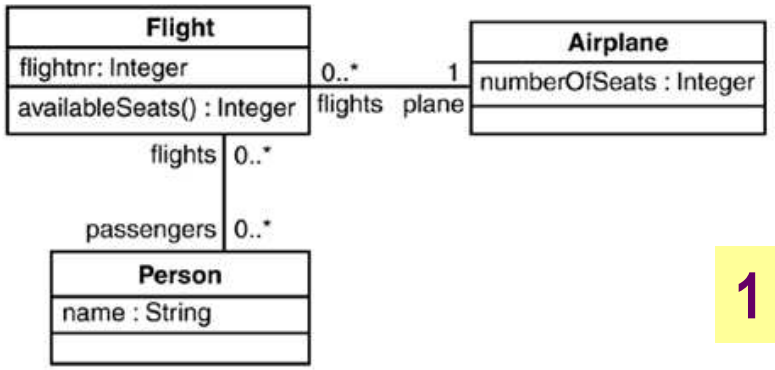
- Formal language with well-defined semantics based on set theory and first-order predicate logic, yet free of mathematical notation and thus friendly to mainstream programmers
- Object-oriented functional language: constructors syntactically combined using functional nesting and object-oriented navigation in expressions that take objects and/or object collections as parameters and evaluates to an object and/or an object collection as return value
- Strongly typed language where all expression and sub-expression has a well-defined type that can be an UML primitive data type, a UML model classifier or a collection of these
- Semantics of an expression defined by its type mapping
- Declarative language that specifies what properties the software under construction must satisfy, not how it shall satisfy them
- Side effect free language that cannot alter model elements, but only specify relations between them (some possibly new but not created by OCL expressions)
- Pure specification language that cannot alone execute nor program models but only describe them
- Both a constraint and query language for UML models and MOF meta-models

What is OCL?
How does it complement UML?

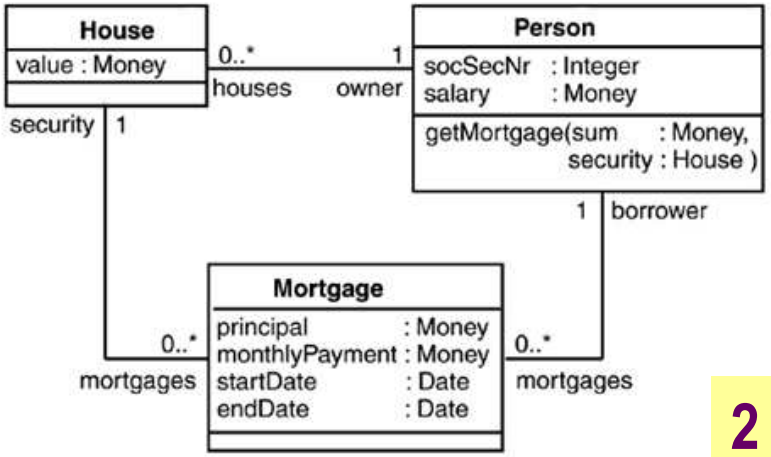
- Structural adornments:
  - Specify complex invariant constraints (value, multiplicity, type, etc) between multiple attributes and associations
  - Specify deductive rules to define derived attributes, associations and classes from primitive ones
  - Disambiguates association cycles
- Behavioral adornments:
  - Specify operation pre-conditions
  - Specify write operation post-conditions
  - Specify read/query operation bodies
  - Specify read/query operation initial/default value

OCL: Motivating Examples

- Diagram 1 allows Flight with unlimited number of passengers
- No way using UML only to express restriction that the number of passengers is limited to the number of seats of the Airplane used for the Flight
- Similarly, diagram 2 allows:
  - A Person to Mortgage the house of another Person
  - A Mortgage start date to be after its end date
  - Two Persons to share same social security number
  - A Person with insufficient income to Mortgage a house



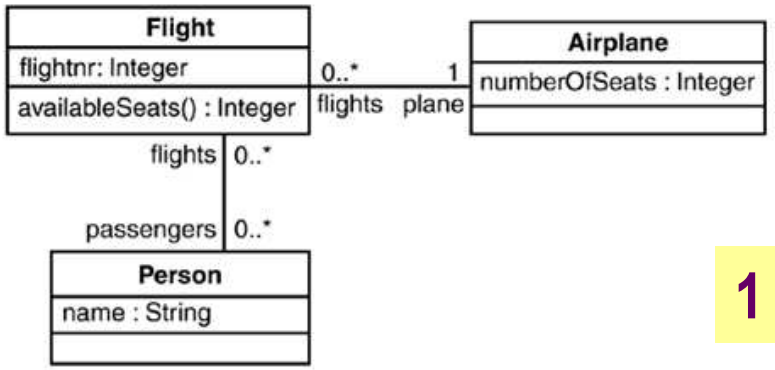
1



2

OCL: Motivating Examples

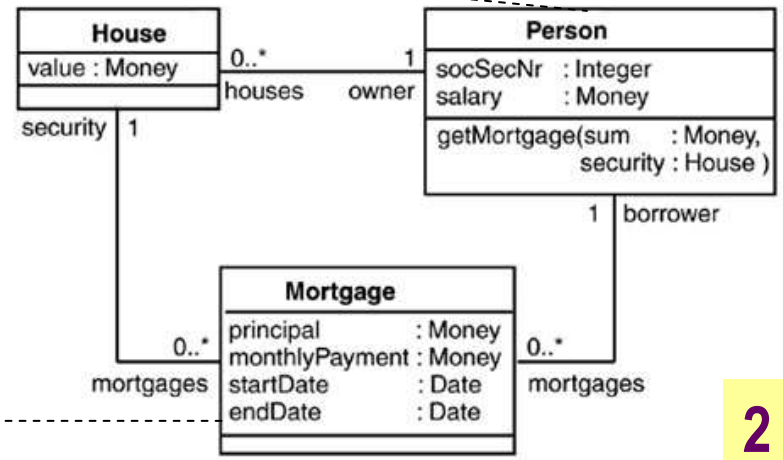
```
context Flight
inv: passengers -> size()
     <= plane.numberOfSeats
```



1

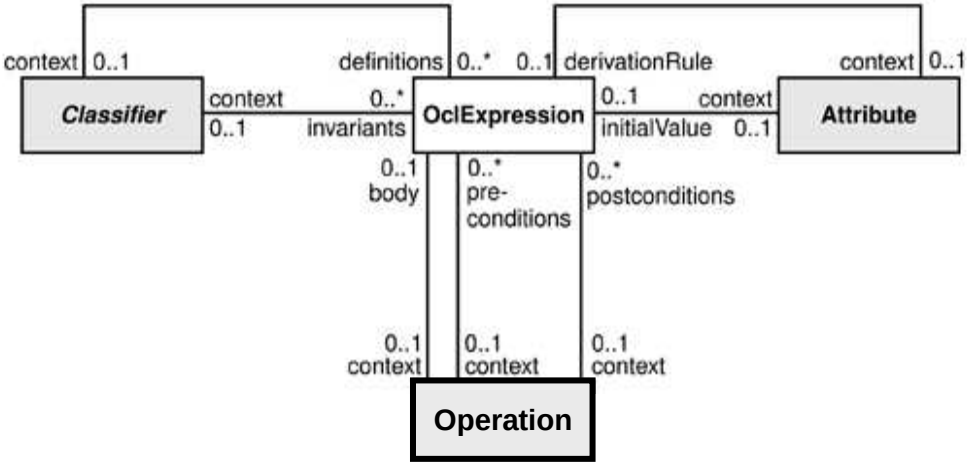
```
context Person
inv: Person::allInstances() -> isUnique(socSecNr)

context Person::getMortgage(sum:Money,security:House)
pre: self.mortgages.monthlyPayment -> sum() <= self.salary * 0.3
```

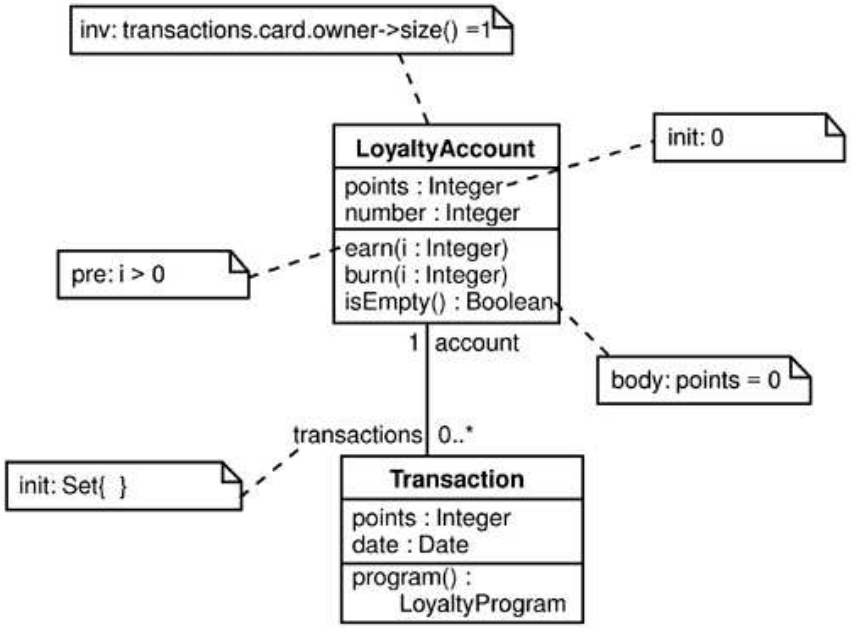
```
context Mortgage
inv: security.owner = borrower
inv: startDate < endDate
```



2

OCL Expression Contexts





OCL Contexts: Specifying Class Invariants

The context of an invariant constraint is a class

When it occurs as navigation path prefix, the self keyword can be omitted:

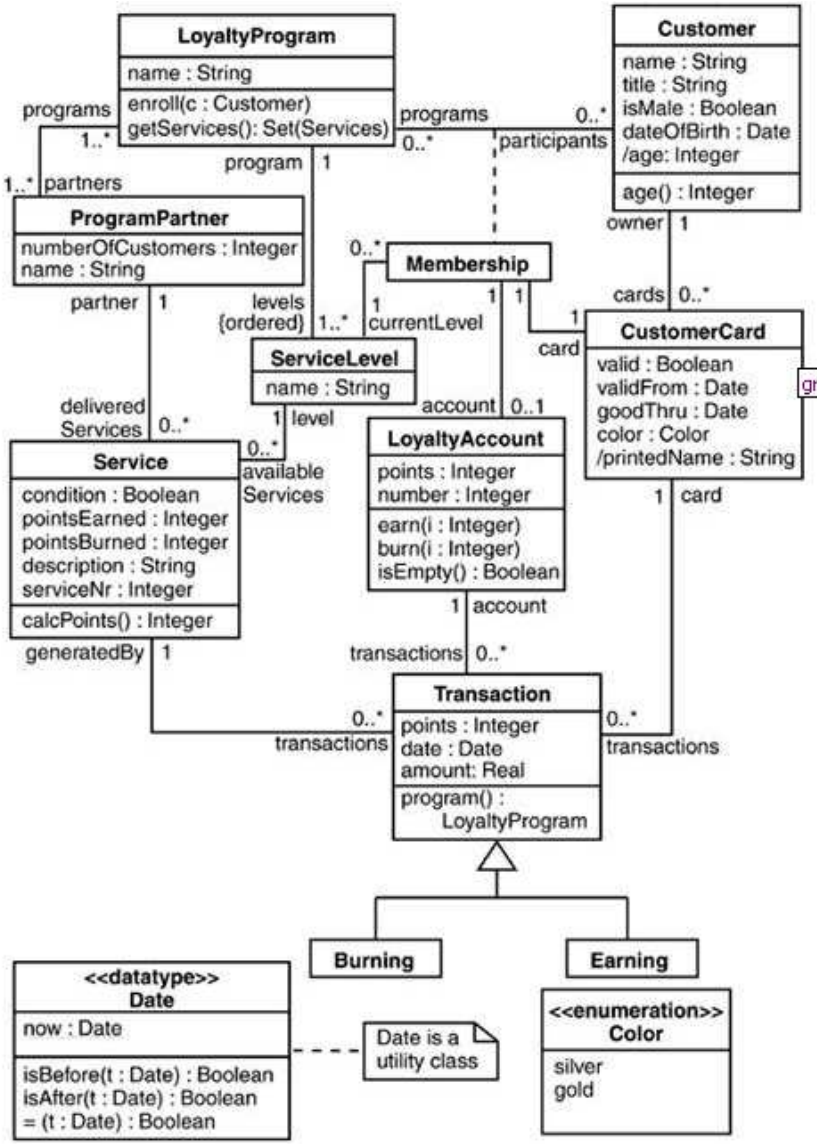
- context Customer inv: self.name = 'Edward'
- context Customer inv: name = 'Edward'

Invariants can be named:

- context Customer inv myInvariant23:
  self.name = 'Edward'
- context LoyaltyAccount
  inv oneOwner: transaction.card.owner
  -> asSet() -> size() = 1

In some context self keyword is required:

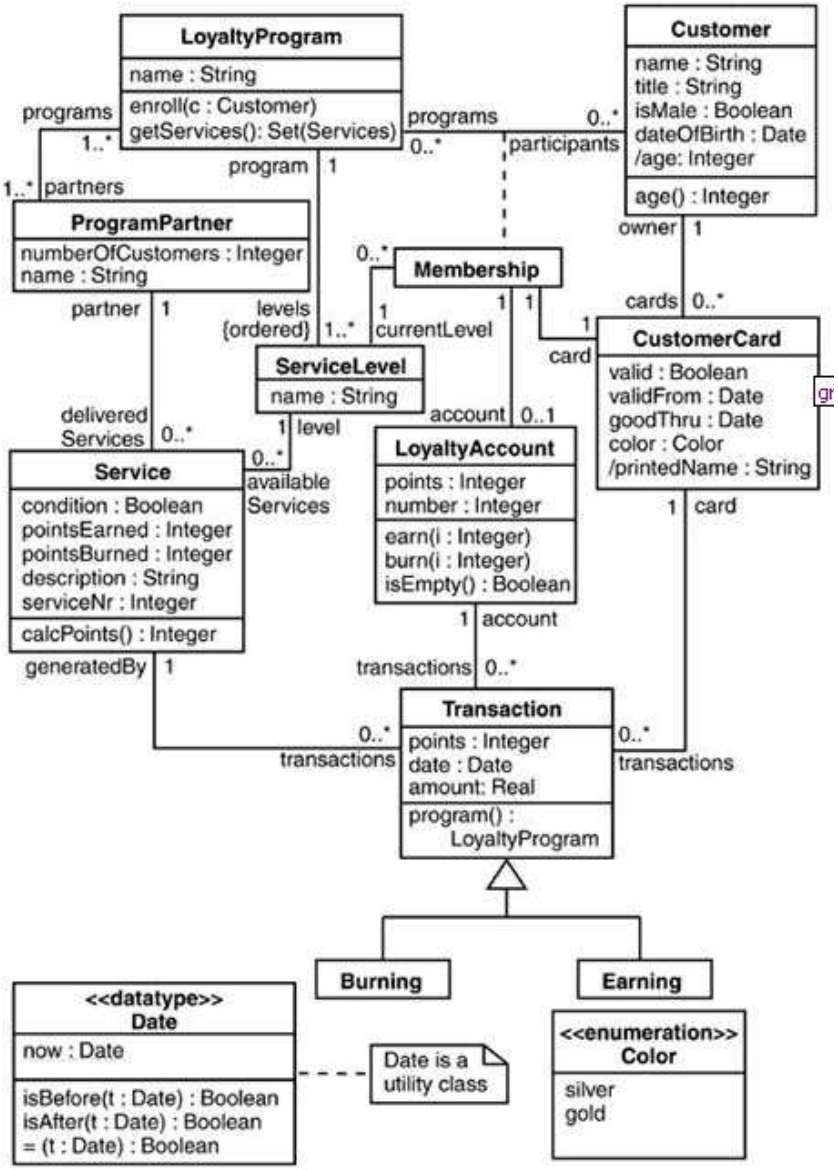
- context Membership
  inv:
  participants.cards.Membership.includes(self)



Specifying Default Attribute Values

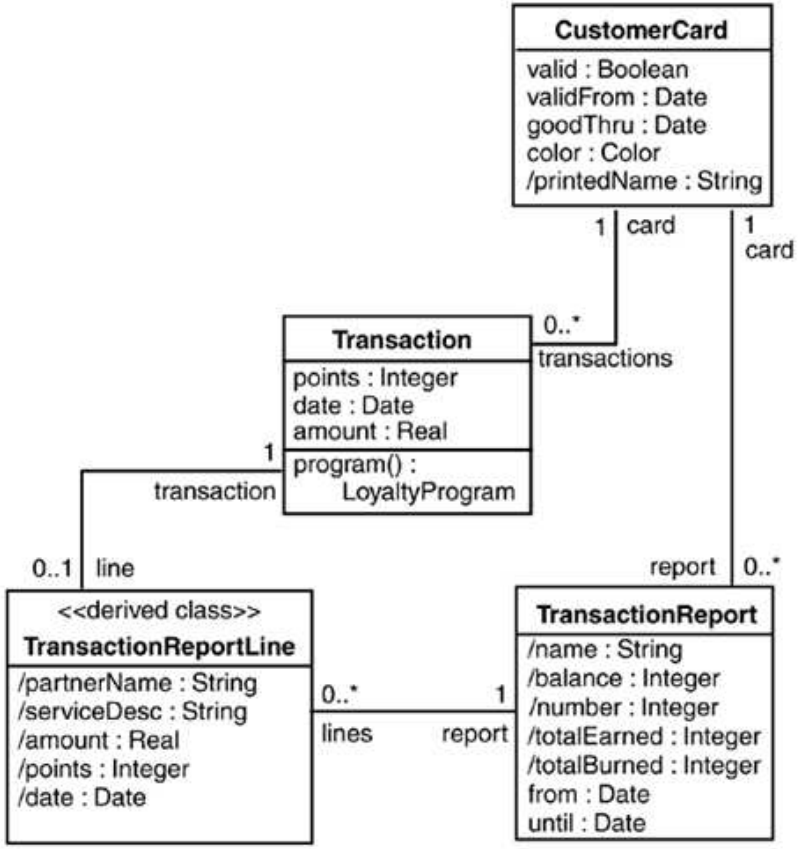
Initial values:

- context LoyaltyAccount::points : integer
  init: 0
- context LoyaltyAccount::transactions
  : Set(Transaction)
  init: Set{}



Specifying Attribute Derivation Rules

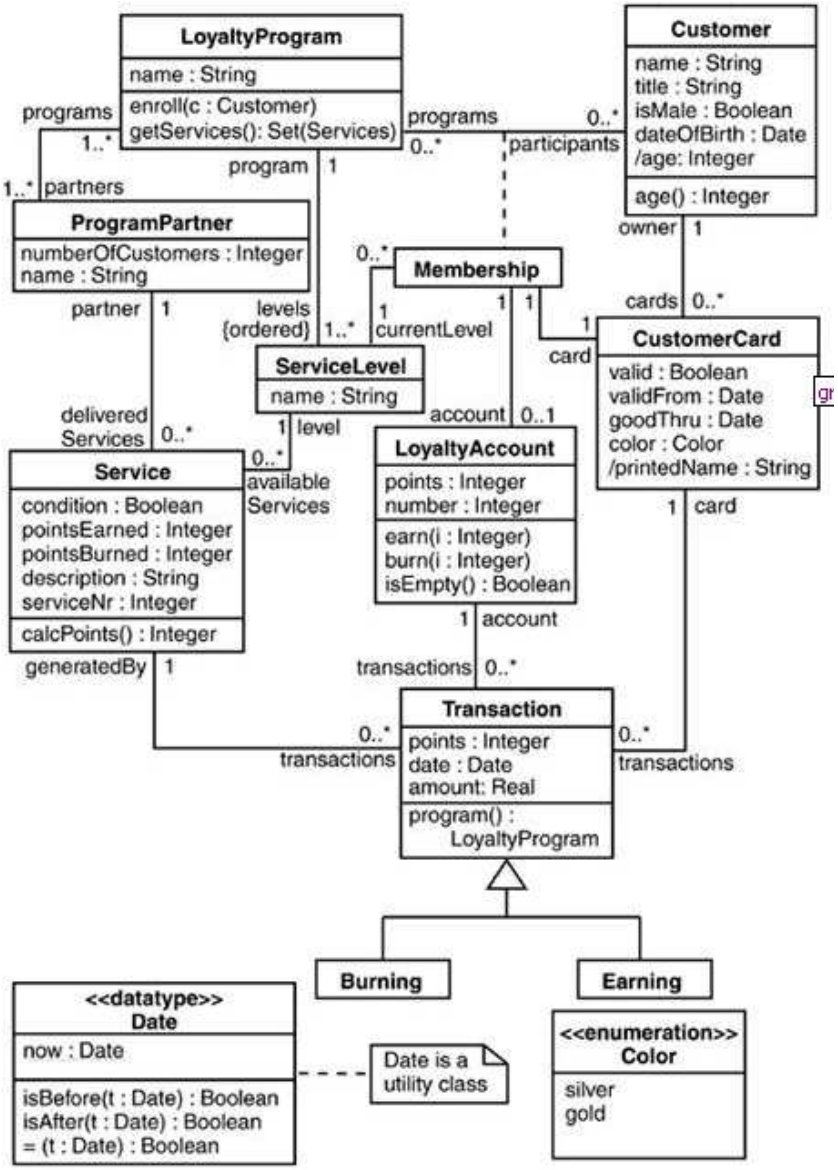
- context CustomerCard::printedName
  derive: owner.title.concat(' ').concat(owner.name)
- context TransactionReportLine: String
  derive self.date = transaction.date
- ...
- context TransactionReport
  inv dates: lines.date -> forAll(d |
  d.isBefore(until) and d.isAfter(from))
- ...



Specifying Query Operation Bodies

Query operations:

- context LoyaltyAccount::getCustomerName()
  : String
  body: Membership.card.owner.name
- context LoyaltyProgram::getServices():
  Set(Services)
  body: partner.deliveredServices -> asSet()



Specifying Operations Pre and Post Conditions

- context LoyaltyAccount::isEmpty(): Boolean
  pre: -- none
  post: result = (points = 0)

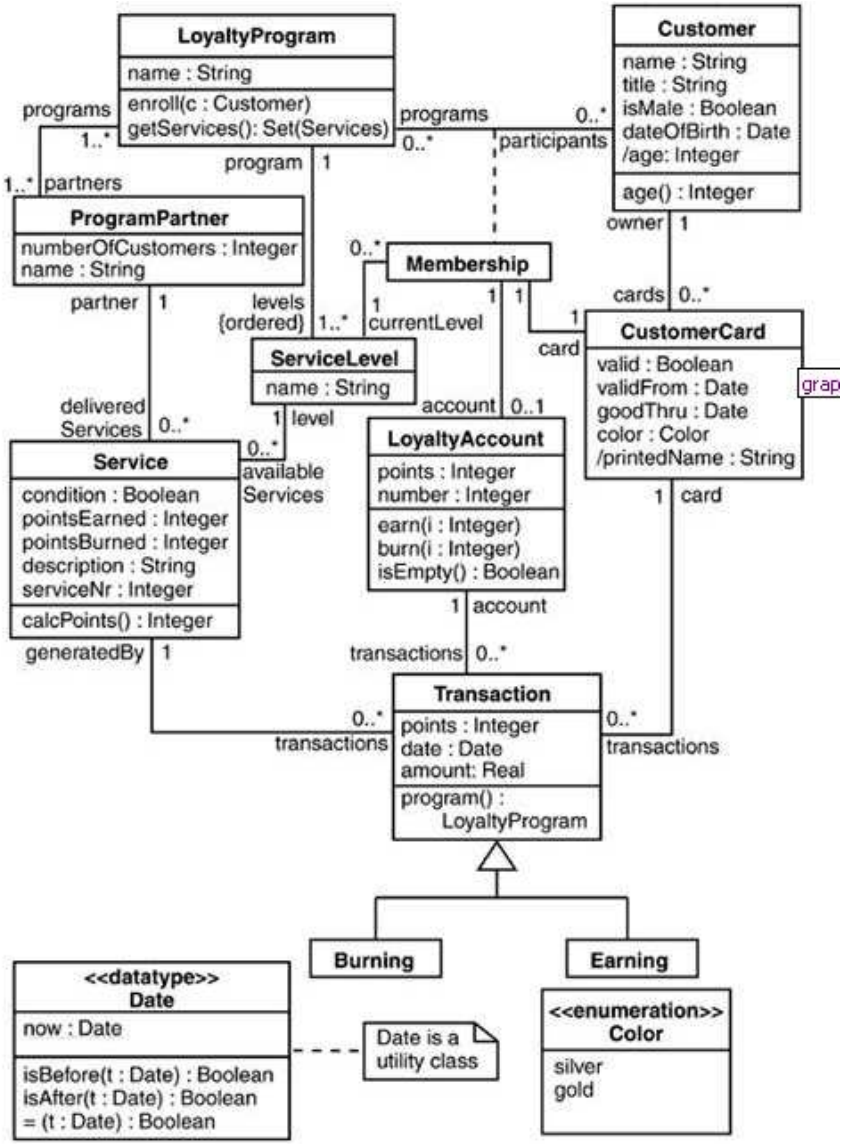
Keyword @pre used to refer in post-condition to the value of a property before the execution of the operation:

- context LoyaltyProgram::enroll(c:Customer)
  pre: c.name <> ''
  post: participants = participants@pre ->
  including(c)

Keyword oclIsNew used to specify creation of a new instance (objects or primitive data):

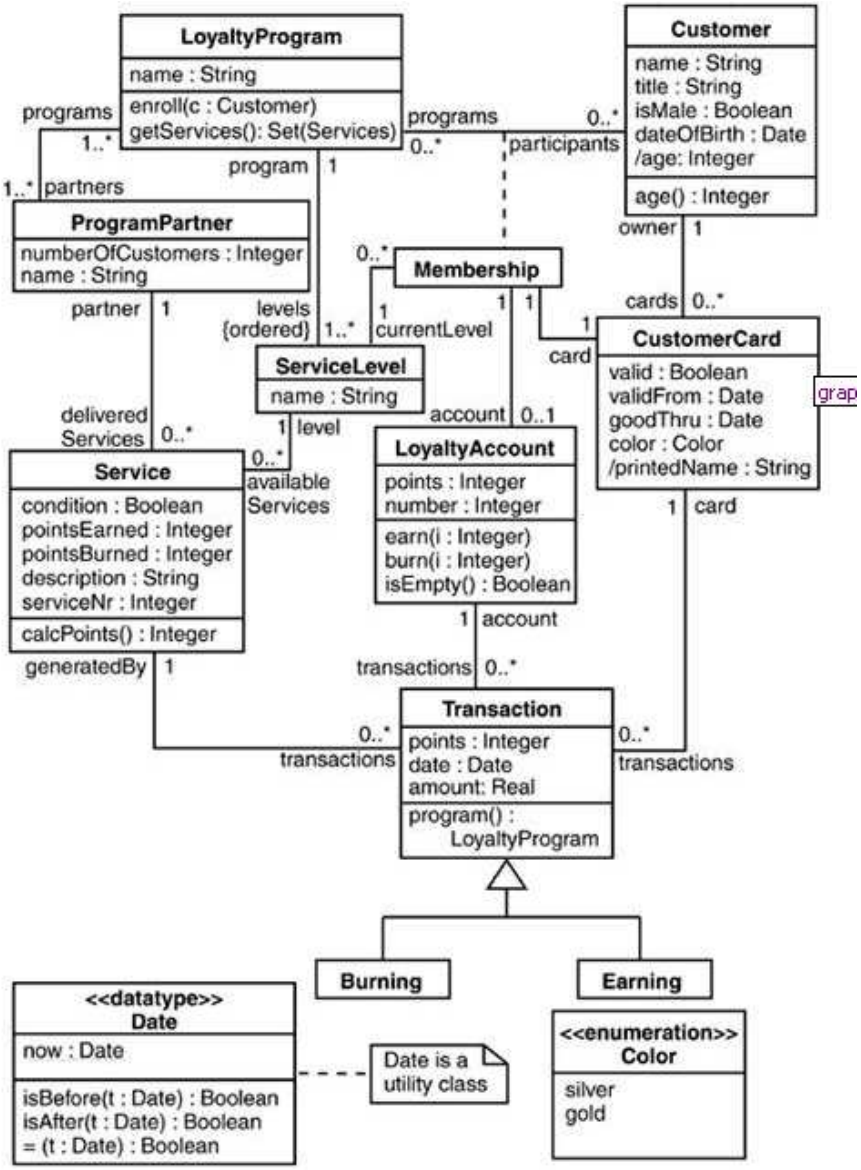
- context LoyaltyProgram::
  enrollAndCreateCustomer(n:String,d:Date):Customer
  post: result.oclIsNew() and result.name = n and
  result.dateOfBirth = d and
  participant -> includes(result)

oclIsNew only specifies that the operation created the new instance, but not how it did it which cannot be expressed in OCL



Association Navigation

- Abbreviation of collect operator that creates new collection from existing one, for example result of navigating association with plural multiplicity:
  - context LoyaltyAccount
    inv: transactions -> collect(points) ->
    exists(p:Integer | p=500)
  - context LoyaltyAccount
    inv: transactions.points ->
    exists(p:Integer | p=500)
- Use target class name to navigate roleless association:
  - context LoyaltyProgram
    inv: levels ->
    includesAll(Membership.currentLevel)
- Call UML model and OCL library operations



Generalization Navigation

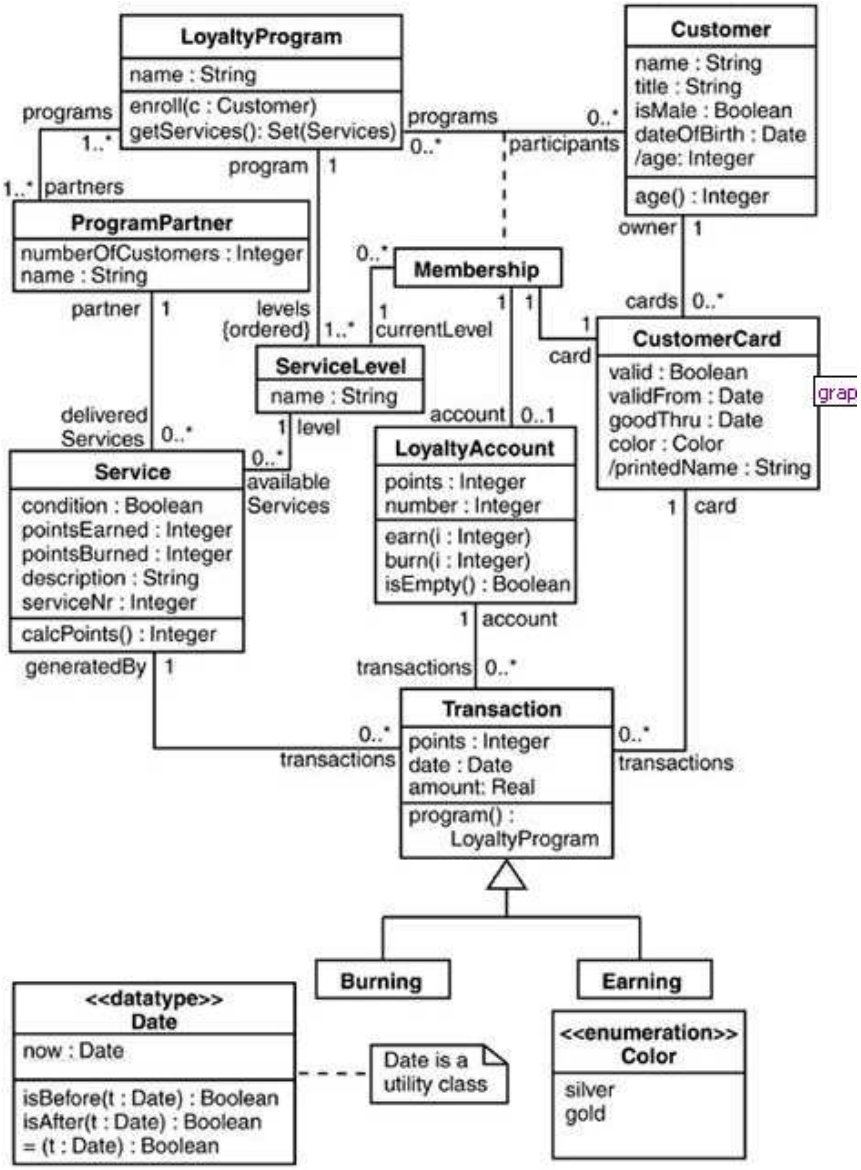
- OCL constraint to limit points earned from single service to 10,000
- Cannot be correctly specified using association navigation:

```
context ProgramPartner
   inv totalPoints:
        deliveredServices.transactions
        .points -> sum() < 10,000
```

adds both Earning and Burning points

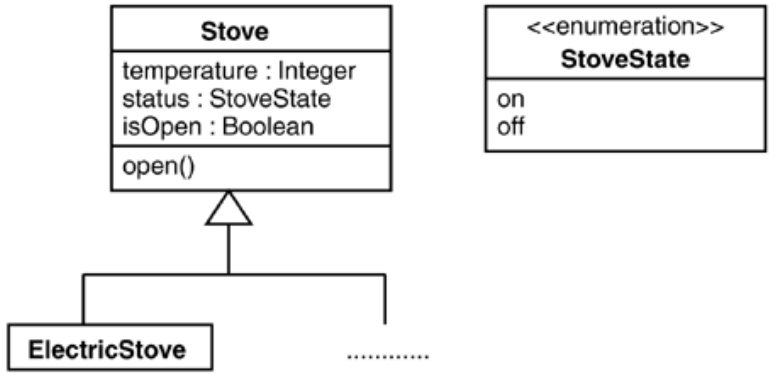
- Operator oclIsTypeOf allows hybrid navigation following associations and specialization links

```
context ProgramPartner
   inv totalPoints:
        deliveredServices.transactions
        -> select(oclIsTypeOf(Earning))
        .points -> sum() < 10,000
```



OCL Visibility and Inheritance

- By default, OCL expressions ignore attribute visibility
  - *i.e.*, an expression that access a private attribute from another class is not syntactically rejected
- OCL constraints are inherited down the classifier hierarchy
- OCL constraints redefined down the classifier hierarchy must follow substituability principle
  - Invariants and post-condition can only become more restrictive
  - Preconditions can only become less restrictive



Examples violating substituability principle:

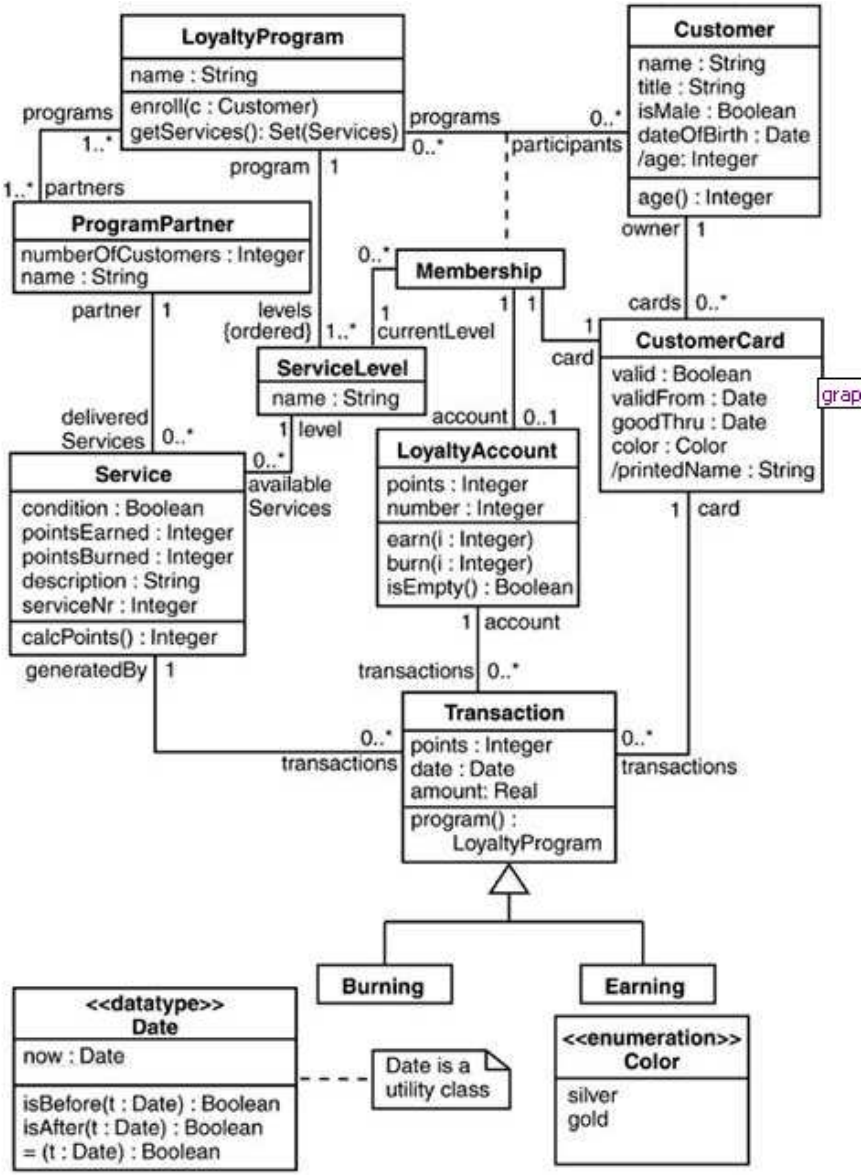
```
context Stove inv: temperature <= 200

context ElectricStove
inv: temperature <= 300

context Stove::open()
pre: status = StoveState::off
post: status = StoveState::off and isOpen

context ElectricStove::open()
pre: status = StoveState::off and
   temperature <= 100
post: isOpen
```

OCL Expressions: Local Variables

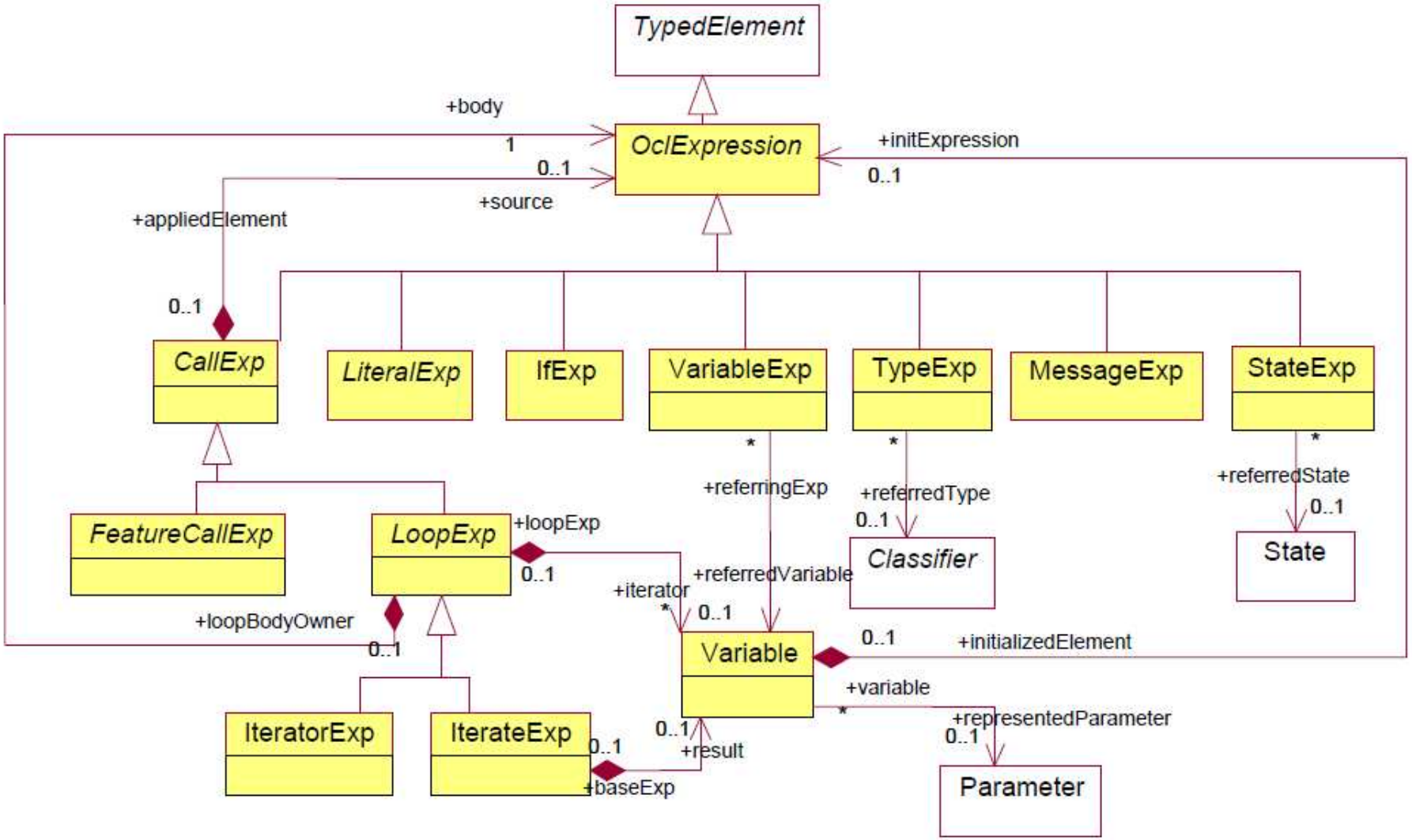
- Let constructor allows creation of aliases for recurring sub-expressions

```
context CustomerCard
inv: let correctDate : Boolean =
       validFrom.isBefore(Date::now) and
       goodThru.isAfter(Date::now)
     in if valid then correctDate = false
        else correctDate = true
        endif
```

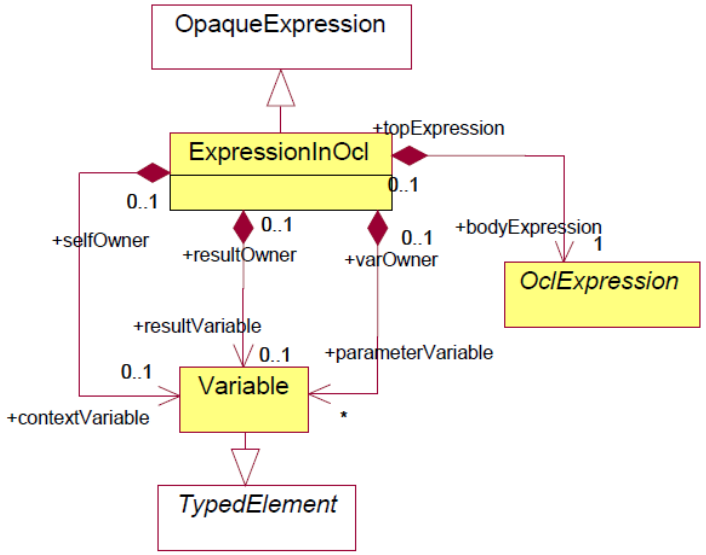
- Syntactic sugar that improves constraint legibility

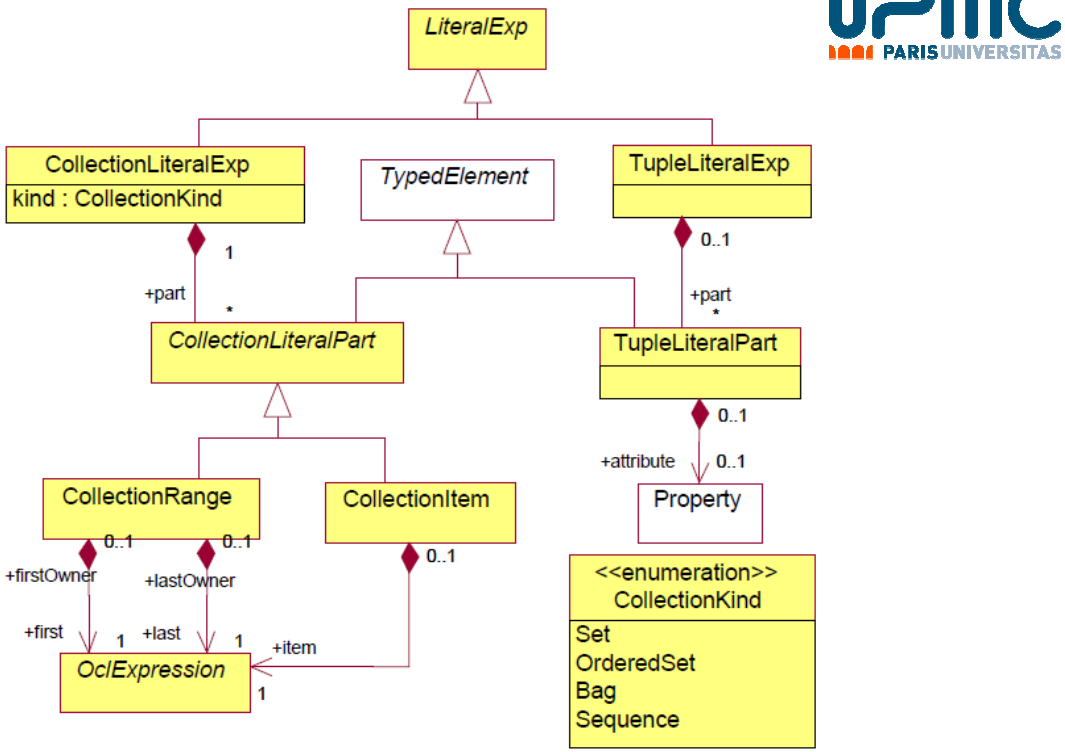


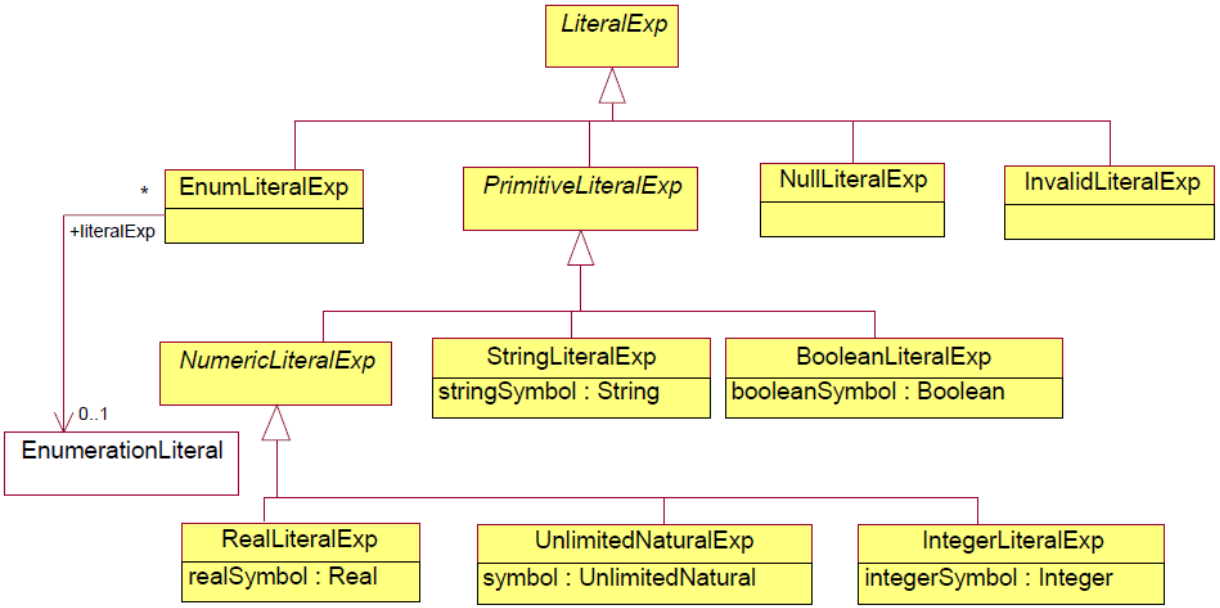
OCL Metamodel



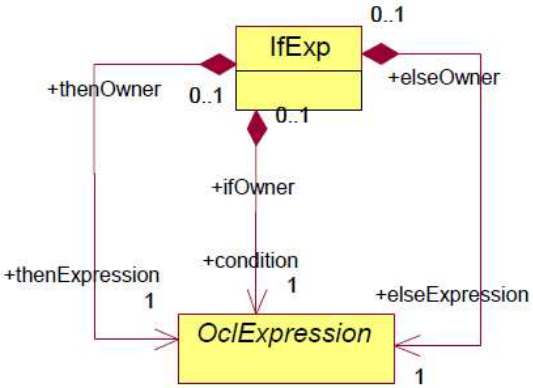
OCL Metamodel

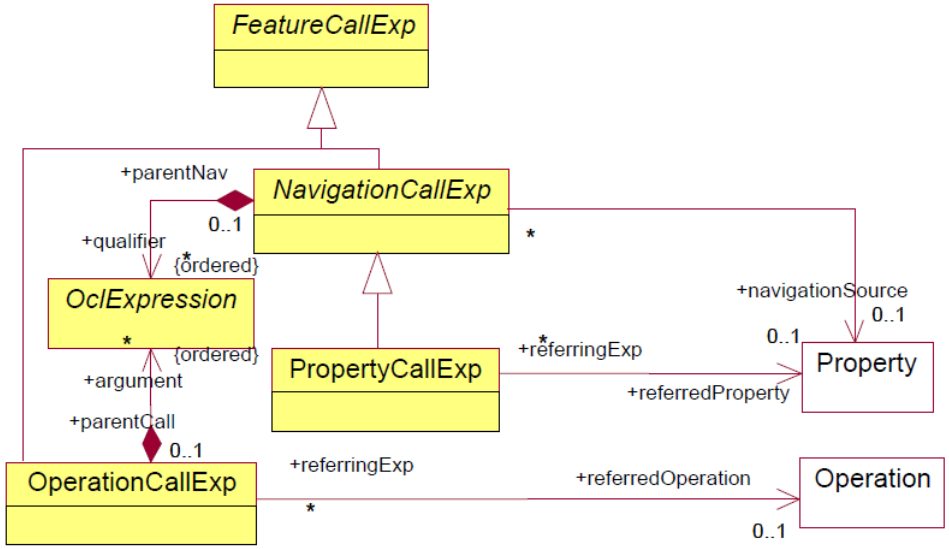


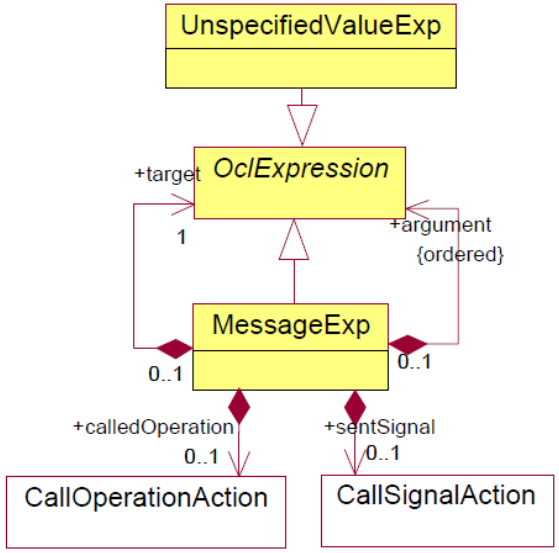


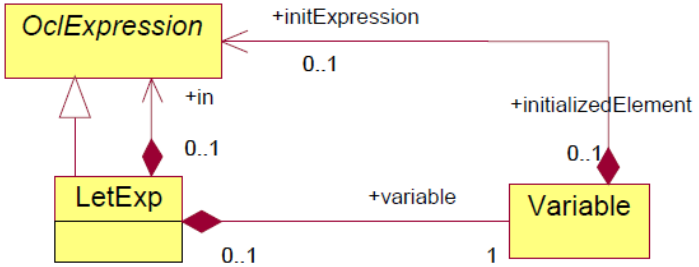


OCL Metamodel

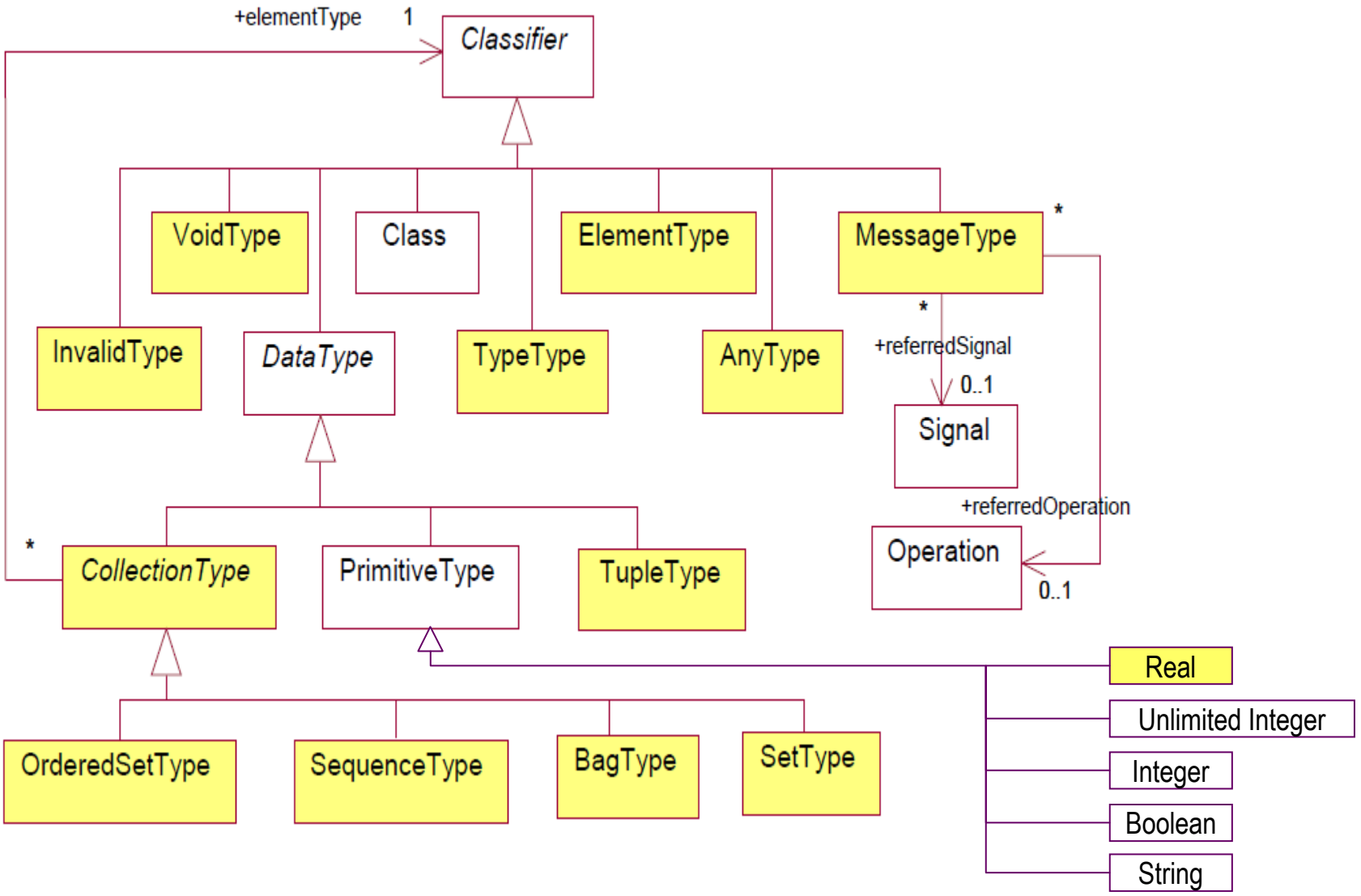








OCL Type System



OCL Types: Collections

- Collection constants can be specified in extension:
  - Set{1, 2, 5, 88}, Set{'apple', 'orange', 'strawberry'}
  - OrderedSet{'black', 'brown', 'red', 'orange', 'yellow', 'green', 'blue', 'purple'}
  - Sequence{1, 3, 45, 2, 3}, Bag{1, 3, 4, 3, 5}
- Sequence of consecutive integers can be specified in intension:
  - Sequence{1..4} = Sequence{1,2,3,4}
- Collection operations are called using -> instead of .
- Collection operations have value types:
  - They do not alter their input only output a new collection which may contain copies of some input elements
- Most collections operations return flattened collections
  - *ex,* flatten{Set{1,2},Set{3,Set{4,5}}} = Set{1,2,3,4,5}
- Operation collectNested must be used to preserve embedded sub-structures
- Navigating through several associations with plural multiplicity results in a bag

OCL Library: Generic Operators

- Operators that apply to expressions of any type
- Defined at the top-level of OclAny

Expression	Result Type
object = (object2 : OclAny)	Boolean
object <> (object2 : OclAny)	Boolean
object.oclIsUndefined()	Boolean
object.oclIsKindOf(type : OclType)	Boolean
object.oclIsTypeOf(type : OclType)	Boolean
object.oclIsNew()	Boolean
object.oclInState()	Boolean
object.oclAsType(type : OclType)	type
object.oclInState(str: StateName)	Boolean
type::allInstances()	Set(type)

OCL Library: Primitive Type Operators

- Boolean: host, parameter and return type boolean
  - Unary: not
  - Binary: or, and, xor, =, <>, implies
  - Ternary: if-then-else
- Arithmetic: host and parameters integer or real
  - Comparison (return type boolean): =, <>, <, > <=, >=,
  - Operations (return type integer or real): +, -, *, /, mod, div, abs, max, min, round, floor
- String: host string
  - Comparison (return type boolean): =, <>
  - Operation: concat(String), size(), toLower(), toUpper(), substring(n:integer,m:integer)

OCL Library: Generic Collection Operators

Operation	Description
count(object)	The number of occurrences of the object in the collection
excludes(object)	True if the object is not an element of the collection
excludesAll(collection)	True if all elements of the parameter collection are not present in the current collection
includes(object)	True if the object is an element of the collection
includesAll(collection)	True if all elements of the parameter collection are present in the current collection
isEmpty()	True if the collection contains no elements
notEmpty()	True if the collection contains one or more elements
size()	The number of elements in the collection
sum()	The addition of all elements in the collection. The elements must be of a type supporting addition

Operation	Description
any(expr)	Returns a random element of the source collection for which the expression *expr* is true
collect(expr)	Returns the collection of objects that result from evaluating *expr* for each element in the source collection
collectNested(expr)	Returns a collection of collections that result from evaluating *expr* for each element in the source collection
exists(expr)	Returns true if there is at least one element in the source collection for which *expr* is true
forAll(expr)	Returns true if *expr* is true for all elements in the source collection
isUnique(expr)	Returns true if *expr* has a unique value for all elements in the source collection
iterate(...)	Iterates over all elements in the source collection
one(expr)	Returns true if there is exactly one element in the source collection for which *expr* is true
reject(expr)	Returns a subcollection of the source collection containing all elements for which *expr* is false
select(expr)	Returns a subcollection of the source collection containing all elements for which *expr* is false
sortedBy(expr)	Returns a collection containing all elements of the source collection ordered by *expr*

OCL Library: Specialized Collection Operators

Operation	Set	OrderedSet	Bag	Sequence
=	X	X	X	X
< >	X	X	X	X
-	X	X	-	-
append(object)	-	X	-	X
asBag()	X	X	X	X
asOrderedSet()	X	X	X	X
asSequence()	X	X	X	X
asSet()	X	X	X	X
at(index)	-	X	-	X
excluding(object)	X	X	X	X
first()	-	X	-	X
flatten()	X	X	X	X
including(object)	X	X	X	X
indexOf(object)	-	X	-	X
insertAt(index, object)	-	X	-	X
intersection(coll)	X	-	X	-
last()	-	X	-	X
prepend(object)	-	X	-	X
subOrderedSet(lower, upper)	-	X	-	-
subSequence(lower, upper)	-	-	-	X
symmetricDifference(coll)	X	-	-	-
union(coll)	X	X	X	X

OCL Constraints *vs.* UML Constraints