Expressions: Standard library (2)

- **Finite quantification**
  - c`->forAll(`i : T | e`)` = c`->iterate(`i : T; a : `Boolean` = `true` | a **and** e`)`
  - c`->exists(`i : T | e`)` = c`->iterate(`i : T; a : `Boolean` = `false` | a **or** e`)`
- Selecting values
  - c`->any(`i : T | e`)` — some element of c satisfying e
  - c`->select(`i : T | e`)` — all elements of c satisfying e
- Collecting values
  - c`->collect(`i : T | e`)` — collection of elements with e applied to each element of c
  - c`.`p — collection of elements $v.p$ for each v in c (short-hand for `collect`)

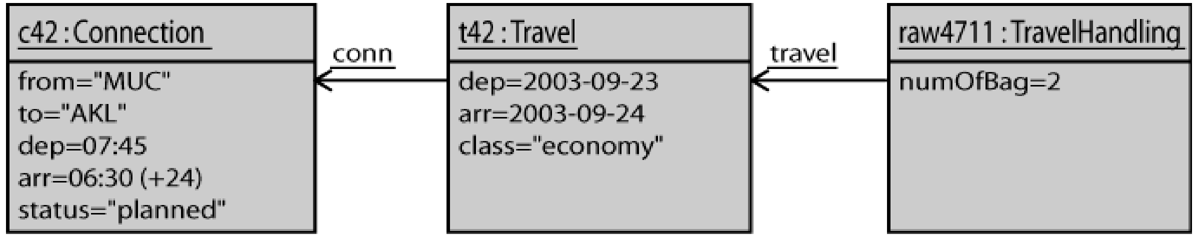
C`.allInstances()`	all current instances of classifier C
o`.oclIsInState(`s`)`	is o currently in state machine state s?
v`.oclIsUndefined()`	is value v null or invalid?
v`.oclIsInvalid()`	is value v invalid?

Evaluation

- Strict evaluation with some exceptions
    - `(`**`if`**` (1/0 = 0) `**`then`**` 0.0 `**`else`**` 0.0 `**`endif`**`).oclIsInvalid() = true`
    - `(1/0).oclIsInvalid() = true`
  - Short-cut evaluation for **and**, **or**, **implies**
    - `(1/0 = 0.0) `**`and`**` false = false`
    - `true `**`or`**` (1/0 = 0.0) = true`
    - `false `**`implies`**` (1/0 = 0.0) = true`
    - `(1/0 = 0.0) `**`implies`**` true = true`
    - **`if`**` (0 = 0) `**`then`**` 0.0 `**`else`**` 1/0 `**`endif`**` = 0.0`

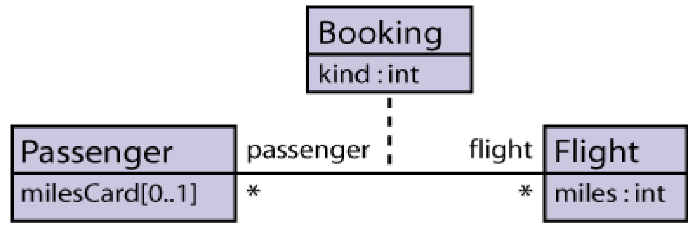
- In general, OCL expressions are evaluated over a system state.

e.g., represented
by an object diagram



Connection to UML

- Import of classifiers and enumerations as types
- Properties accessible in OCL
  - Attributes
    - p . milesCard (with p : Passenger)
  - Association ends
    - p . flight, p . booking, p . booking [flight]
  - { query } operations
  - Access to stereotypes via v`.stereotype`



- **Representation of multiplicities**

a[1] : T	a : T
a[0..1] : T	a : `Set(`T`)` or T
a[m..n] : T	a : `Set(`T`)`
a[*] : T { unordered }	a : `Set(`T`)`
a[*] : T { ordered }	a : `OrderedSet(`T`)`
a[*] : T { bag }	a : `Bag(`T`)`

Invariants

context classifier

boolean expression

```
context Passenger
inv: ma.statusMiles > 10000 implies
     status = Status::Albatros
```

Notational variants

explicit self (refers to instance of discourse)

```
context Passenger
inv statusLimit: self.ma.statusMiles > 10000 implies
                 self.status = Status::Albatros
```

optional name

```
context p : Passenger
inv statusLimit: p.ma.statusMiles > 10000 implies
                 p.status = Status::Albatros
```

replacement for self

Semantics of invariants

- Restriction of valid states of classifier instances
  - when observed from outside
- Invariants (as all constraints) are inherited via generalizations
  - but how they are combined is not predefined
- One possibility: Combination of several invariants by **conjunction**

context C
inv: I_1

context C
inv: I_2

$\rightsquigarrow$

context C
inv: I_1 **and** I_2

Pre-/post-conditions

- In UML models, pre- and post-conditions are defined separately
  - not necessarily as pairs
  - «precondition» and «postcondition» as constraint stereotypes

```
context Passenger::consumeMiles(b : Booking) : Boolean
pre: ma->notEmpty() and
     ma.flightMiles >= b.flight.miles

context Passenger::consumeMiles(b : Booking) : Boolean
post: ma.flightMiles = ma.flightMiles@pre-b.flight.miles and
      result = true
```

- Some constructs only available in post-conditions
  - values at pre-condition time — *p*`@pre`
  - result of operation call — `result`
  - whether an object has been newly created — *o*`.oclIsNew()`
  - messages sent — *o*`^`*op*`()`, *o*`^^`*op*`()`

Semantics of pre-/post-conditions

- Standard interpretation
  - A pre-/post-condition pair (P, Q) defines a relation R on system states such that $(\sigma, \sigma') \in R$, if $\sigma \models P$ and $(\sigma, \sigma') \models Q$.
    - system state σ on operation invocation
    - system state σ' on operation termination (Q may refer to σ by `@pre`).
  - Thus (P, Q) equivalent to (`true`, P`@pre` **and** Q).

- **Meyer's contract view**
  - A pre-/post-condition pair (P, Q) induces benefits and obligations.
  - benefits and obligations differ for implementer and user

	obligation	**benefit**
user	satisfy P	Q established
implementer	if P satisfied, establish Q	P established