

# OCL

- Object Constraint Language
- Official part of UML
- Strongly typed, declarative specification of system properties
- Assertions + collection classes + UML diagram navigation
- Supported by Rational Rose, ArgoUML, Poseidon/Octopus, Enterprise Architect

# Why?

- UML diagrams, as a graphical language, are limited in what they can express
  - Structural relationships, behavioral descriptions
  - The language needs a mechanism for specifying precise semantics
- OCL extends UML with
  - Class invariants
  - Operation pre and post conditions
  - Guards on state-machine transitions

# OCL Overview

- Pure expression language
  - Focus on values; no side effects expressible
- Declarative not procedural
  - No control flow mechanisms
- Strongly typed
  - Built-in types plus types introduced in diagrams
- Highest level mechanism is the *constraint*

# Uses of OCL

- As a query language
- To specify invariants on classes and types in the class model
- To specify type invariant for Stereotypes
- To describe pre- and post conditions on Operations and Methods
- To describe Guards
- To specify target (sets) for messages and actions
- To specify constraints on operations
- To specify derivation rules for attributes for any expression over a UML model

# Syntax

```
context <identifier> <constraintType>:  
<Boolean expression>
```

- **context** is a keyword indicating the start of a new constraint
- <identifier> is a class or operation name
  - The current position from which relative path names are derived
- <constraintType> is **inv**, **pre**, or **post**
- <Boolean expression> is usually an equation asserting a property

# Invariants

- Statement of a property that is always true
  - Except perhaps in the middle of the execution of an operation
- Expresses a key system requirement
- **inv** keyword
- Might be an essential relationship among the values of the attributes of an object
- Might express a relationship between classes
- *The student registration system never allows a student to be registered for two classes that meet at the same time*
- **context** Company  
**inv:** `self.numberOfEmployees > 50`

# Pre and Post Conditions

- UML operation semantics can be expressed using **pre** and **post** condition constraints
  - The **pre** condition says what must be true for the operation to meaningfully take place
  - The **post** condition expresses what is guaranteed to be true after the operation completes
    - About the return value
    - About any state changes (e.g. instance variables)
    - **@pre** can be used to distinguish the value of an instance variable before an operation from the value afterwards

# Pre and Post Conditions - 2

- *The argument to the square root routine must be non-negative*
- *The square of the computed result must equal the argument*

```
context Person::income(d : Date) :  
    Integer post: result = 5000
```

# OCL Built-in Primitive Types

<b>type</b>	<b>values</b>	<b>operations</b>
Boolean	true, false	and, or, xor, not, implies, if-then-else
Integer	1, -5, 2, 34, 26524, ...	*, +, -, /, abs()
Real	1.5, 3.14, ...	*, +, -, /, floor()
String	'to be or not to be...'	toUpper(), concat()

# OCL Keywords

<b>keyword</b>	<b>description</b>
<code>inv, pre, post</code>	Introduces constraints
<code>if-then-else-endif</code>	Conditional expression
<code>not, or, and, xor, implies</code>	Boolean operators
<code>package, endpackage</code>	Packages
<code>context</code>	Modifies use of names
<code>def</code>	Global definition
<code>let, in</code>	Local definition
<code>derive</code>	Attribute derivation
<code>init</code>	Initial value description
<code>attr</code>	?
<code>oper</code>	?
<code>result</code>	Result value from an operation

# let Clause

- A `let` clause is a way of introducing an abbreviation
- It has two parts, both expressions
  - The first part between indicates one or more names and binds values to them
  - The second part is like a local block; it limits the scope of where the bindings apply
- ```
let income : Integer = self.job.salary->sum() in
  if isUnemployed then income < 100
    else income >= 100
endif
```

# Navigation

- OCL constraints are associated with class model diagrams
  - They can also be associated with statecharts
- Each constraint specifies a context
  - A specific class or method relative to which other diagram elements can be designated
- Navigation is done by walking through a diagram from the context class to another diagram element using intermediary relationship lines and class rectangles
  - Each step adds a name to a list, separated by periods

# Multiplicity

- When an association has multiplicity greater than one, the result of a traversal is a collection (set, bag or sequence)
- If an operation is performed on that association, the  $\rightarrow$  symbol is used, rather than the period, to separate the collection from the operation

# Collections

- OCL features four kinds of collections that support associations with many-to-many and one-to-many multiplicities
- The parent class **Collection** provides features that hold true of all collections
  - **size**, **includes**, **count**, **includesAll**, **isEmpty**, **notEmpty**, **sum**, **exists**, **forAll** and **iterate**
- **Set**, **Bag**, (multiset) and **Sequence** (ordered multiset) provide specialized operations

# Other OCL Features

- Enumerations
- Automatic flattening
- Access to the UML metamodel
- Tuple expression for dealing with structs/ records
- Message expression denoting the sending of a signal or the invoking of an operation