WHITESTEIN Technologies

Unified Modeling Language

Classes

Radovan Cervenka

Class (Structural) Model

→ **Structure of the system expressed in terms of classes, interfaces, objects and their relationships.**

Consists of:

- Class diagrams.
- Object diagrams.
- Package diagrams.
- Element descriptions.

Supported by:

- State machines.
- Activities.
- Interactions.

Used (mainly) in:

- Requirements ⇒ domain/conceptual model.
- Analysis ⇒ analytical (logical) model.
- Design ⇒ design model.

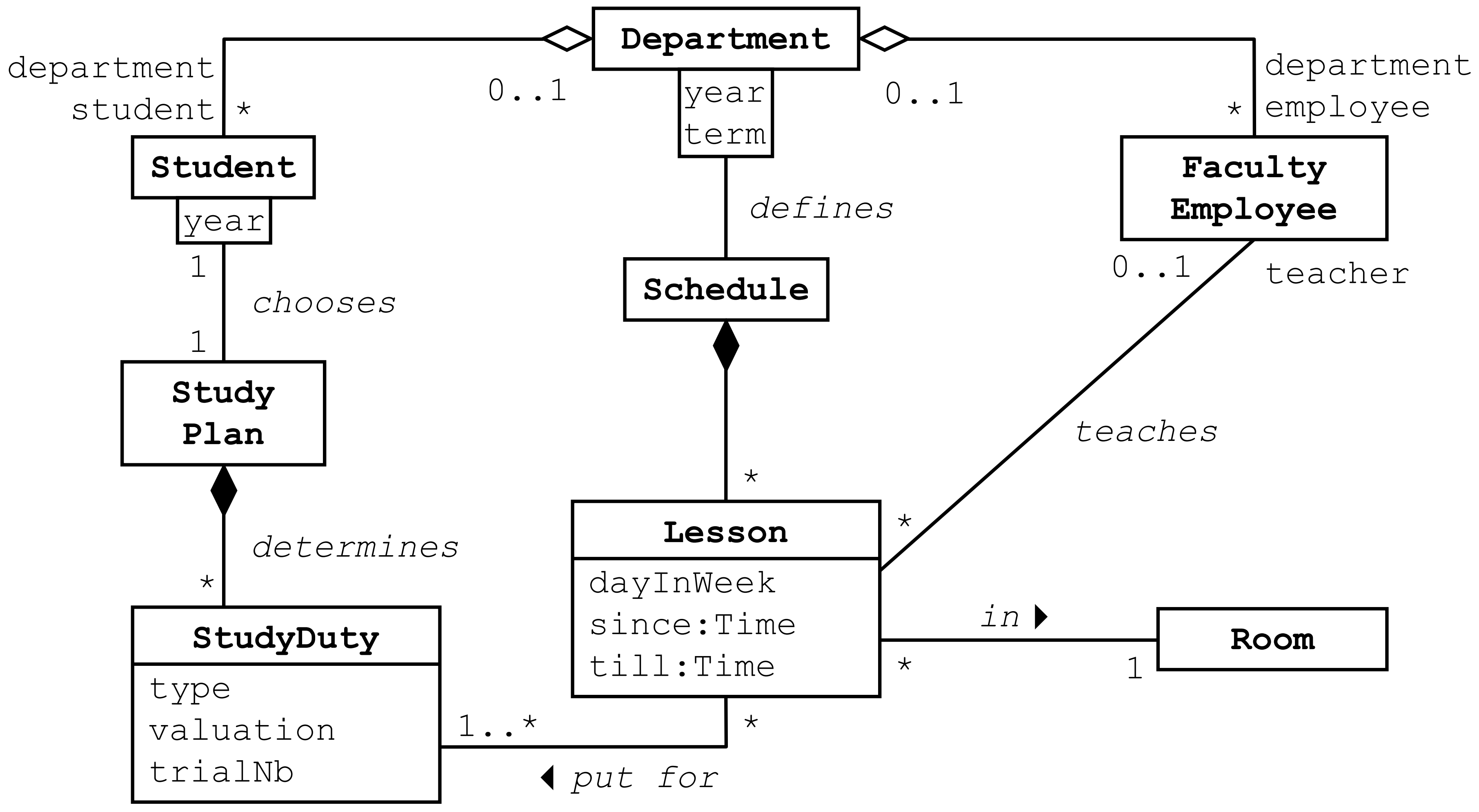
Diagrams

- ***Structure Diagram***
  - → An abstract diagram type showing the static structure of the objects in a system.
  - Has several specific diagrams: class diagram, object diagram, composite structure diagram, component diagram, deployment diagram, and package diagram.
- ***Class Diagram***
  - → Classes, interfaces and their relationships.
- ***Object Diagram***
  - → Static structure of instances (objects and links).
  - A snapshot of the state of the system at a point in time.
  - Possibly compatible with a particular class diagram.
- ***Package Diagram***
  - → Packages and their relationships.

Perspectives

- *Conceptual*
  - Conceptual/domain model.
  - No (little) regard for the SW implementation.
  - Used in Requirements.
- *Specification*
  - Logical application model.
  - Focused on software.
  - Concerning on types rather than implementation.
  - Used in Analysis.
- *Implementation*
  - Implementation model.
  - Used in Design.

Example of Class Diagram

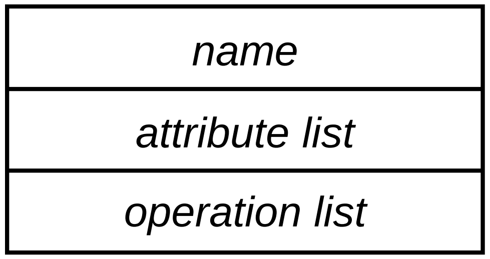


Class

- → An abstraction of set of objects that share a common structure (attributes, operations and links) and a common behavior/semantics.
- ■ A kind of classifier whose features are attributes and operations.
- ■ Format of attributes (owned properties):

  property ::= [*visibility*] ['/'] *name* [':' *type*] ['[' *multiplicity* ']'] ['='*default*] ['{' *prop-modifier* [',' *prop-modifier*]* '}']

  - Visibility: '+' public, '-' private, '#' protected, '~' package
  - Derived property, which can be computed from other properties, is marked by '/'.
  - Multiplicity:
    - positive number (0, 1, 2, ...)
    - interval: *lower-bound* '..' *upper-bound*
    - '*' for infinite upper bound
    - examples: 3, 1..4, 1..*, *



Class (cont.)

- Property modifier:
  - ‘readOnly’ means that the property is read only.
  - ‘union’ means that the property is a derived union of its subsets.
  - ‘subsets’ *property-name* means that the property is a proper subset of the property identified by *property-name*.
  - ‘redefines’ *property-name* means that the property redefines an inherited property identified by *property-name*.
  - ‘ordered’ means that the property is ordered.
  - ‘unique’ means that there are no duplicates in a multi-valued property.
  - *prop-constraint* is an expression that specifies a constraint that applies to the property.

Class (cont.)

- Format of operations:

  [*visibility*] *name* ‘(‘ [*parameter-list*] ‘)’ [‘:’ [*return-type*]
  [‘{‘ *oper-property* [‘,’ *oper-property*]* ‘}’]]

  - Parameters:

    parameter-list ::= *parameter* [‘,’ *parameter*]*

    parameter ::= [*direction*] *parameter-name* ‘:’ *type-expression* [‘[‘*multiplicity*’]’] [‘=’ *default*]
    [‘{‘ *parm-property* [‘,’ *parm-property*]* ‘}’]

    – direction: ‘in’, ‘out’, ‘inout’ (defaults to ‘in’ if omitted)

  - Operation properties (modifiers):

    – ‘redefines’ *oper-name* means that the operation redefines an inherited operation identified by *oper-name*.

    – ‘query’ means that the operation does not change the state of the system.

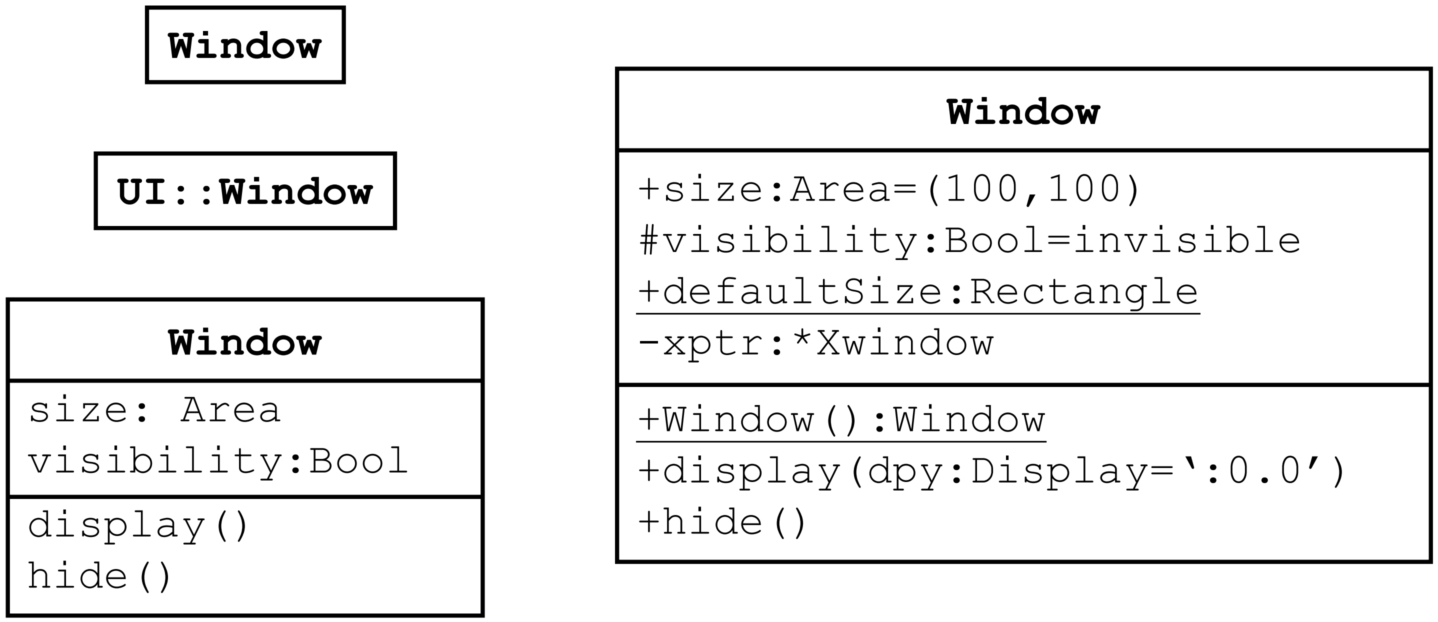
    – ‘ordered’ means that the values of the return parameter are ordered.

    – ‘unique’ means that the values returned by parameters have no duplicates.

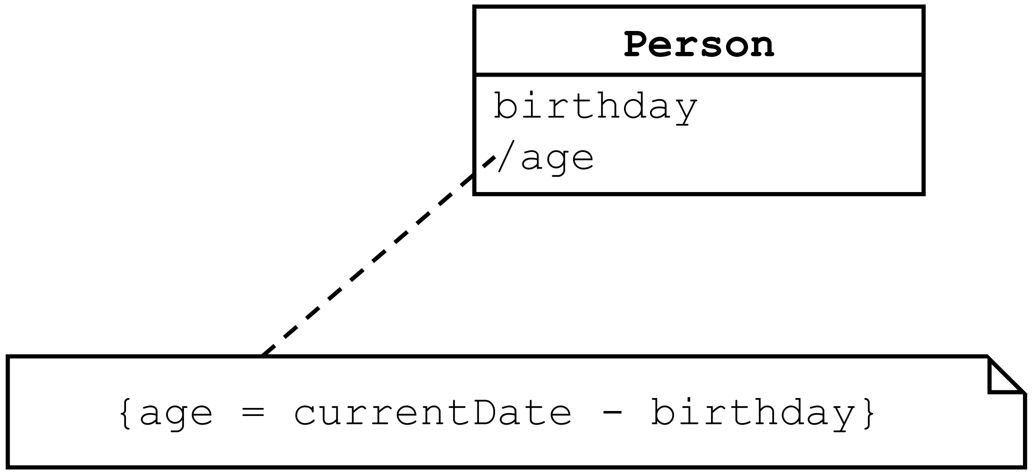
    – *oper-constraint* is a constraint that applies to the operation.

- Class (static) attributes and operations are underlined.

Examples of Classes



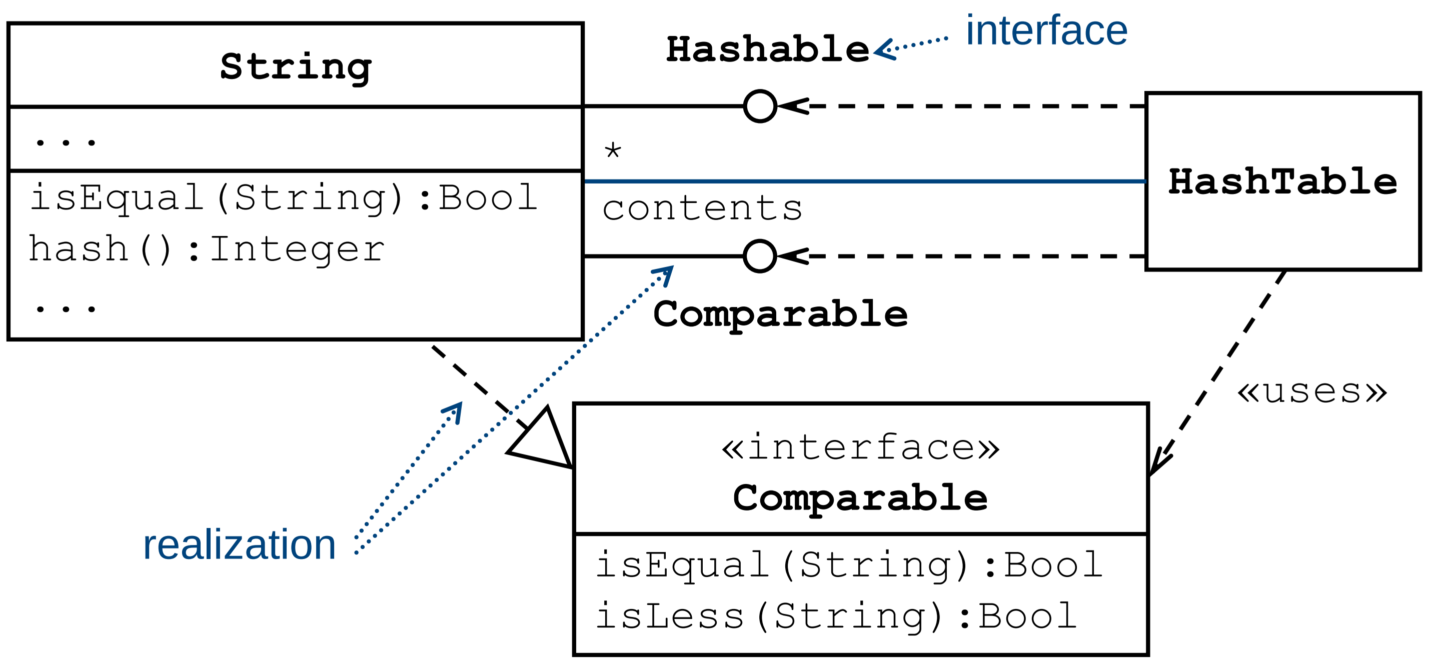
Example of Derived Attribute



Interface

→ A kind of classifier that represents a declaration of a set of coherent public features and obligations.

- Specifies a *contract*; any instance of a classifier that realizes the interface must fulfill that contract.
- An interface is ***not instantiable***; instead, an interface is *implemented* by an instantiable classifier, which means that the instantiable classifier presents a public facade that conforms to the interface specification.



Association

→ A relationship that can occur between typed instances.

- An association declares that there can be *links* between instances of the associated types.
  - A link is a tuple with one value for each end of the association, where each value is an instance of the type of the end.
- It has at least two ends represented by properties, each of which is connected to the type of the end.
- More than one end of the association may have the same type.
- Association end:
  - Association role name.
  - Multiplicity.
  - Ownership of the end by the association: indicated by a small circle.
  - Navigability:
    - navigable ——→
    - non-navigable ——×
    - unspecified ——

Association (cont.)

- Association end (cont.):
  - Visibility: +, -, #, ~
  - Aggregation kind (only for binary associations):
    - None.
    - Shared (for aggregation):
      - A weak relationship between the *whole* and its *parts*.
      - Parts can exist independently on the whole.
      - Also called “ownership by a reference”.
    - Composite (for composition):
      - A strong relationship between the *whole* and its *parts*.
      - A part instance must be included in at most one composite (whole) at a time. If a composite is deleted, all of its parts are normally deleted with it.
      - Compositions may be linked in a directed acyclic graph with transitive deletion characteristics.

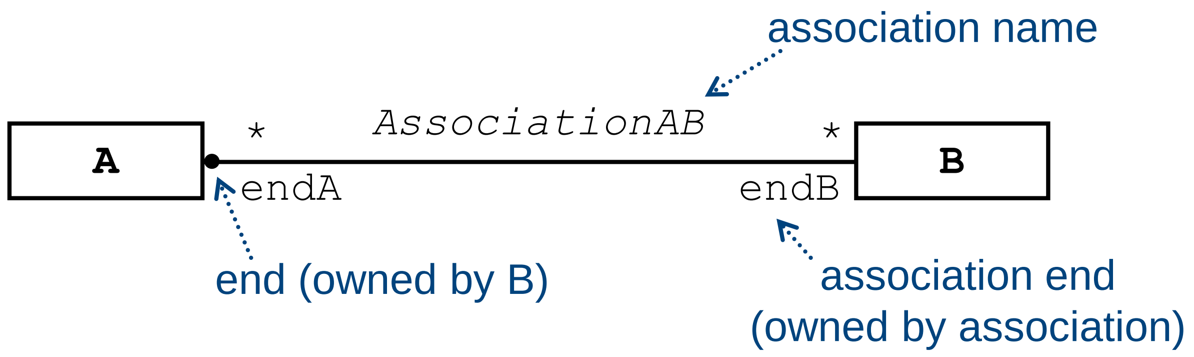
Association (cont.)

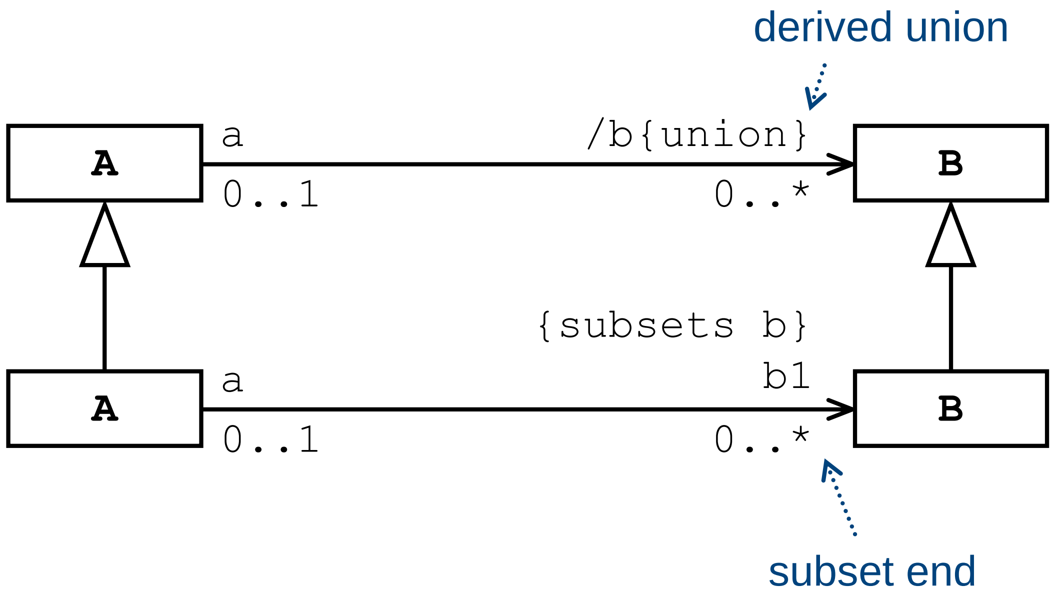
- Association end (cont.):
  - Property string (enclosed in curly braces):
    - {subsets *property-name*} to show that the end is a subset of the property called *property-name*.
    - {redefines *end-name*} to show that the end redefines the one named *end-name*.
    - {union} to show that the end is derived by being the union of its subsets.
    - {ordered} to show that the end represents an ordered set.
    - {bag} to show that the end represents a collection that permits the same element to appear more than once.
    - {sequence} or {seq} to show that the end represents a sequence (an ordered bag).
  - Qualifier: an attribute or a list of attributes whose values serve to partition the set of links.
- Association and its ends may be derived; marked by '/' before their names.

Association Class

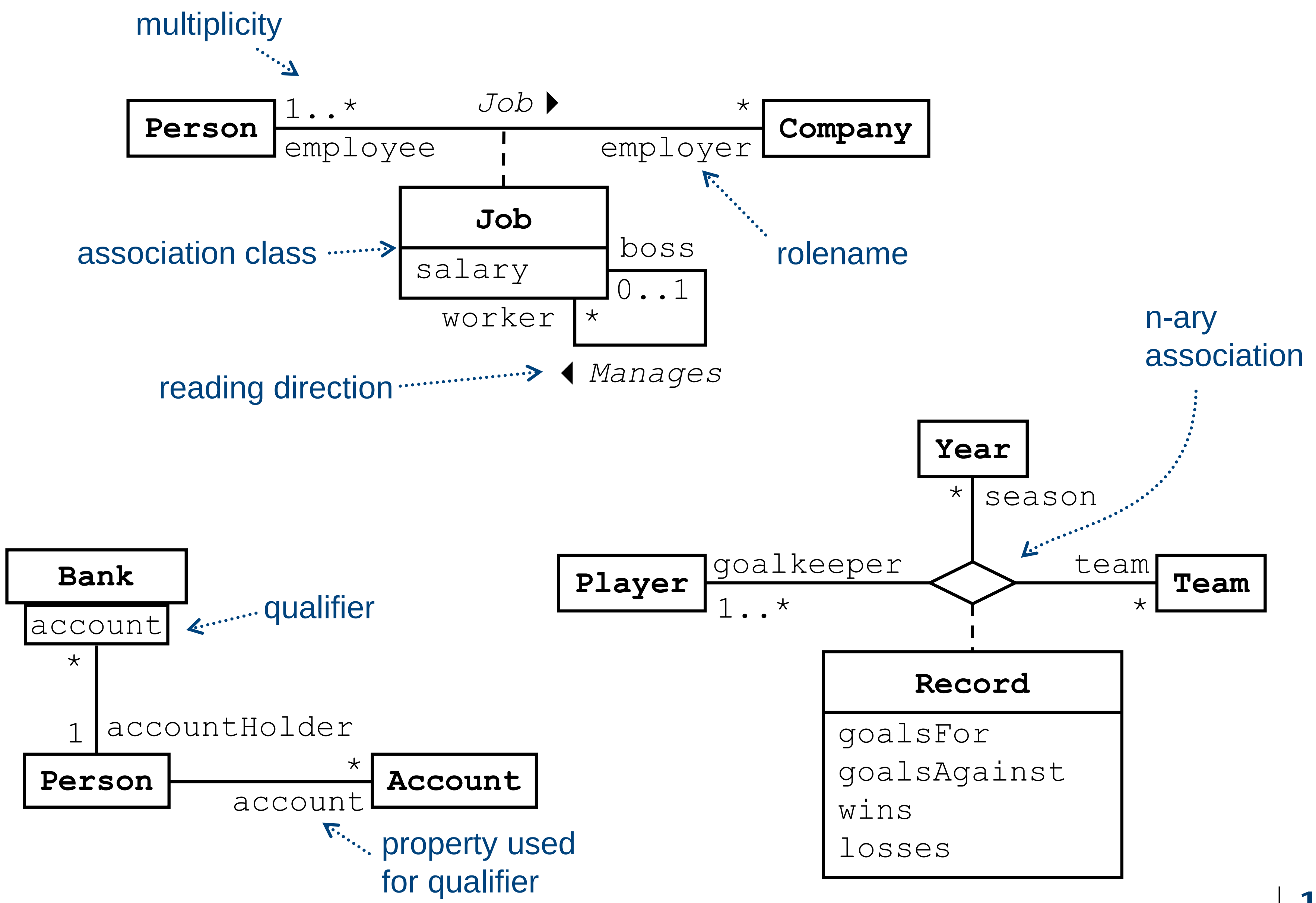
- → An association with class-like properties (attributes, operations, relations, behavior).
- ■ It not only connects a set of classifiers but also defines a set of features that belong to the relationship itself and not to any of the classifiers.
- ■ An association and its connected association class represent the same model element.
  - Therefore, they must have the same name.

Examples of Associations



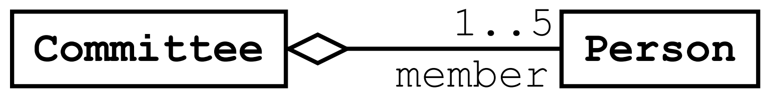


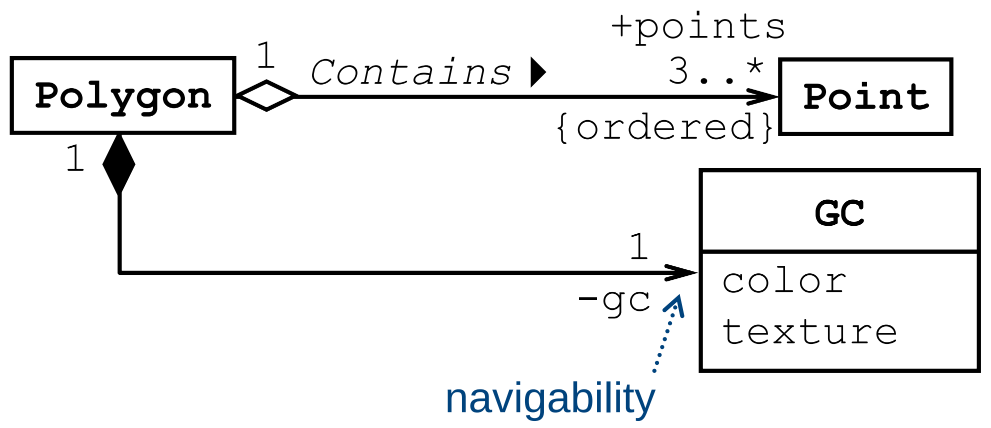
Examples of Associations and Assoc. Classes



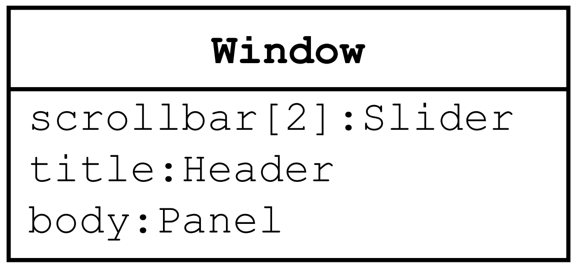
Examples of Aggregations and Compositions

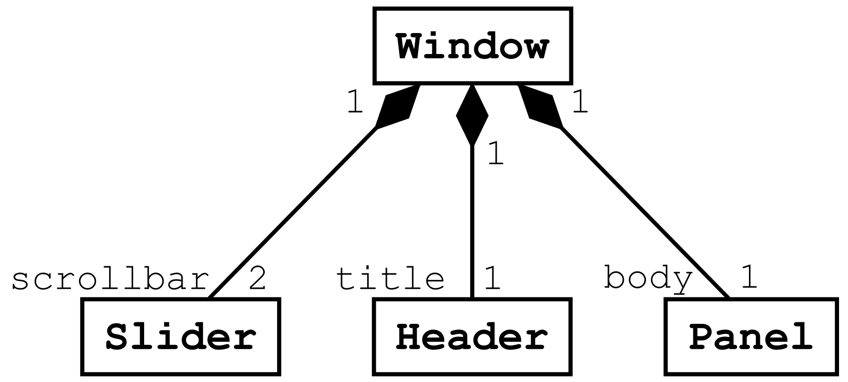
Aggregation:





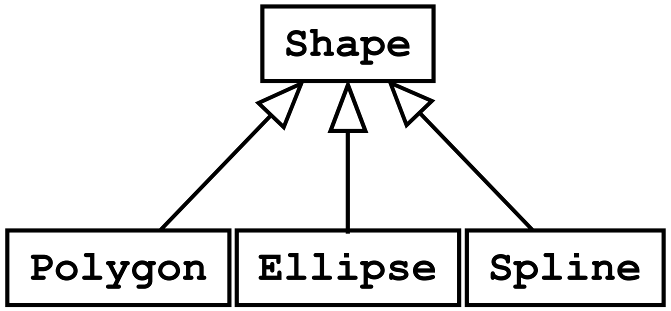
Composition:

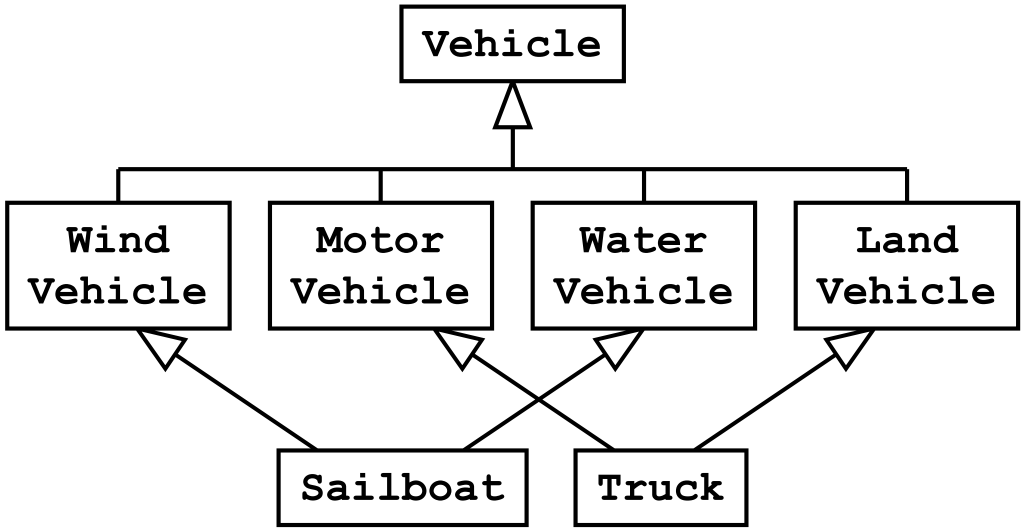




Generalization

- → The taxonomic relationship between a more general classifier and a more specific classifier.
- ■ The specific classifier inherits the features of the more general classifier.
- ■ Each instance of the specific classifier is also an indirect instance of the general classifier.



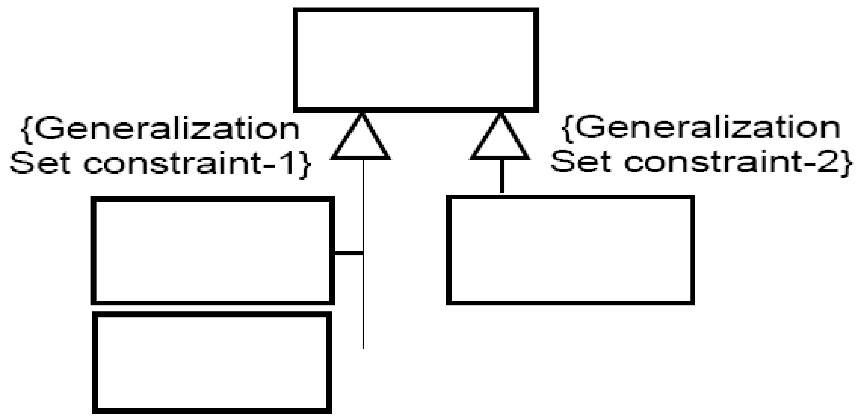


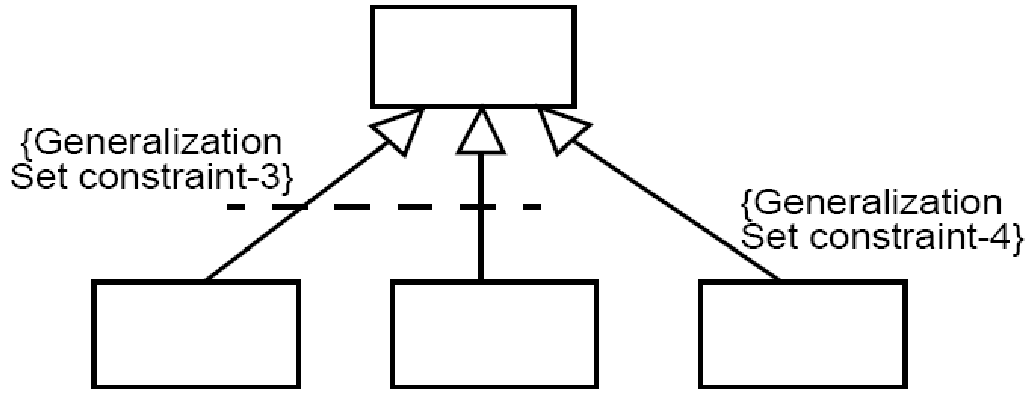
Generalization Set

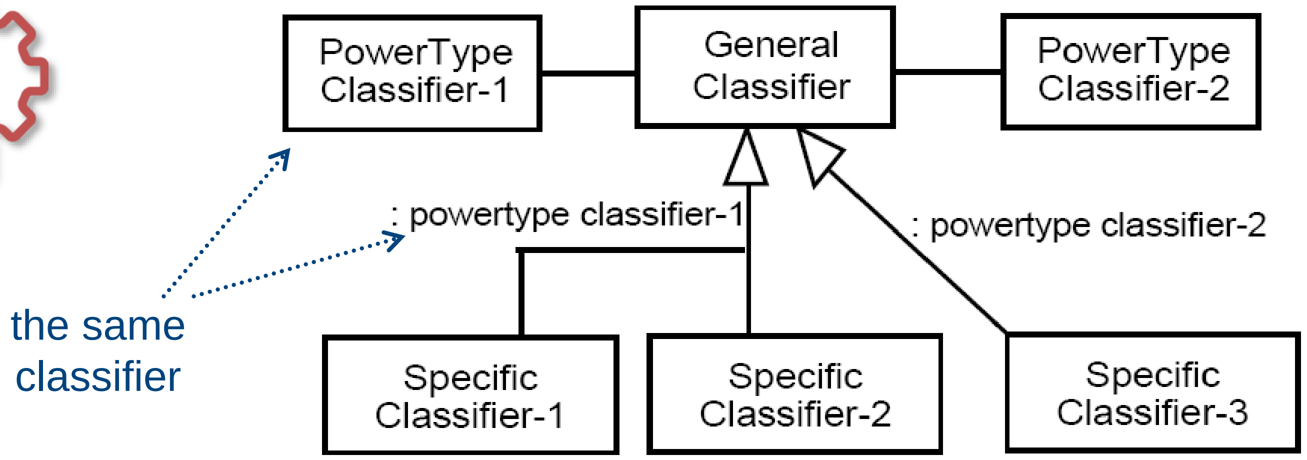
→ Defines a particular set of generalization relationships that describe the way in which a general classifier (or superclass) may be divided using specific subtypes.

- Usually, a generalization set describes a particular aspect of specialization.
- Covering and disjoint properties of a generalization set:
  - {complete, disjoint} - Indicates the generalization set is covering and its specific classifiers have no common instances.
  - {incomplete, disjoint} - Indicates the generalization set is not covering and its specific classifiers have no common instances.
  - {complete, overlapping} - Indicates the generalization set is covering and its specific classifiers do share common instances.
  - {incomplete, overlapping} - Indicates the generalization set is not covering and its specific classifiers do share common instances.
  - default is {incomplete, disjoint}
- Generalization set may define the ***powertype*** - a (meta)class whose instances are subclasses of another class.

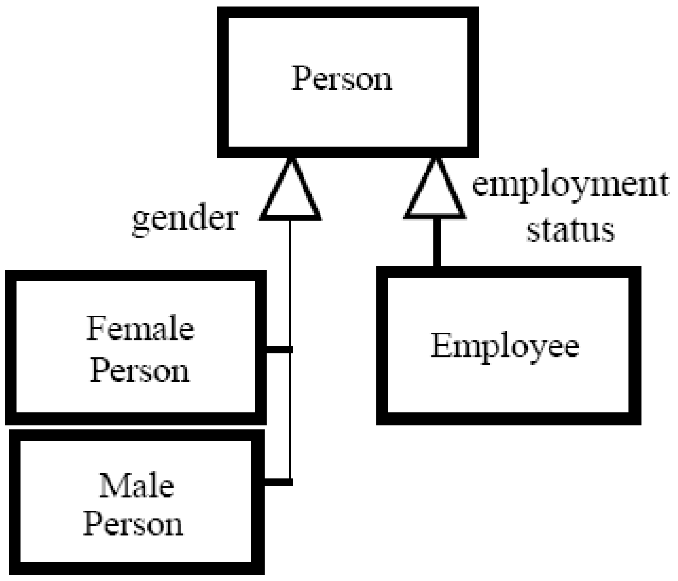
Possible Notations of Generalization Sets

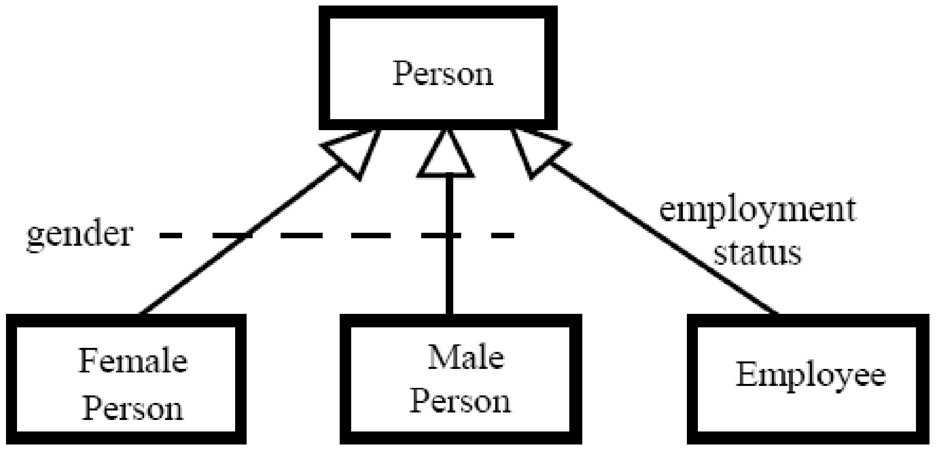

{Generalization Set constraint-1}
{Generalization Set constraint-2}

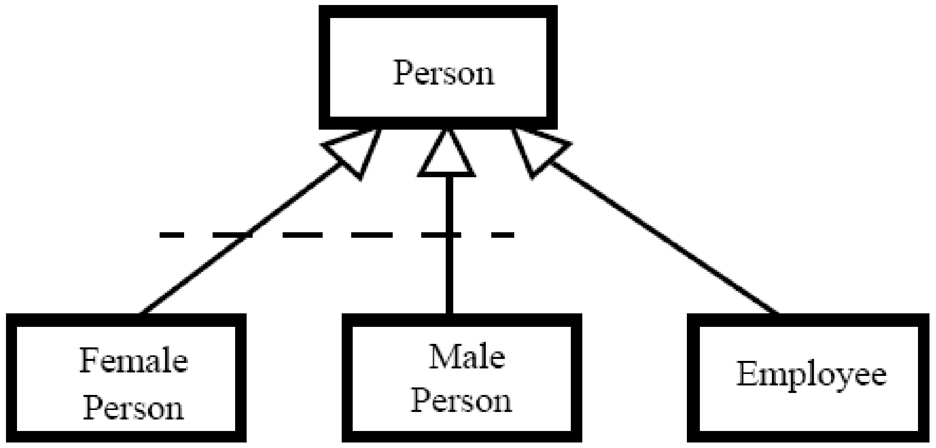

{Generalization Set constraint-3}
{Generalization Set constraint-4}



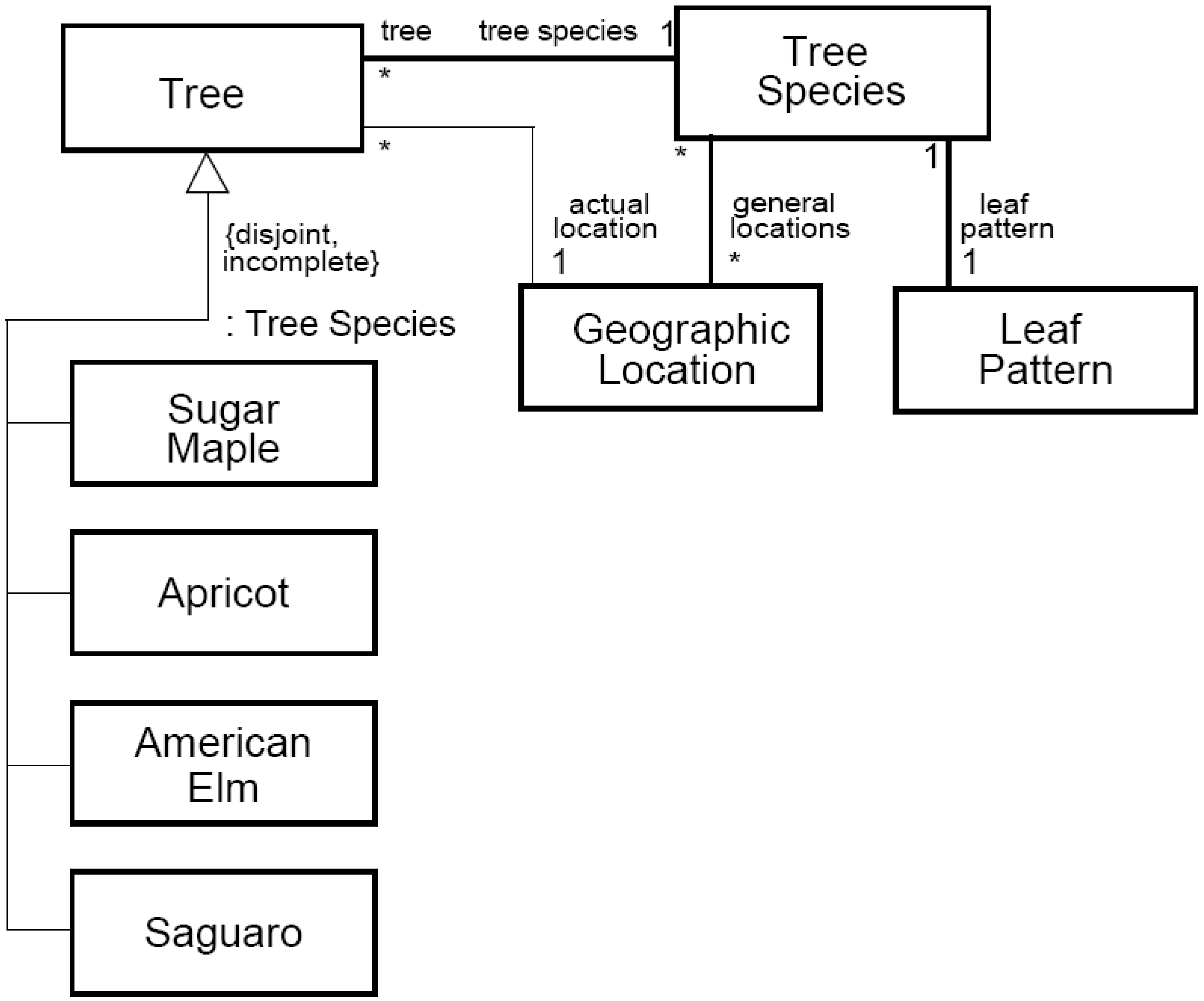
Examples of Generalization Sets (1)



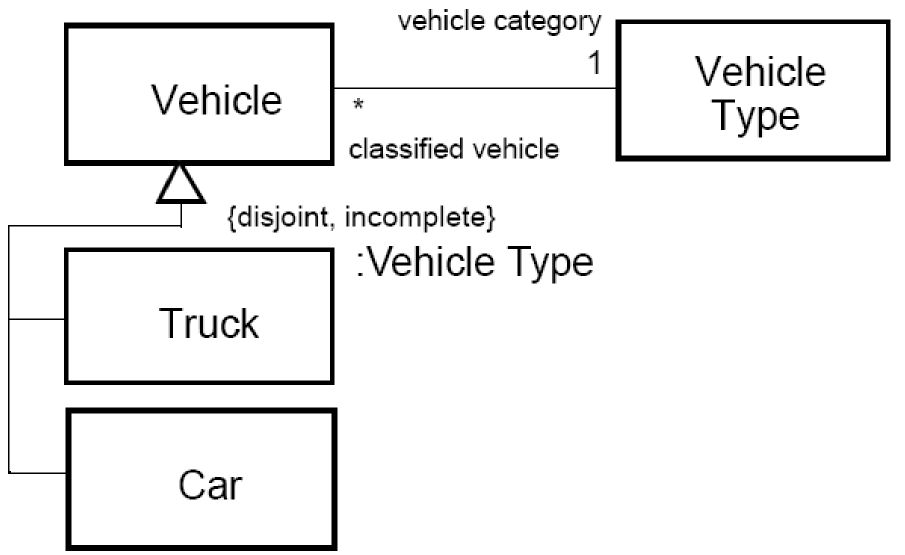


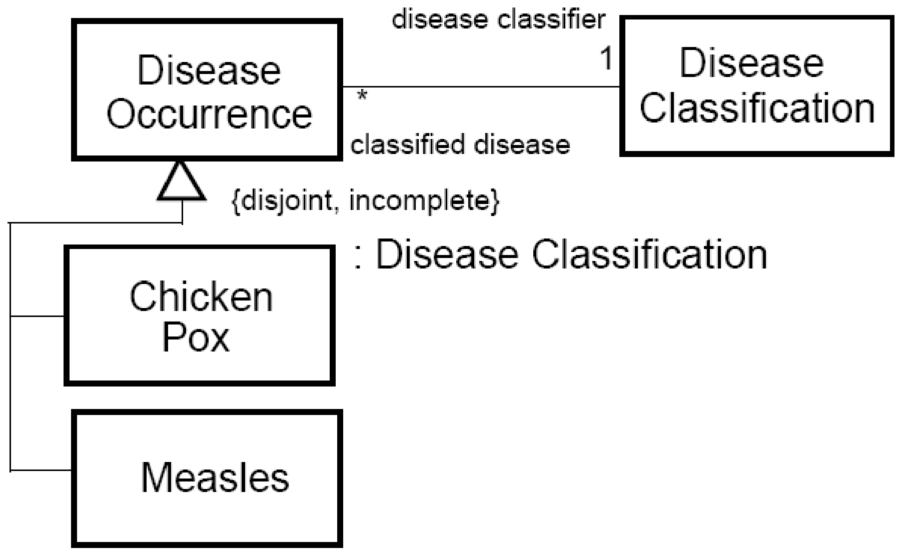


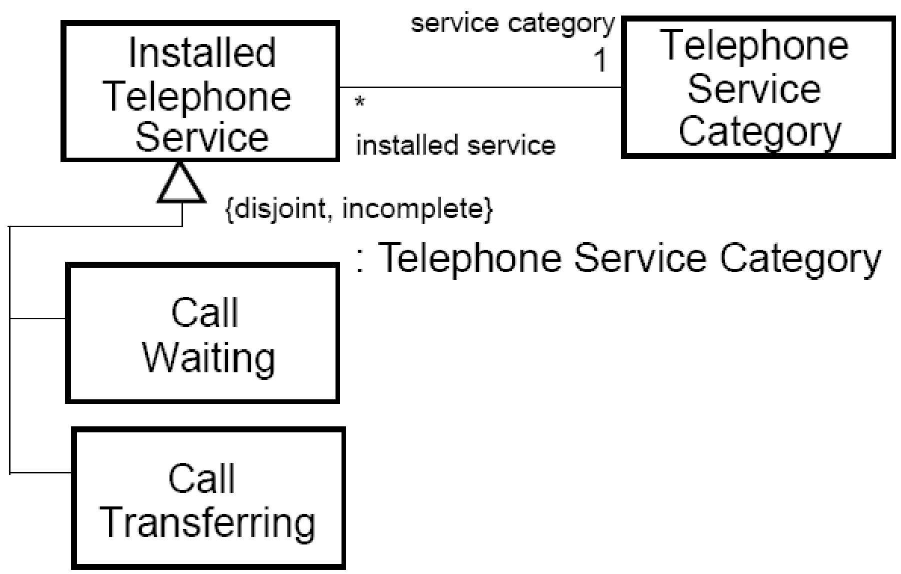
Examples of Generalization Sets (2)

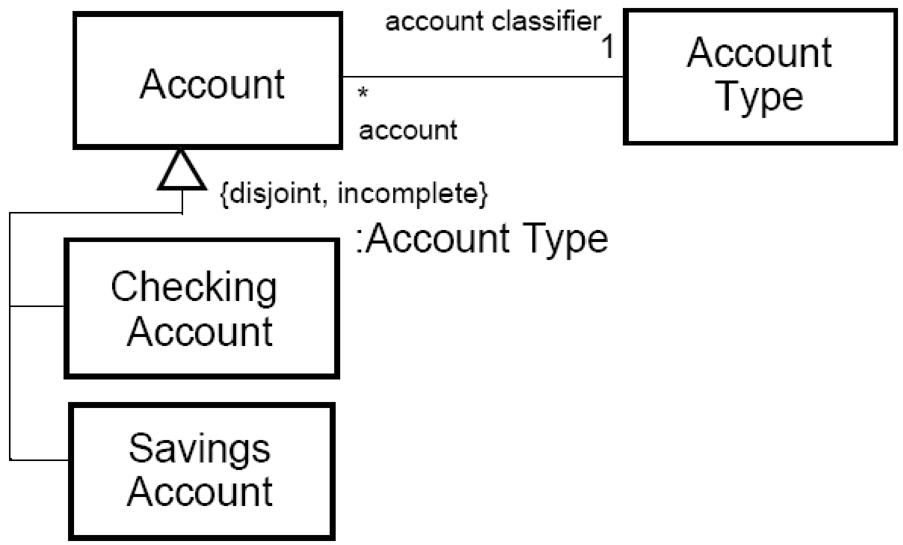


Examples of Generalization Sets (3)







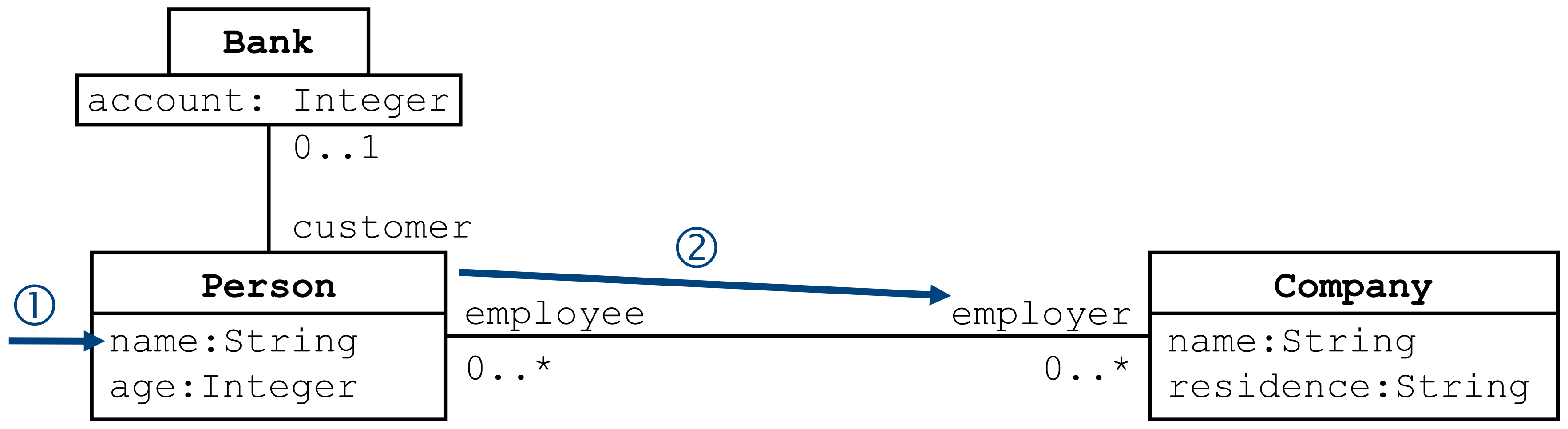


Navigation Expressions (OCL)

→ Allow to express navigation in 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*]
  - The *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.

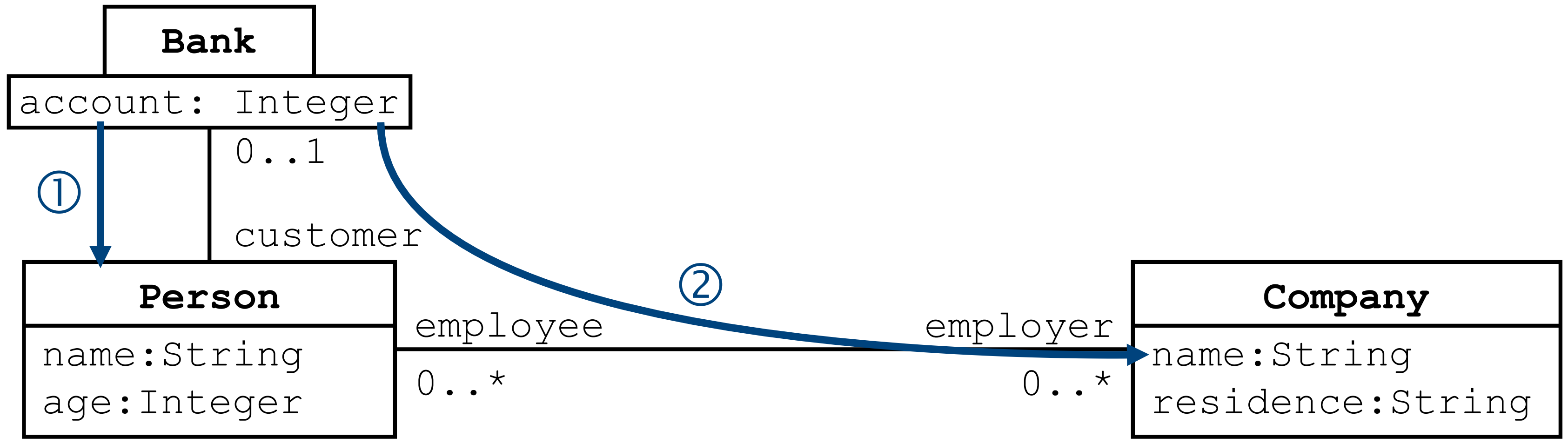
Examples of Model Navigation (1)



① Name of a person:
`Person.name`

② Names of person's employers:
`Person.employer.name`

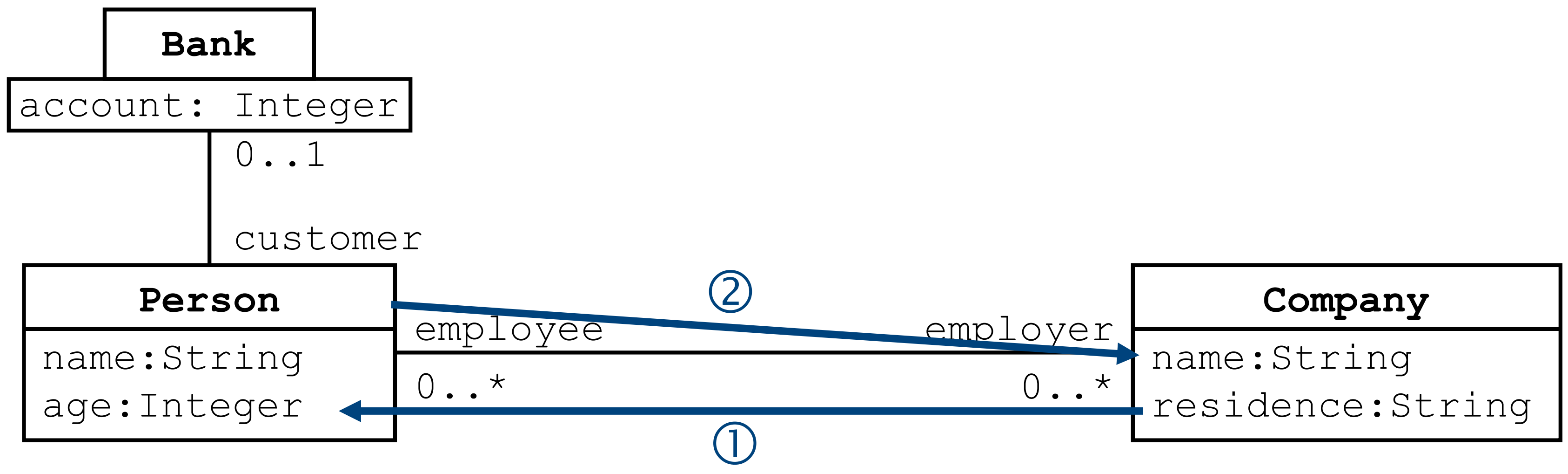
Examples of Model Navigation (2)



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`

Examples of Model 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`

Instance Specification

→ Representation of an instance in a modeled system.

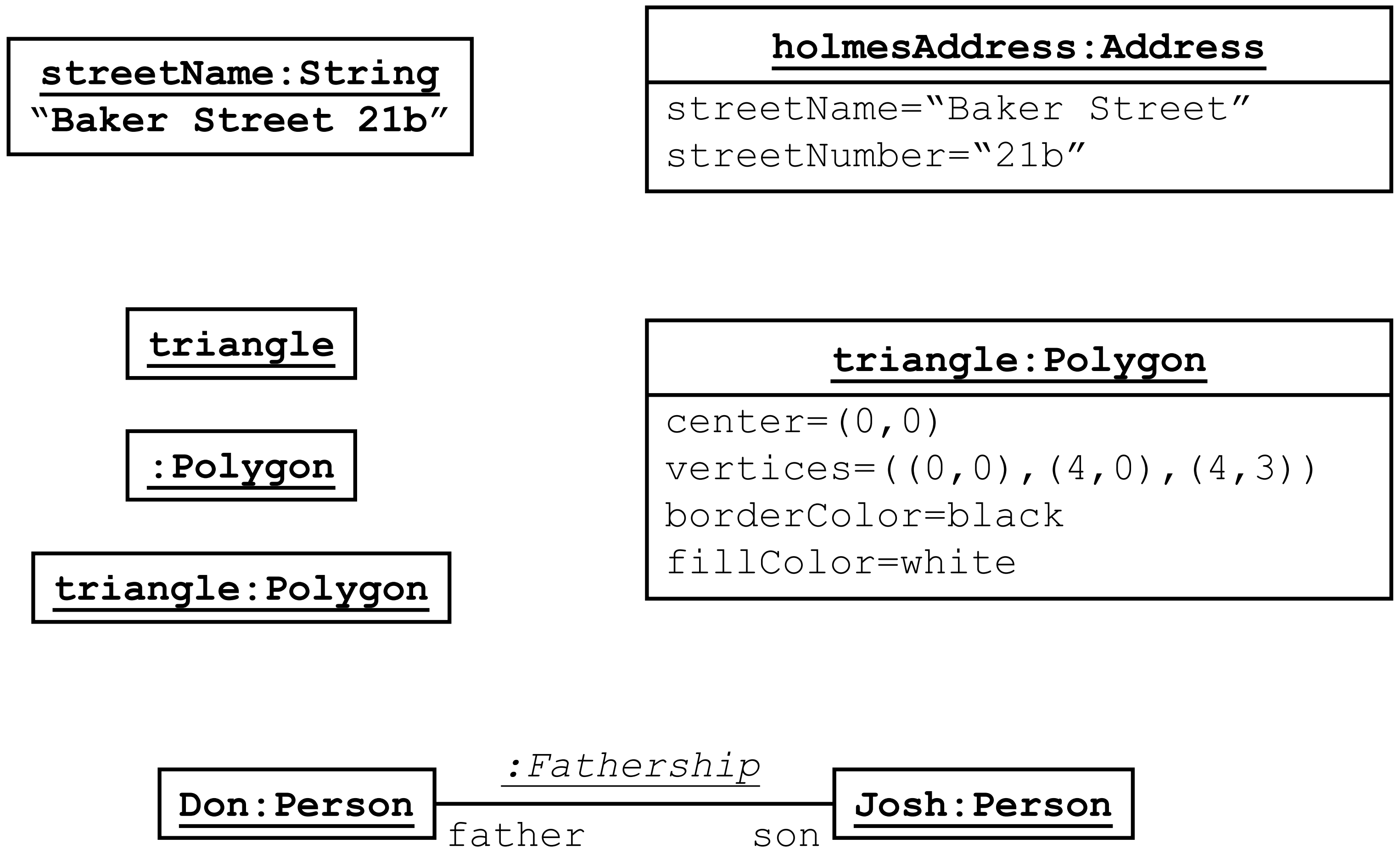
- Can specify name and one or more classifiers:

  [*name*] ‘:’ [*classifier-name* [‘,’ *classifier-name*]*]

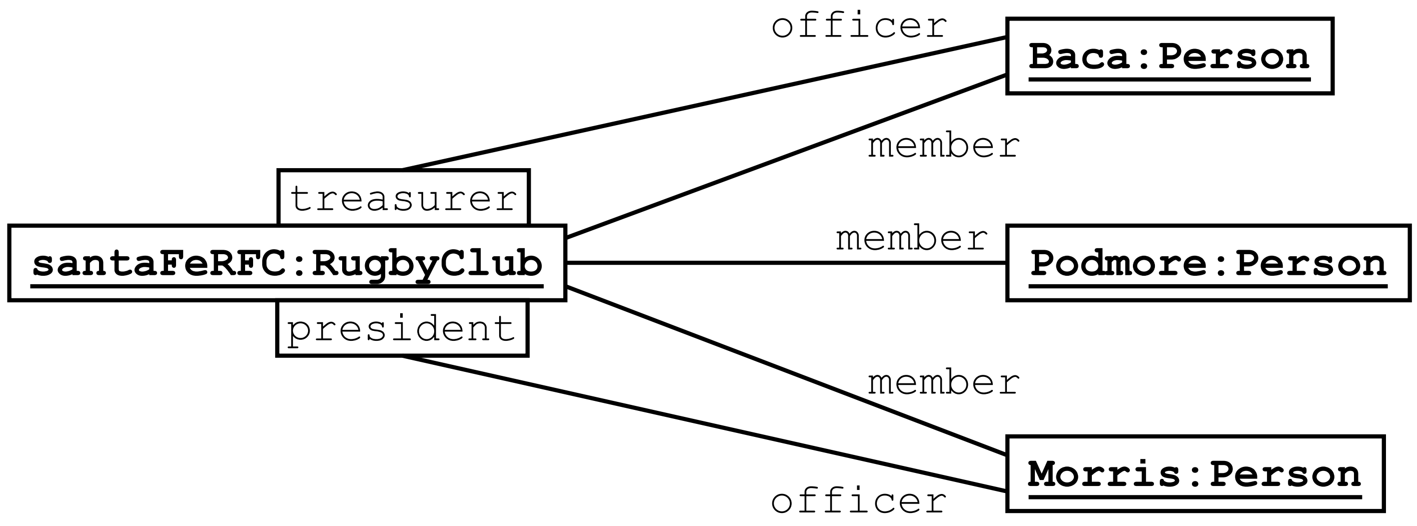
- Kind of the instance specification is given by its classifier(s). It can be:
  - Object
    - An instance of a class.
    - Can specify values of structural features of the entity–slots:

      [[*name*] [‘:’ *type*] ‘=‘] *value*
  - Link
    - A tuple (mostly a pair) of object references.
    - An instance of an association.
    - Association adornments can be shown, except of multiplicity.
  - etc.
- Visually, the instance specification shares the shape of its classifier(s).

Examples of Instance Specifications (1)



Examples of Instance Specifications (2)



Process of Class Modeling

1. Identify classes
   - From glossary.
   - From a business model or business-related artifacts.
   - From the stored information items and business artifacts.
   - From use case realizations.
2. Specify the semantics of classes
   - Responsibility.
   - Attributes, operations and interfaces.
3. Identify relationship among classes
   - Domain-based associations.
   - From object interactions.
   - Generalization and aggregation relationships.
4. Structure the model into packages
5. Repeat the process and refine the model.