

UNIFIED MODELING LANGUAGE

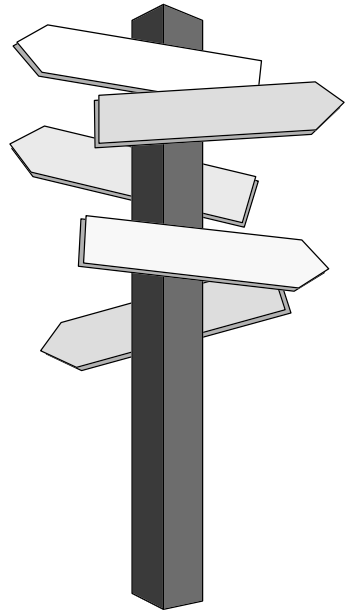
Aitecon

Static Structure Modeling

Radovan Červenka, October 1998 (version 0.04)

Context

- ✓ Introduction
- ✓ Generic Mechanisms
- ✓ Use Case Modeling
- **Static Structure Modeling**
- Dynamic Behavior Modeling
- Interaction Modeling
- Physical Structure Modeling
- General Extension Mechanisms



Static Structure Model

→ static structure of the system's architecture

Represented by:

- Class Diagrams
- Object Diagrams
- Element Descriptions

Supported by:

- Statechart diagrams
- Interaction diagrams

Used (mainly) in:

- Requirements Capture ⇒ domain model
- Analysis ⇒ analytic model
- Design and Implementation ⇒ design model

Static Structure Diagrams

Class Diagram

→ classes, their internal structure and their relationships

- organized into package

(Static) Object Diagram

→ static structure of instances (objects and links)

- a snapshot of the state of the system at a point in time
- compatible with a particular class diagram

* there are also “dynamic” object diagrams called *Collaboration Diagrams*

Perspectives

Conceptual

- conceptual/domain model
- no (little) regard for the SW implementation
- in *Requirements Capture*

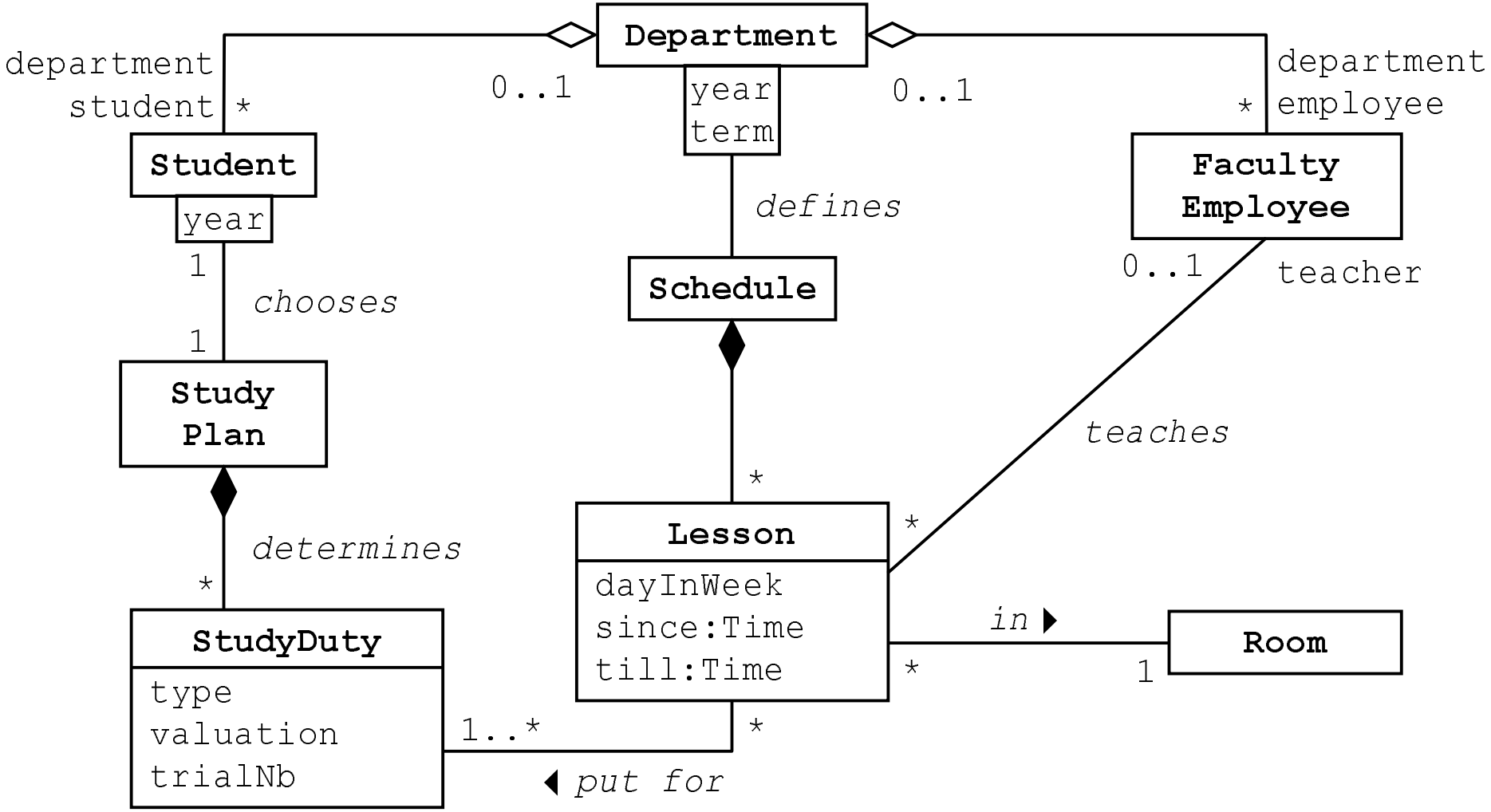
Specification

- logical application model
- looking at SW
- concerning on types rather than implementation
- in *Analysis*

Implementation

- implementation model
- in *Design*

Example of the Class Diagram



Class

→ an abstraction of set of objects that share a common structure (attributes, operations and links) and a common behavior/semantics

- attributes

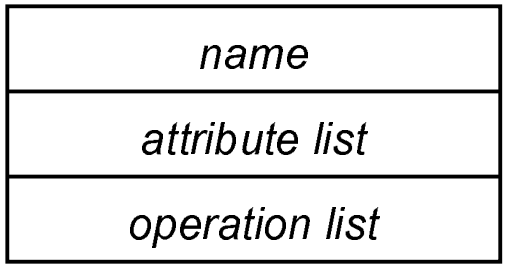
  visibility name [*multiplicity*] : *type* = *default*

- operations

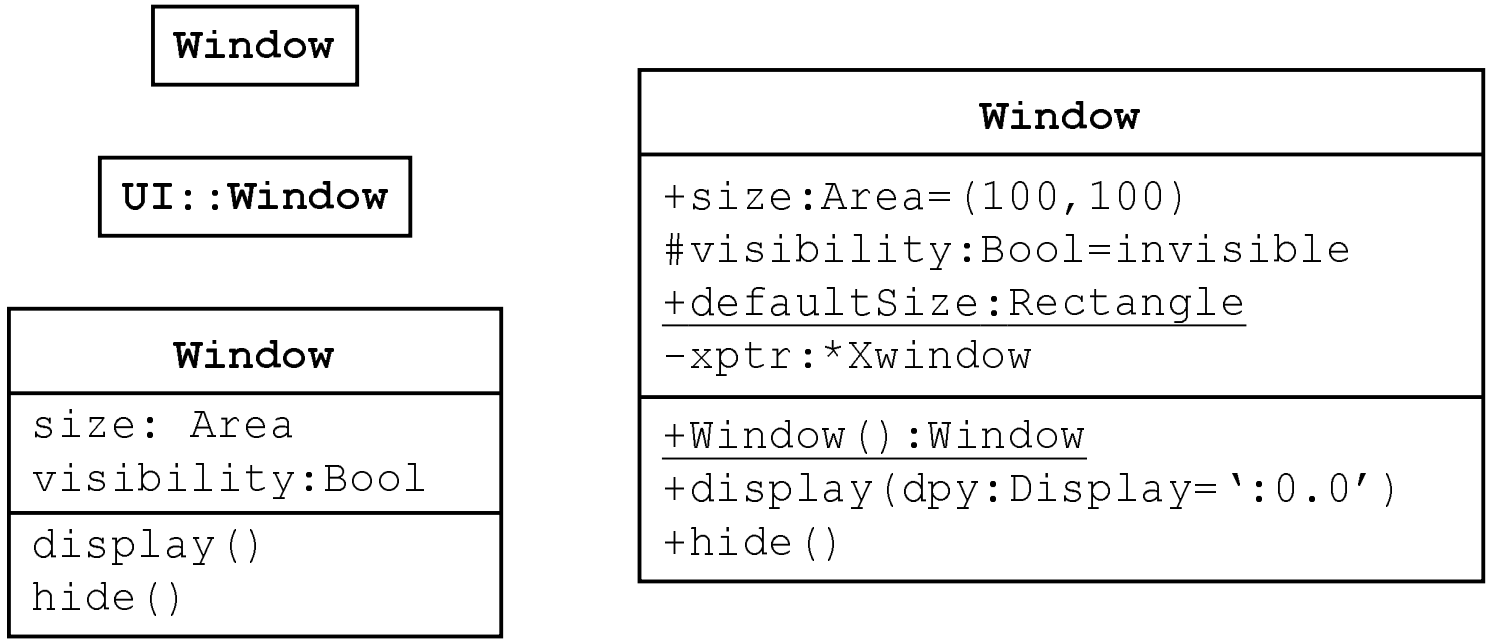
  visibility name (*parameters*) : *return_type*

  – format of parameters = attributes list

- visibility: '+' public, '#' protected, '–' private
- class attributes and operations are underlined



Examples of Classes



Type vs. Implementation Class

Type

→ the specification of allowable attributes that an object may comprise and the set of operations that may be performed upon the object

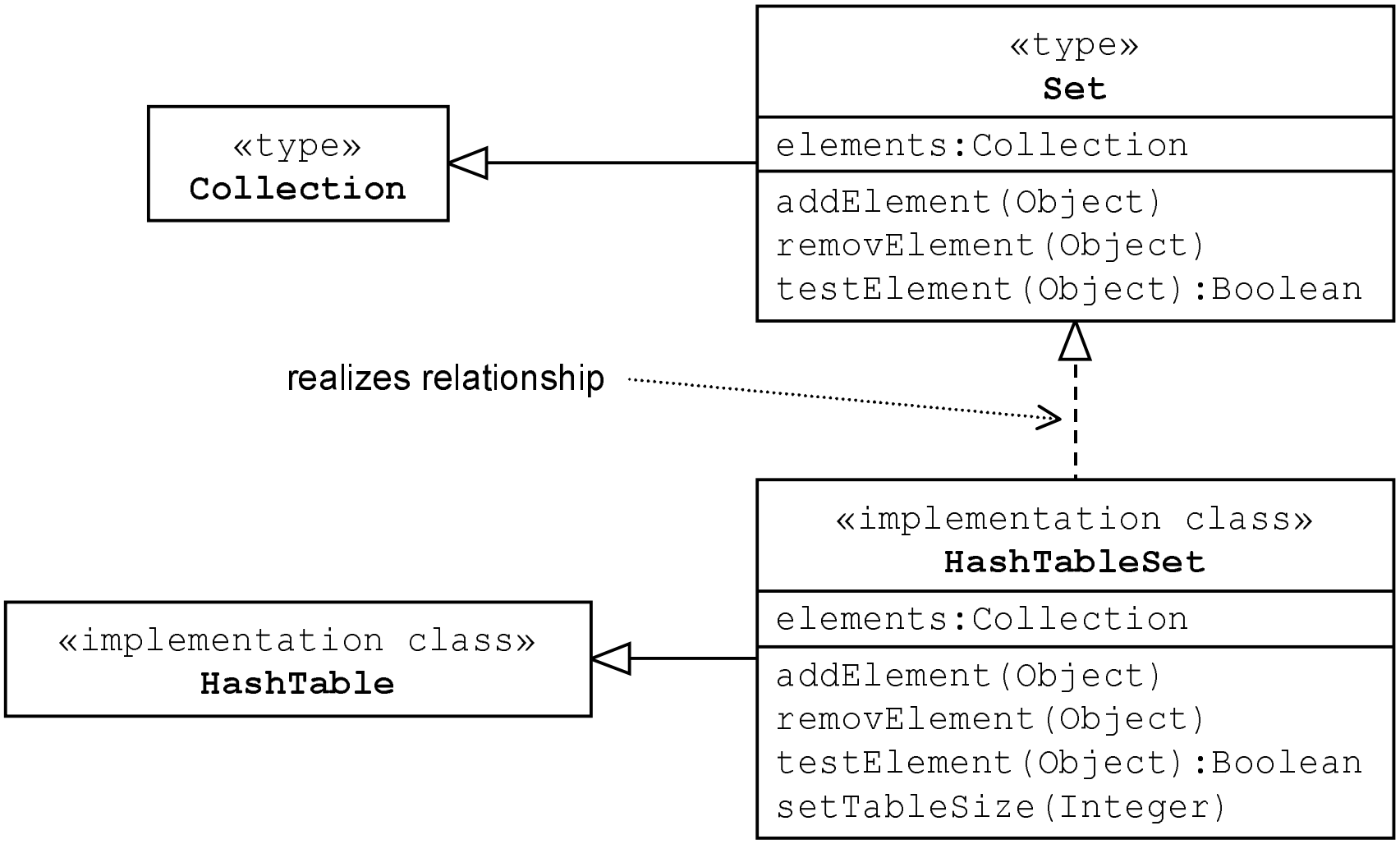
- conformance to a common protocol
- notation: as a class with stereotype «type»

Implementation Class

→ defines the physical data structure and methods of an object as implemented in traditional languages (C++, Smalltalk,etc.)

- notation: as a class with stereotype
  «implementation class»
- relation between an implementation class and a type is modelled as the *realizes relationship*

Example of Type and Implementation Class

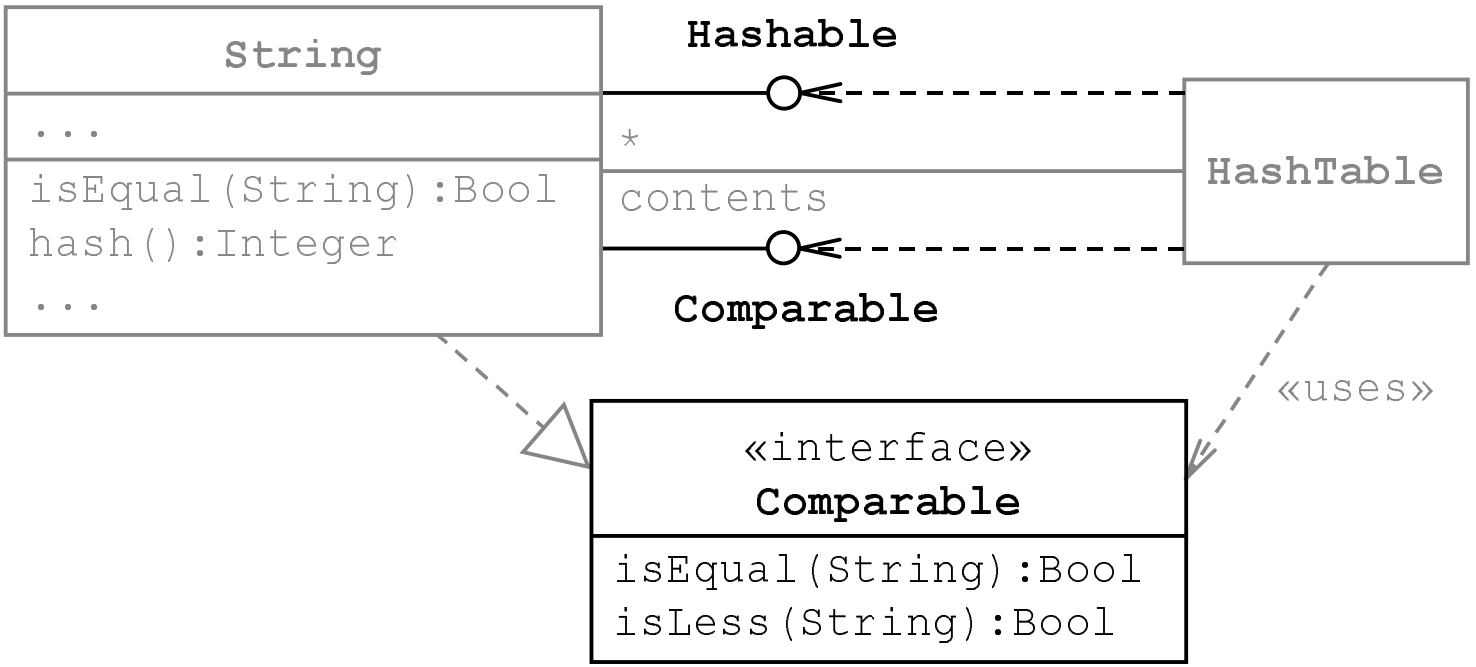


Interface

→type describing externally visible operations of a class, component, package etc. without specification of the internal structure

- no attributes, states or associations - only operations

  ⇒no implementation



Parameterized Class and Bound Element

Parameterized Class (Template)

→the descriptor for a class with unbound formal parameters

- defines a family of classes
- a class is specified by “binding” the parameters to actual values
- parameter can represent used type, attribute or operation
- format of parameter list: *name* : *type*, *name* : *type*, ...

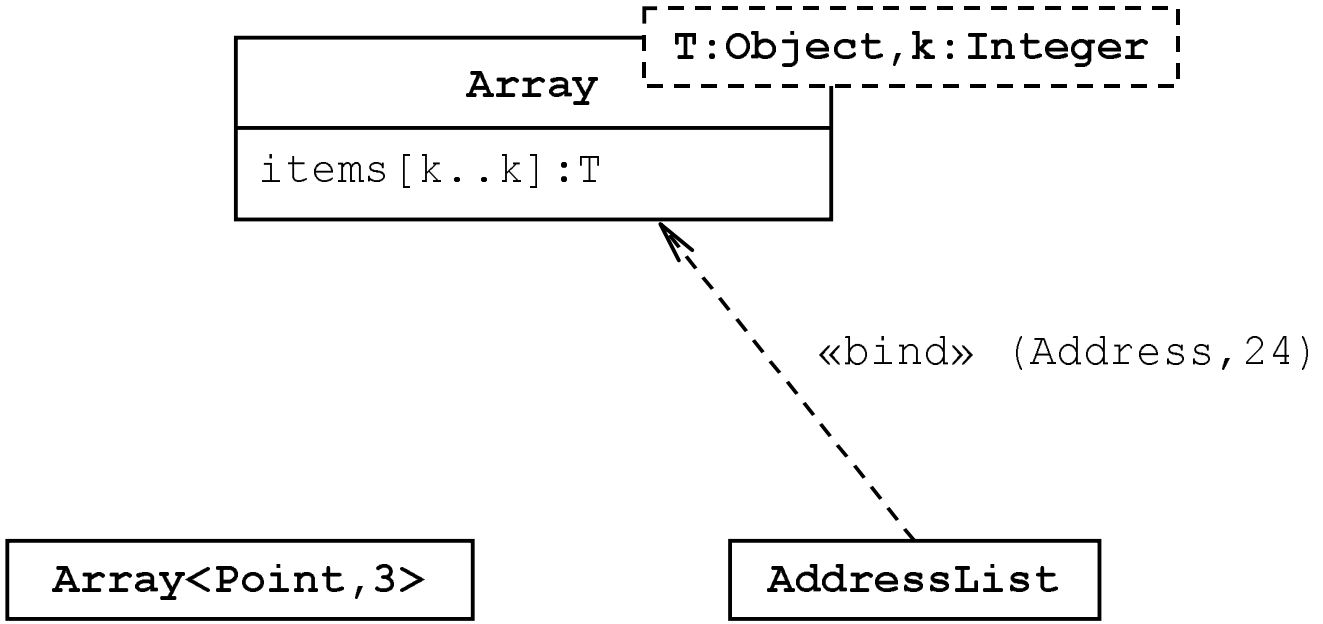
Bound Element

→a class specified by binding of template’s formal parameters to actual values

- it is fully specified by the template

  ⇒template’s contents cannot be extended

Example of a Parameterized Class

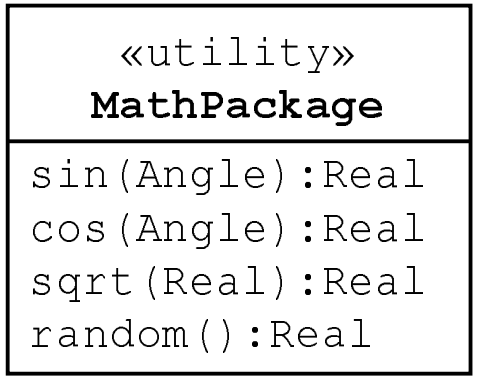


Utility

Utility

→ a grouping of global variables and procedures

- notation: as a class with stereotype «utility»



Association and Association Class

Association

→ relation among classes describing a set of links

- they are *binary* and *n-ary*
- ∗ name is optional; it can include the direction in which to read an association

Association Class

→ an association with class-like properties (attributes, operations, relations, behaviour)

- an association and its connected association class represent the same model element

  ⇒ they must have the same name

Association End [1]

→ an end of an association where it connects to a class

- part of the association, not the class

Multiplicity

→ the range of allowable cardinalities that a set may assume

- format: a list of values and intervals expressed as

  lower limit `..` *upper limit*

- possible values : *number* or * (many)

```
1
*
0..*
1..*
2,4,6..10,20..*
```

Ordering: `{ordered}` or `{unordered}`

Association End 2

Navigability

→ indicates that navigation is supported toward attached class

Rolename

→ the role played by the class attached to end of the path near the rolename

Changeability: `{frozen}` or `{addOnly}`

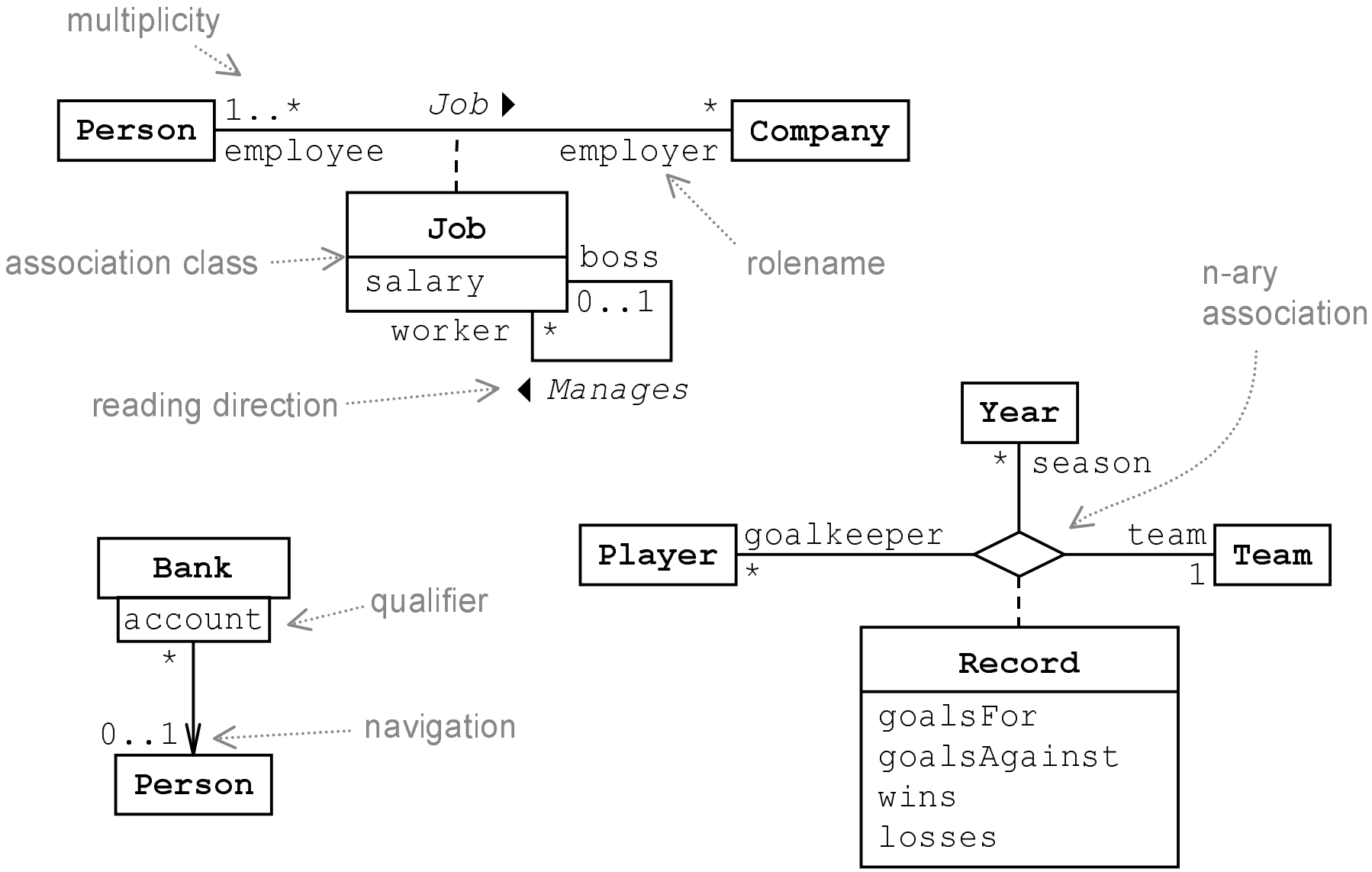
Visibility:

`{public}` (+)
`{protected}` (#)
`{private}` (-)

Qualifier

→ an attribute or a list of attributes whose values serve to partition the set of links

Examples of Associations



Aggregation and Composition

Aggregation

→ a special association representing the relationship between the *whole* and its *parts*

* aggregation can also be treated as an “ownership by a reference”

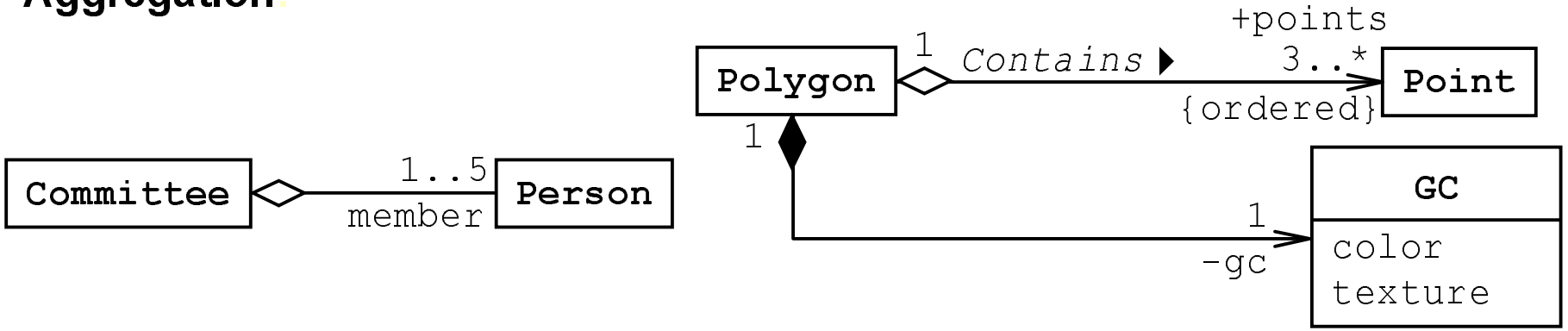
Composition

→ a special kind of aggregation with “strong ownership” and coincided lifetime of part with the whole (all parts can be created after the creation of the whole and they are explicitly destroyed before the destruction of the whole)

* a composition can also be treated as an “ownership by a value”

Examples of Aggregations and Compositions

Aggregation:

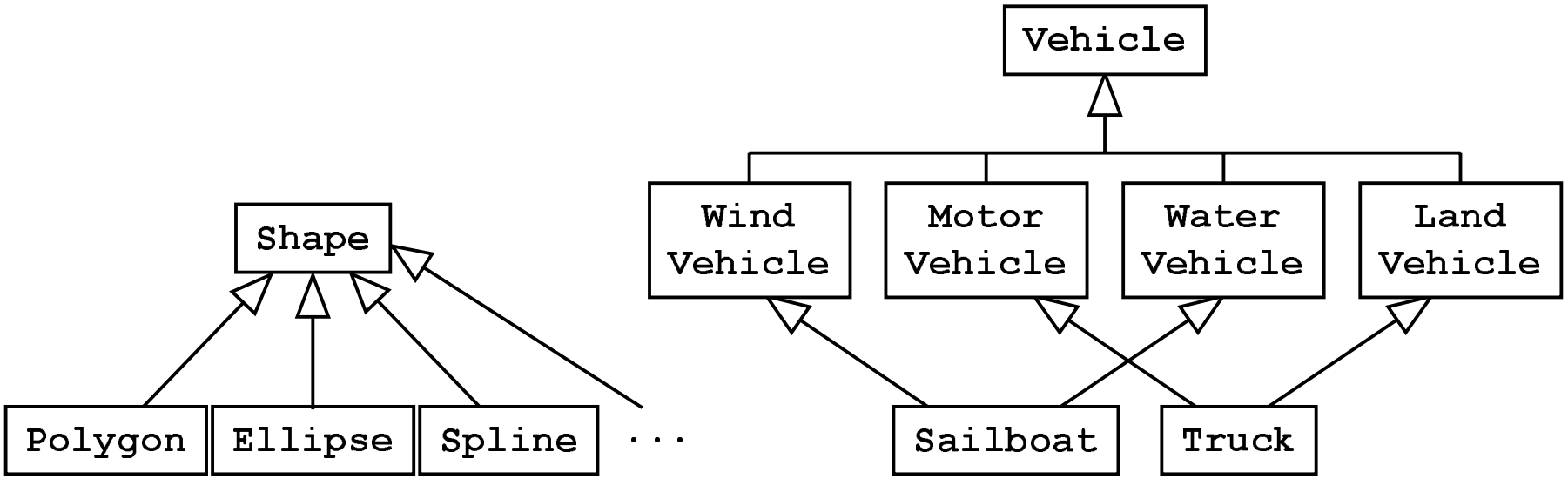


Composition:

Generalization

→ the taxonomic relationship between a more general and more specific elements

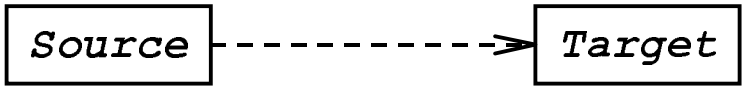
- *sub-elements* inherit structure and/or behaviour from *super-elements*
- may be applied to classes, packages or use cases



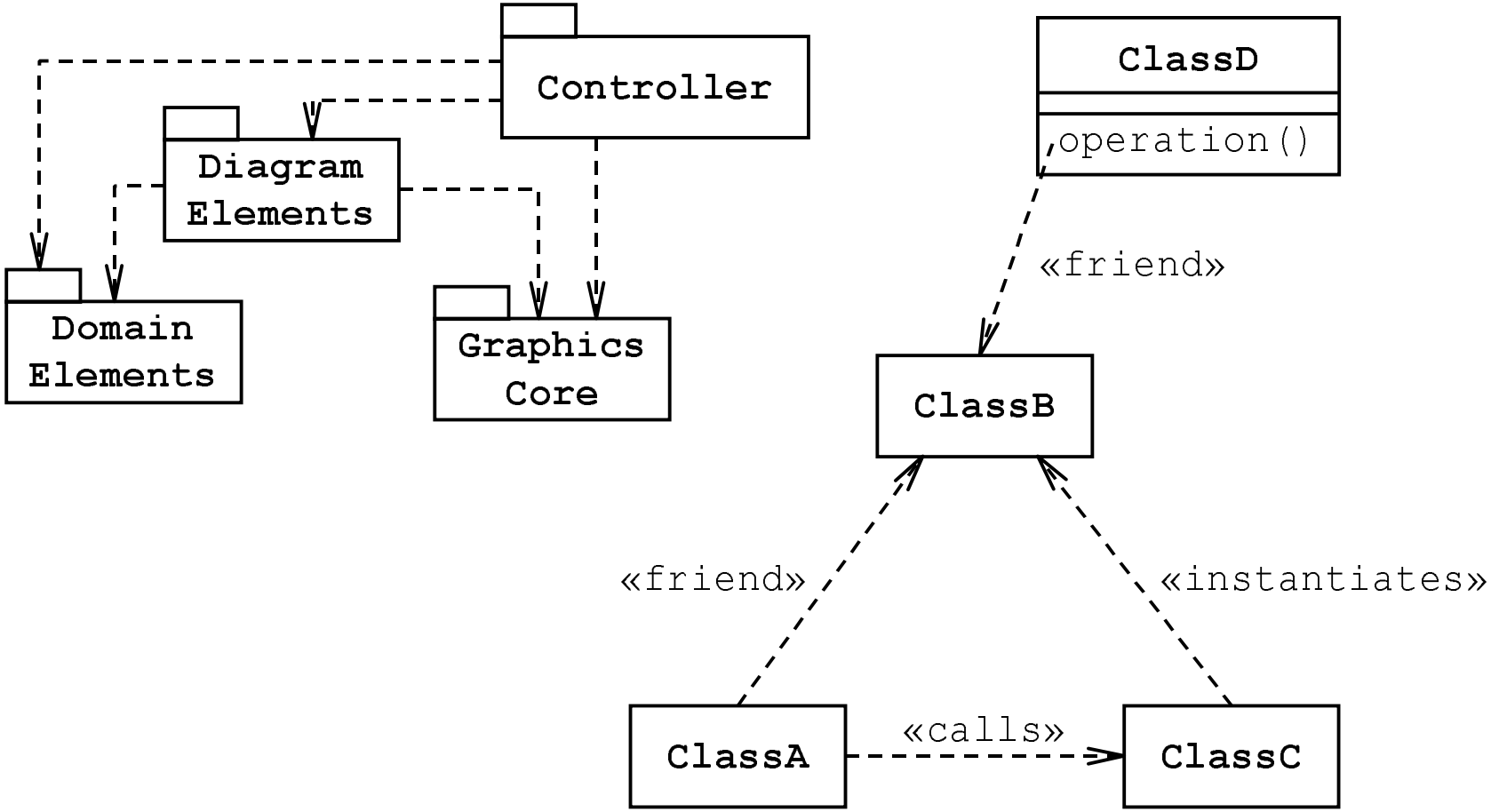
Dependency

→ a semantic relationship between two model elements; it indicates: “a change of the *target element* can cause a change of the *source element*”

- relates to model elements themselves; not to their instances



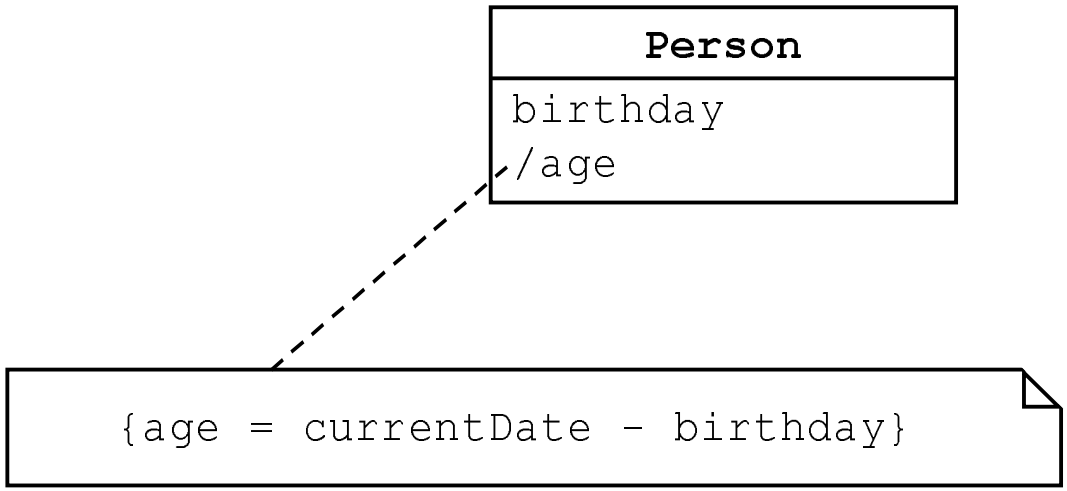
Examples of Dependencies



Derived Element

→ an element that can be computed from another one, but is shown for clarity or for design purposes

- notation: it is shown by placing a slash (/) in front of the element name



Navigation Expressions (OCL)

→ they allow to express navigation in the class models

item.*selector*

- the *selector* is the name of an attribute in the *item* or the role name of the target end of a link attached to the *item*. The result is the value of an attribute or related object(s).

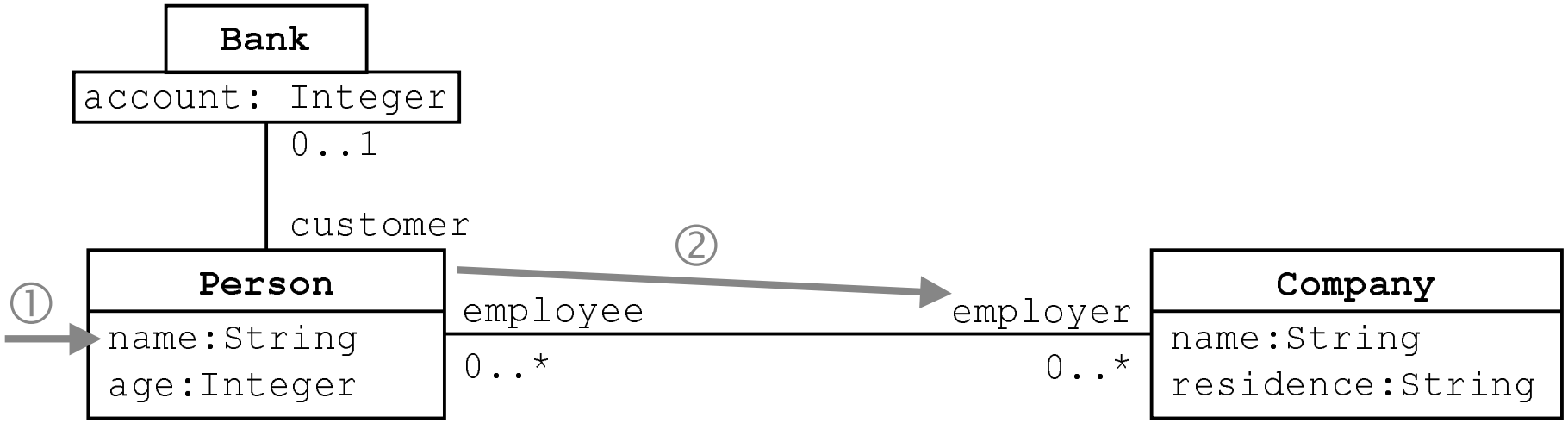
item.*selector* [*qualifier-value*]

- a *selector* designates a qualified association that qualifies the *item*. The *qualifier-value* is a value for the qualifier attribute. The result is related object selected by the qualifier.

set->`select(` *boolean-expression* `)`

- the *boolean-expression* is written in terms of objects within the *set*. The result is the subset of objects in the *set* for which the *boolean-expression* is true.

Example of Navigation [1]



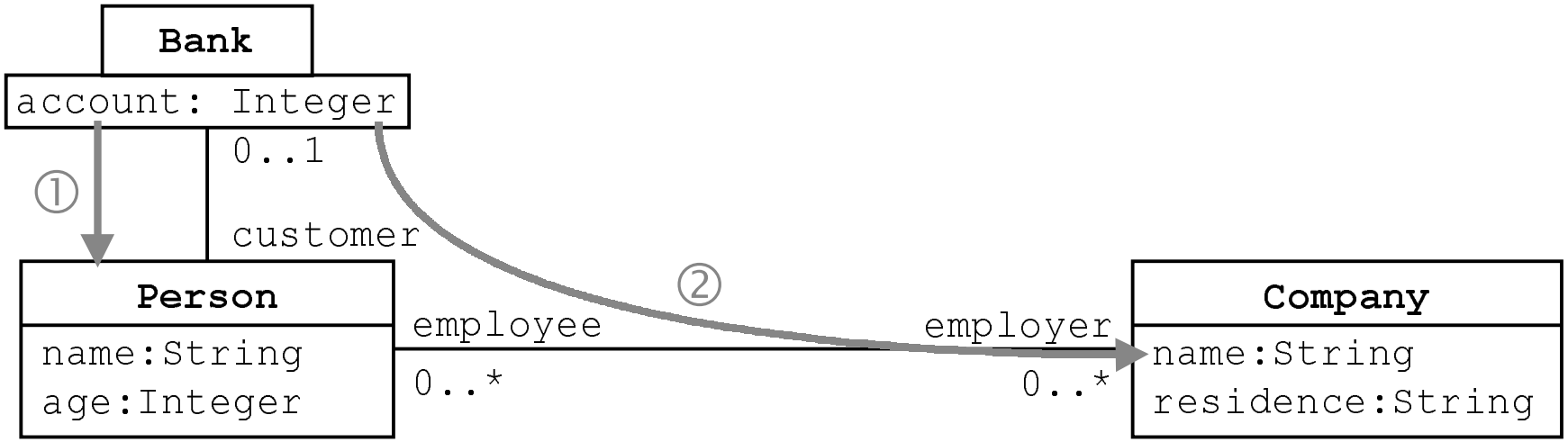
① Name of a person :

Person.name

② Names of person's employers :

Person.employer.name

Example of Navigation 2



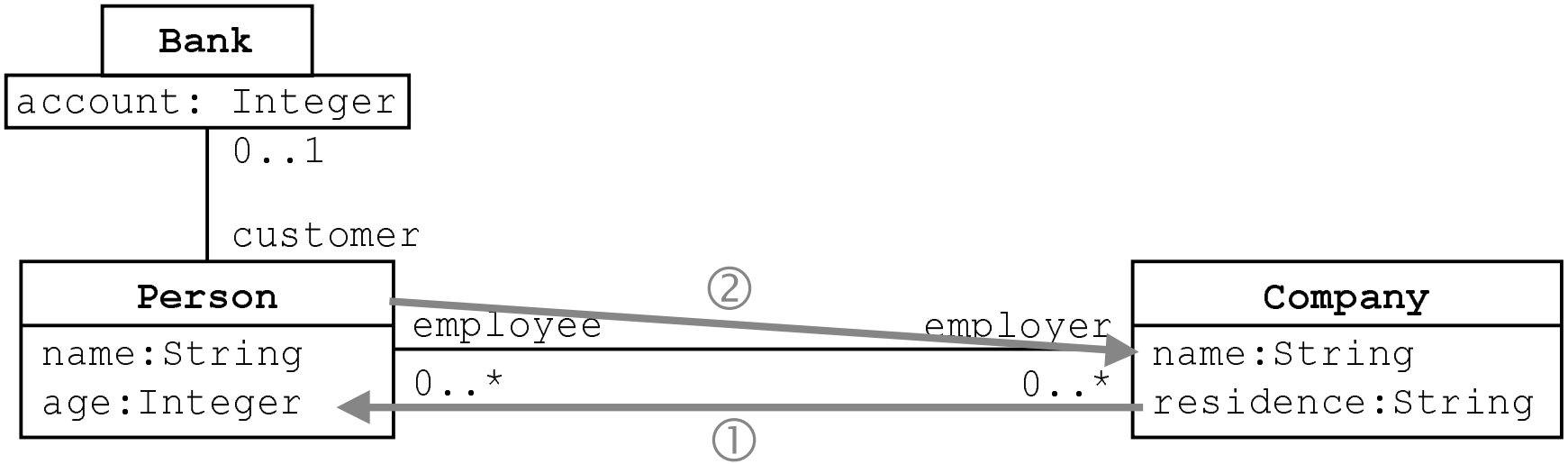
① A customer of the bank with the account num. 8526:

`Bank.customer[8526]`

② Employers of an owner of the account 6251:

`Bank.customer[6251].employer.name`

Example of Navigation 3



① Employees older than 50:

```
Company.employee -> select(p|p.age>50)
```

② Names of employers from Bratislava:

```
Person.employer ->
select(c|c.residence='Bratislava').name
```

Object

Object

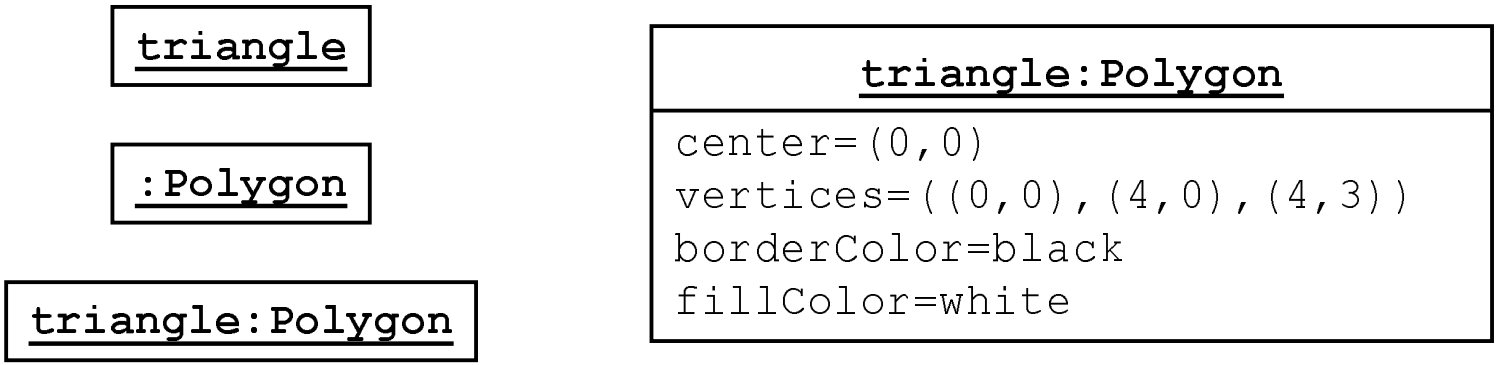
→ a particular instance of a class

- object name:

object_name : *class_name*

- attribute values list:

name : *type* = *value*



Link

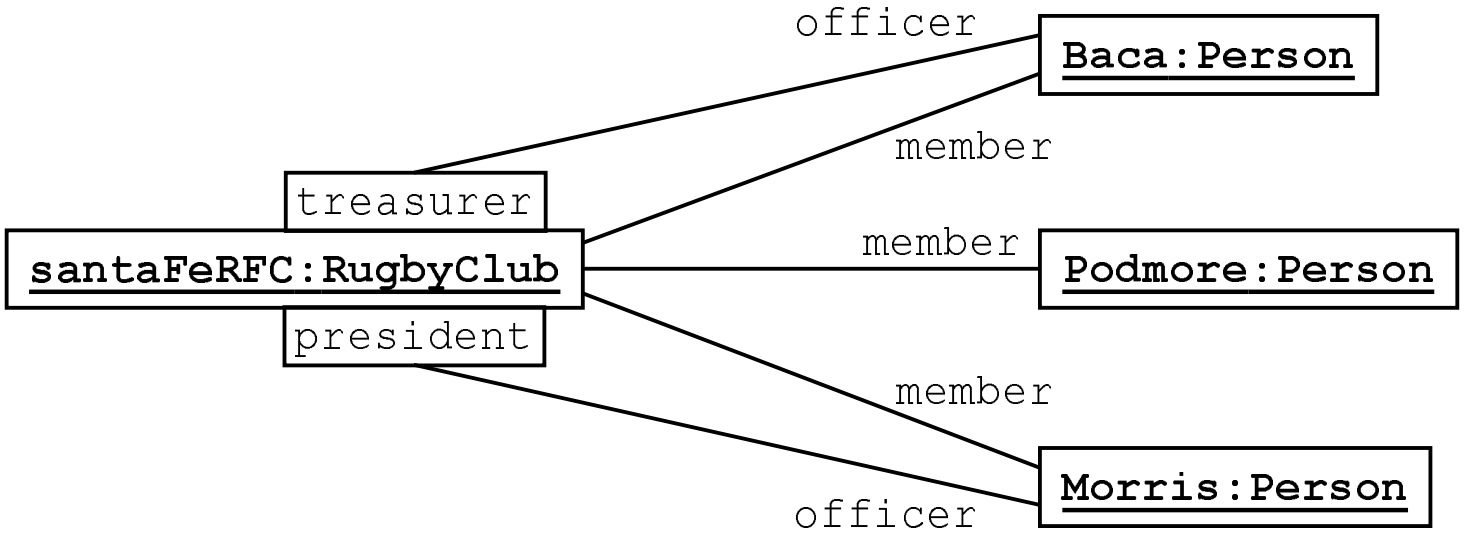
→ a tuple (mostly a pair) of object references

- an instance of an association
- multiplicity is NOT shown

  ⇐ link is an instance
- other association adornments may be shown
- stereotypes:

«association»	association (default, unnecessary to write)
«parameter»	procedure parameter
«local»	local variable of a procedure
«global»	global variable
«self»	self link (sending a message to itself)

Example of links



Process of Static Structure Modeling

- **Identify classes**
  - from Glossary
  - from Business Model
  - stored information items
  - from use case realizations
- **Specify the semantics of classes**
  - responsibility
  - attributes, operations and interfaces
- **Identify relationship among classes**
  - domain-based associations
  - object interactions
  - generalization and aggregation relationships
- **Structure the model into packages**

Summary

- Class
- Type
- Implementation Class
- Interface
- Parameterized Class
- Bound Element
- Utility
- Association
- Aggregation
- Composition
- Generalization
- Dependency
- Derived Element
- Navigation Expressions
- Object
- Link
- Process of Static Structure Modeling