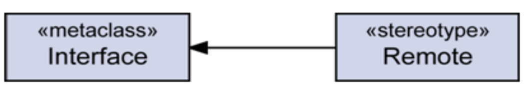
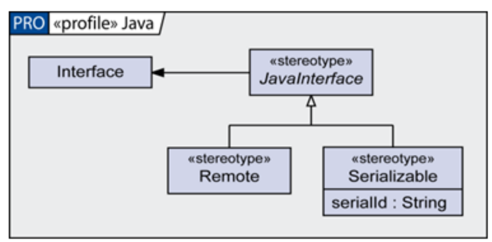
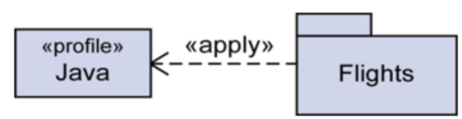
Unified Modeling Language 2

Profiles

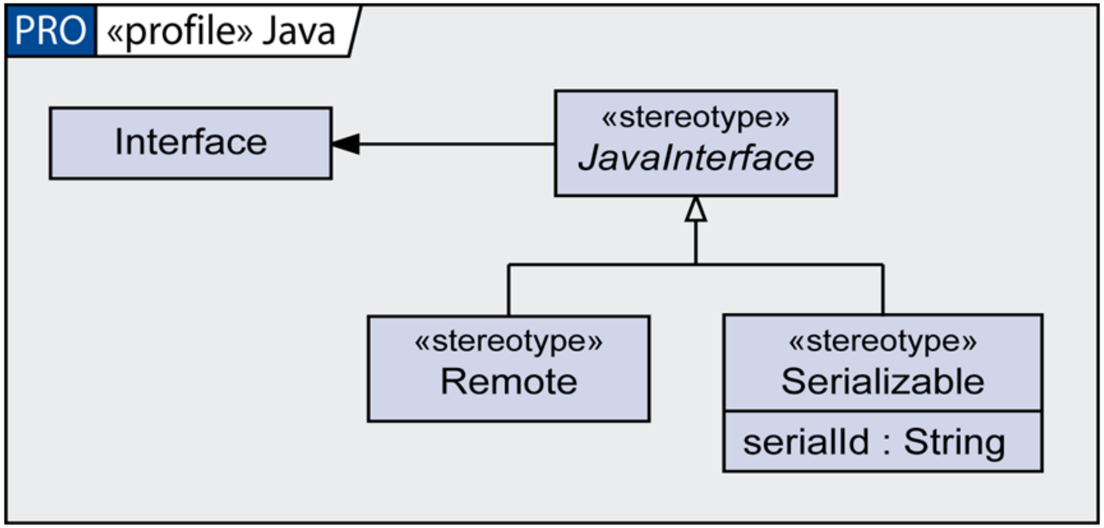






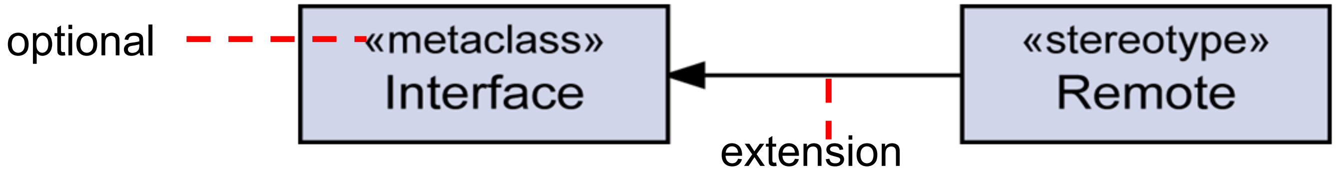
Usage scenarios

- **Metamodel customization** for
  - adapting terminology to a specific platform or domain
  - adding (visual) notation
  - adding and specializing semantics
  - adding constraints
  - transformation information
- **Profiling**
  - packaging domain-specific extensions
  - “domain-specific language” engineering

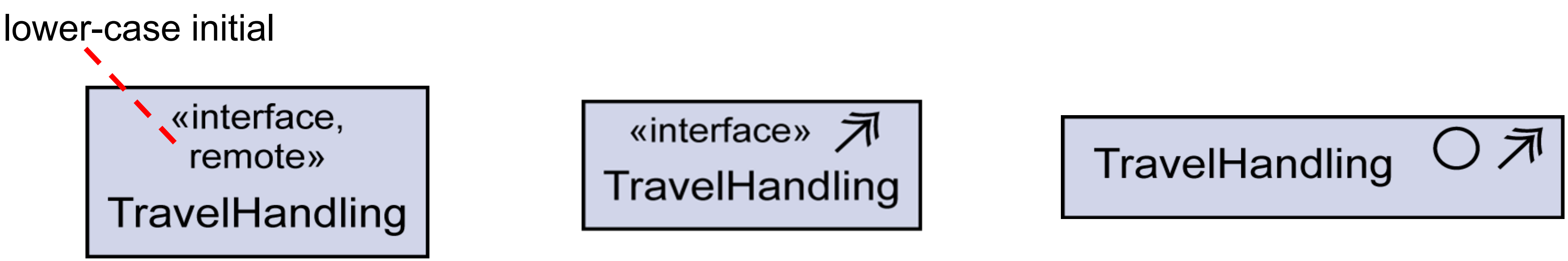


Stereotypes (1)

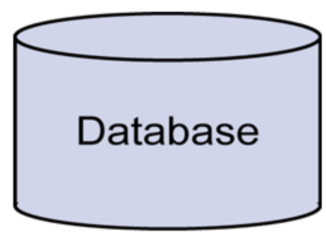
- Stereotypes define how an existing (UML) metaclass may be extended.



- Stereotypes may be applied **textually** or **graphically**.

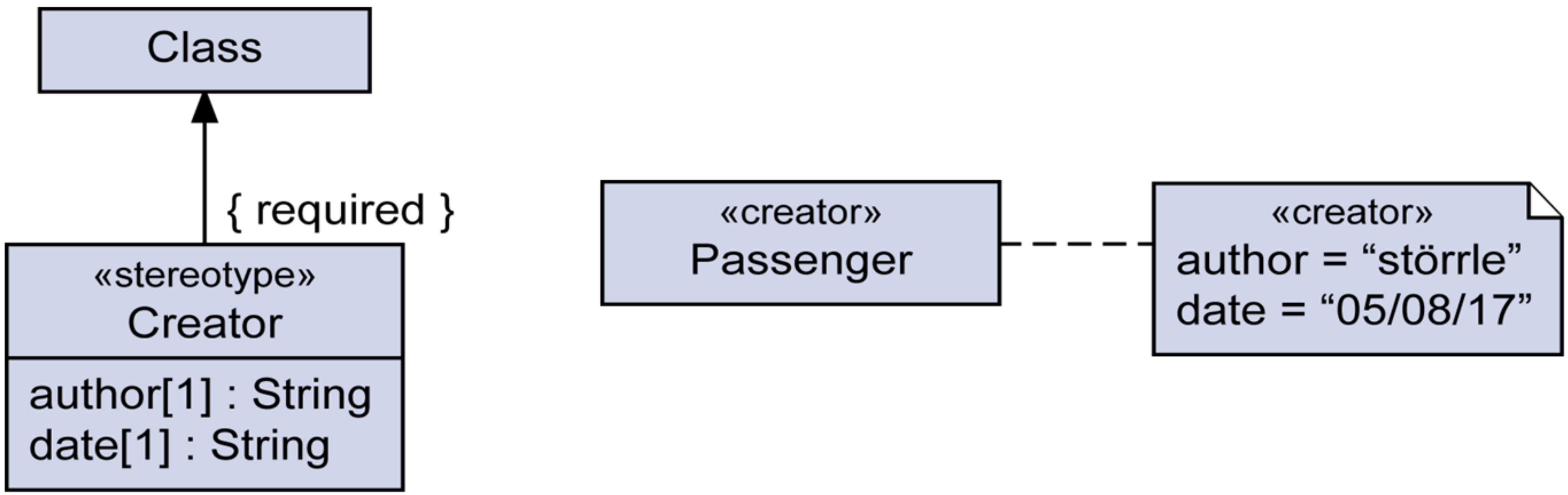


- Visual stereotypes may replace original notation.
  - But the element name should appear below the icon…



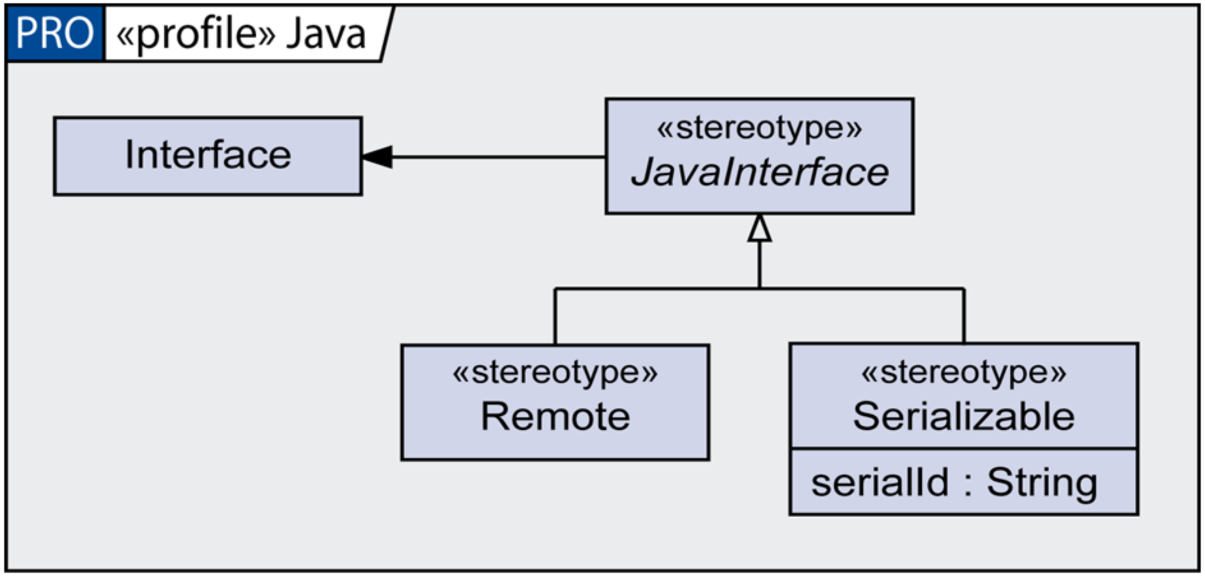
Stereotypes (2)

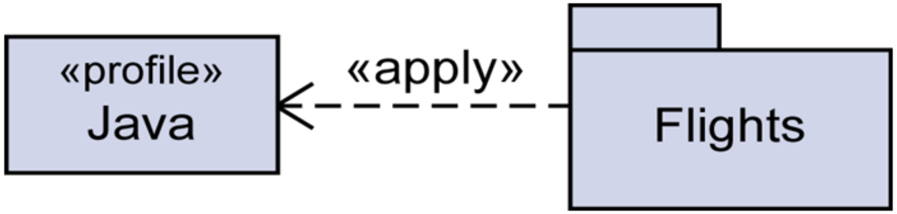
- Stereotypes may define **meta-properties**.
  - commonly known as “tagged values”
- Stereotypes can be defined to be **required**.
  - Every instance of the extended metaclass has to be extended.
  - If a required extension is clear from the context it need not be visualized.



Profiling

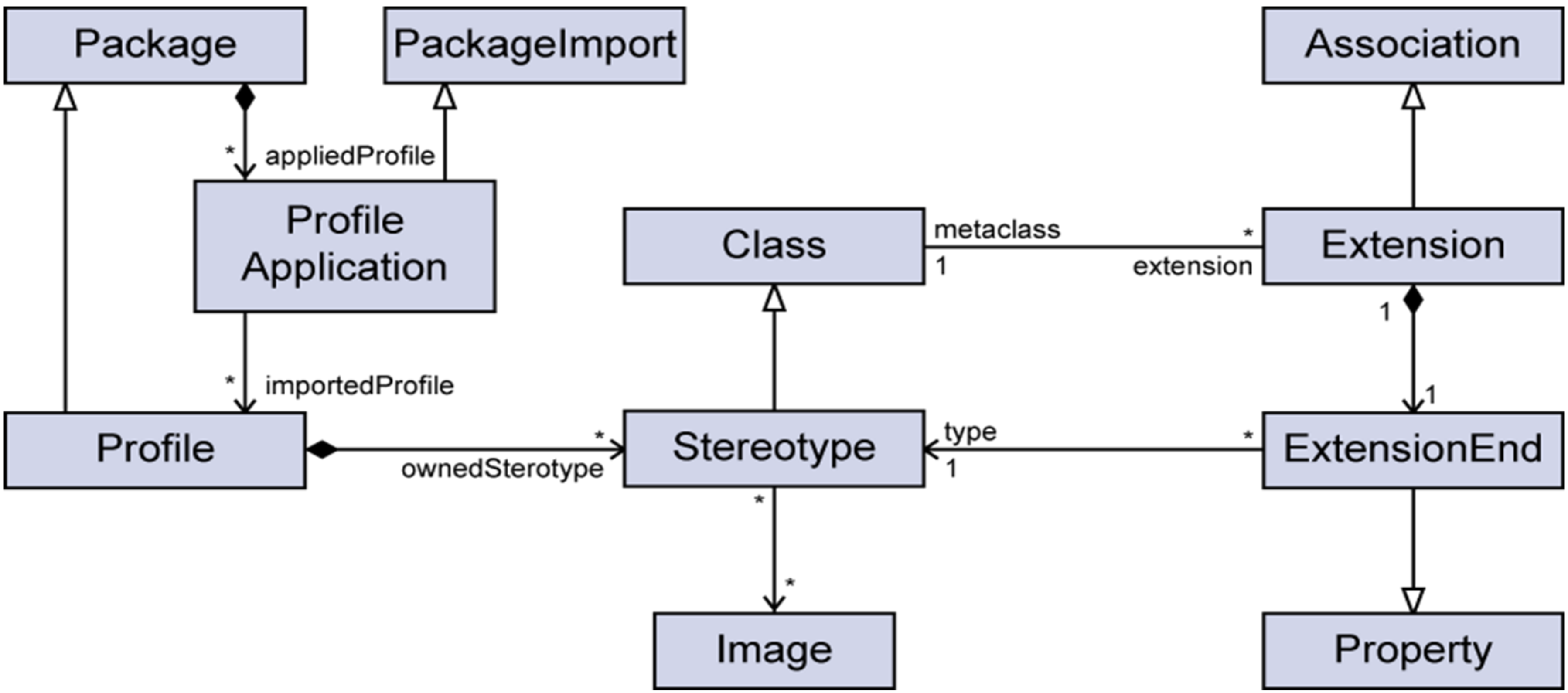
- Profiles **package** extensions.





Metamodel

- Based on **infrastructure library** constructs
  - Class, Association, Property, Package, PackageImport



Metamodeling with Profiles

- Profile extension mechanism imposes **restrictions** on how the UML metamodel can be modified.
  - UML metamodel considered as “read only”.
  - No intermediate metaclasses

- Stereotypes metaclasses below UML metaclasses.

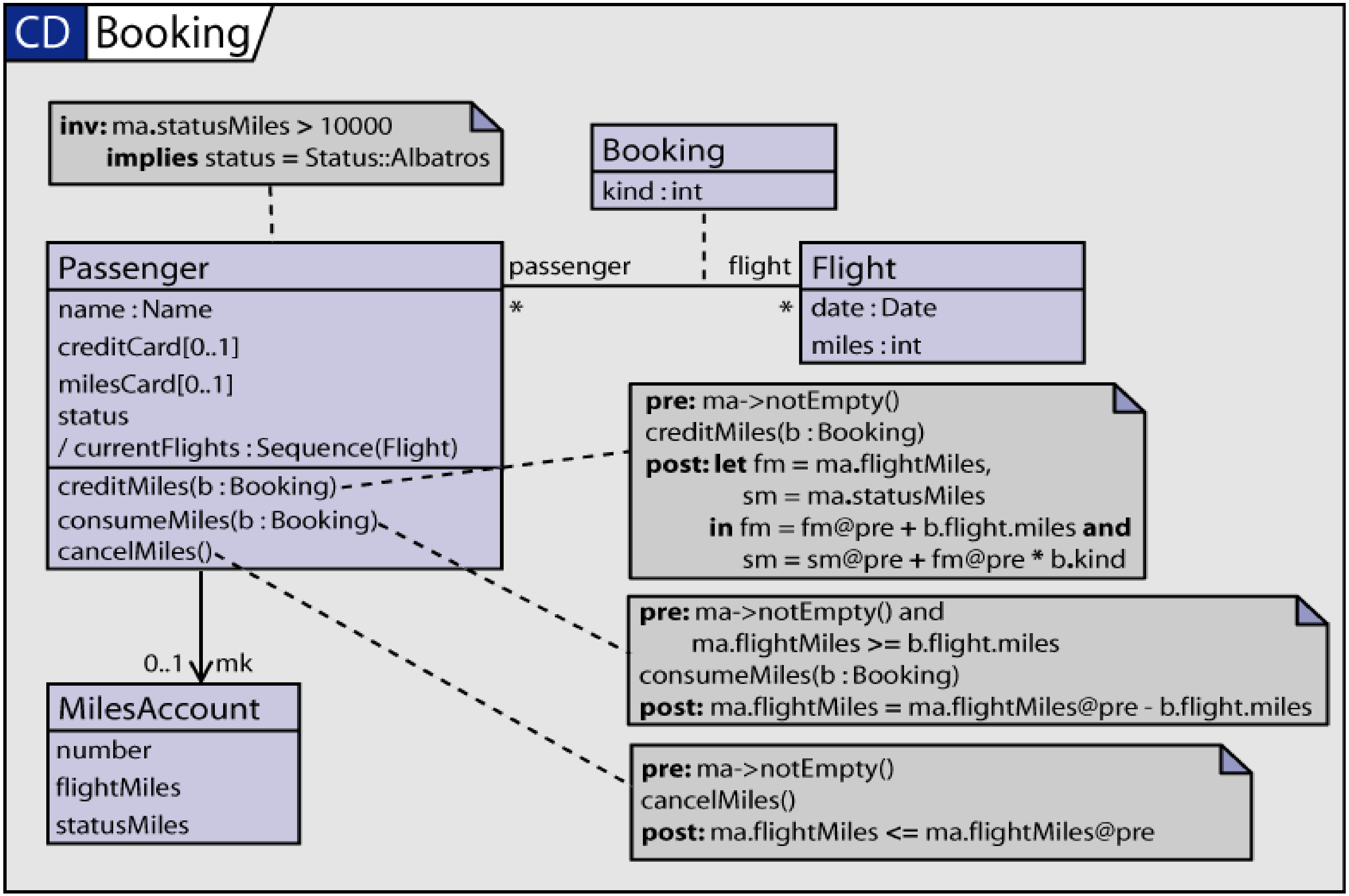
Wrap up

- Metamodel extensions
  - with stereotypes and meta-properties
  - for restricting metamodel semantics
  - for extending notation

- Packaging of extensions into profiles
  - for declaring applicable extensions
  - “domain-specific language” engineering

Object Constraint Language 2

A first glimpse



History and predecessors

- **Predecessors**
  - Model-based specification languages, like
    - Z, VDM, and their object-oriented variants; B
  - Algebraic specification languages, like
    - OBJ3, Maude, Larch

- Similar approaches in programming languages
  - ESC, JML

- **History**
  - developed by IBM as an easy-to-use formal annotation language
  - used in UML metamodel specification since UML 1.1
  - current version: OCL 2.3.1
    - specification: formal/2012-01-01

Usage scenarios

- Constraints on implementations of a model
  - invariants on classes
  - pre-/post-conditions for operations
    - cf. protocol state machines
  - body of operations
  - restrictions on associations, template parameters, …
- Formalization of side conditions
  - derived attributes
- Guards
  - in state machines, activity diagrams
- Queries
  - query operations
- **Model-driven architecture (MDA)/query-view-transformation (QVT)**

Language characteristics

- Integration with UML
  - access to classifiers, attributes, states, …
  - navigation through attributes, associations, …
  - limited reflective capabilities
  - model extensions by derived attributes
- **Side-effect free**
  - *not* an action language
  - only possibly describing effects
- **Statically typed**
  - inherits and extends type hierarchy from UML model
- Abstract and concrete syntax
  - precise definition new in OCL 2

Simple types

- Predefined primitive types
  - `Boolean` `true`, `false`
  - `Integer` `-17, 0, 3`
  - `Real` `-17.89, 0.0, 3.14`
  - `String` `“Hello”`
- Types induced by UML model
  - Classifier types, like
    - Passenger no denotation of objects, only in `context`
  - Enumeration types, like
    - Status `Status::Albatros, #Albatros`
  - Model element types
    - `OclModelElement`, `OclType`, `OclState`
- Additional types
  - `OclInvalid` invalid (`OclUndefined`)
  - `OclVoid` null
  - `OclAny` top type of primitives and classifiers

Parameterized types

- Collection types
  - `Set(`T`)` sets
  - `OrderedSet(`T`)` like `Sequence` without duplicates
  - `Bag(`T`)` multi-sets
  - `Sequence(`T`)` lists
  - `Collection(`T`)` abstract
- Tuple types (records)
  - `Tuple(`a_1 `:` T_1`, …,` a_n `:` T_n`)`
- Message type
  - `OclMessage` for operations and signals

Examples

- `Set{Set{ 1 }, Set{ 2, 3 }} : Set(Set(Integer))`
- `Bag{1, 2.0, 2, 3.0, 3.0, 3} : Bag(Real)`
- `Tuple{x = 5, y = false} : Tuple(x : Integer, y : Boolean)`

Type hierarchy

- Type conformance (reflexive, transitive relation ≤)
  - `OclVoid` $\leq T$ for all types T but `OclInvalid`
  - `OclInvalid` $\leq T$ for all types T
  - `Integer` ≤ `Real`
  - $T \leq T' \Rightarrow C(T) \leq C(T')$ for collection type C
  - $C(T) \leq$ `Collection(`T`)` for collection type C
  - generalization hierarchy from UML model
  - $B \leq$ `OclAny` for all primitives and classifiers B

Counterexample

  - ¬(`Set(OclAny)` ≤ `OclAny`)

- Casting
  - v`.oclAsType(`T`)` if $v : T'$ and ($T \leq T'$ or $T' \leq T$)
  - upcast necessary for accessing overridden properties

Expressions

- Local variable bindings

  ```
  let x = 1 in x+2
  ```

- Iteration

  `c->iterate(i : T; a : T’ = e’ | e)`

  source collection

  iteration variable
  (bound to current value in *c*)

  accumulator with initial value *e’*
  (gathers result, returned after iteration)

  iteration expression
  (using variables *i* and *a*)

Example:

```
Set{1, 2}->iterate(i : Integer; a : Integer = 0 | a+i) = 3
```

- Many operations on collections are **reduced** to `iterate`

Expressions: Standard library (1)

- Operations on primitive types (written: v`.`op`(`…`)`)
  - operations without ( ) are mixfix

`OclAny`	=, <>, `oclIsTypeOf(`T`)`, `oclIsKindOf(`T`)`, …
`Boolean`	**`and`**, **`or`**, **`xor`**, **`implies`**, **`not`**
`Integer`	`+`, `-`, `*`, `/`, `div(`i`)`, `mod(`i`)`, …
`Real`	`+`, `-`, `*`, `/`, `floor()`, `round()`, …
`String`	`size()`, `concat(`s`)`, `substring(`l`,` u`)`, …

- Operations on collection types (written: v`->`op`(`…`)`)

`Collection`	`size()`, `includes(`v`)`, `isEmpty()`, …
`Set`	`union(`s`)`, `including(`v`)`, `flatten()`, `asBag()`, …
`OrderedSet`	`append(`s`)`, `first()`, `at(`i`)`, …
`Bag`	`union(`b`)`, `including(`v`)`, `flatten()`, `asSet()`, …
`Sequence`	`append(`s`)`, `first()`, `at(`i`)`, `asOrderedSet()`, …