

UML

Jeszenszky, Péter

University of Debrecen, Faculty of Informatics
jeszenszky.peter@inf.unideb.hu

Kocsis, Gergely (English version)

University of Debrecen, Faculty of Informatics
kocsis.gergely@inf.unideb.hu

Last modified: 23.03.2017

What is UML?

- *„The OMG's Unified Modeling Language (UML) helps you specify, visualize, and document models of software systems, including their structure and design [...]. (You can use UML for business modeling and modeling of other non-software systems too.)”*
 - Source: *Introduction To OMG's Unified Modeling Language* <http://www.omg.org/UML/what-is-uml.htm>

Design Principles

- Source: *OMG UML, Infrastructure, Version 2.4.1*
 - **Modularity:** Strong cohesion and loose coupling is applied to group constructs into packages and organize features into metaclasses.
 - **Layering:**
 - The package structure is layered to separate the metalanguage core constructs from the higher-level constructs.
 - A 4-layer metamodel architectural pattern is consistently applied to separate concerns
 - **Partitioning:** Partitioning is used to organize conceptual areas within the same layer.
 - **Extensibility**
 - **Reuse**

Object Management Group (OMG)

- The Object Management Group® (OMG®) is an international, open membership, not-for-profit technology standards consortium, founded in 1989. <http://www.omg.org/>
- Standards:
 - *Common Object Request Broker Architecture (CORBA)* <http://www.omg.org/corba/>
 - *MetaObject Facility (MOF)* <http://www.omg.org/mof/>
 - *Model Driven Architecture (MDA)* <http://www.omg.org/mda/>
 - *Unified Modeling Language (UML)* <http://www.uml.org/>
 - *XML Metadata Interchange (XMI)* <http://www.omg.org/spec/XMI/>
 - ...

History

- Previous solutions were object oriented software engineering methods:
 - Booch (Grady Booch)
 - OMT (*object-modeling technique*) (James E. Rumbaugh and others)
 - OOSE (*object-oriented software engineering*) (Ivar Jacobson)
- „*Three Amigos*”: Booch, Jacobson and Rumbaugh
 - By the lead of these three people the construction of UML has started
- See more:
 - *The Unified Modeling Language – Versions of UML*
<http://www.uml-diagrams.org/>

The current standard

- *OMG Unified Modeling Language (OMG UML) Version 2.5*. March 2015. <http://www.omg.org/spec/UML/2.5/>
- *Diagram Definition (DD) Version 1.1*. June 2015. <http://www.omg.org/spec/DD/1.1/>
- *XML Metadata Interchange (XMI) Version 2.5.1*. June 2015. <http://www.omg.org/spec/XMI/2.5.1/>
- *OMG Meta Object Facility (MOF) Core Specification Version 2.5*. June 2015. <http://www.omg.org/spec/MOF/2.5/>
- *Object Constraint Language Version 2.4*. February 2014. <http://www.omg.org/spec/OCL/2.4/>

UML 2.5

- The specification is a *minor revision of the previous release*
- The text of the specification has been rewritten at many points for the sake of readability. Redundancies have been removed. Some parts have been explained better.
 - e.g. there are less forward references in the text.
- The whole UML specification is one document.
 - In the previous version there were two separate documents (*Infrastructure*, *Superstructure*), see:
<http://www.omg.org/spec/UML/2.4.1/>

Object Constraint Language (OCL)

- A formal language with which expressions can be written about UML models
- Not a programming but a modeling language language!
 - OCL expressions cannot be executed
- OCL expressions do not have side effects
- Strongly typed
- There are multiple uses of it:
 - Query language.
 - Describe invariant constraints about classes and types
 - Describe pre-, and post conditions of methods and operations
 - ...

XML Metadata Interchange (XMI)

- XML format used to exchange metadata between applications
- It is most widely used as a tool for exchanging UML models, but also can be used for other purposes

Diagram Definition (DD)

- The Diagram Definition (DD) specification provides a basis for modeling and interchanging graphical notations, specifically node and arc style diagrams as found in UML, SysML, and BPMN, for example, where the notations are tied to abstract language syntaxes defined with MOF.
- It provides a framework for other modeling languages to specify their own diagrams

Model

- A model is a description of a system, where the word „system” can mean e.g. software, organizations, processes, etc...).
- The model describes the system from a given viewpoint for a given group of audience (e.g. the users or the developers) at a given level of abstraction
- It is full in the sense that it covers the whole system however only those aspects are included that are important for the given purposes

Metamodel

- Model of the model.
- In UML, metamodel is a model modeling itself
 - It can be used to model other models and meta-models as well (besides itself)
 - e.g. MOF model is a metamodel (<http://www.omg.org/mof/>)

!!br0ken!!

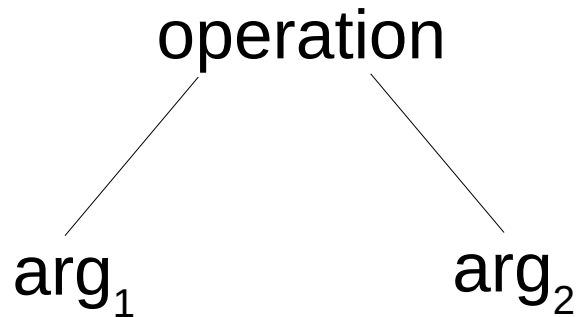
- In object-oriented programming, a metaclass is a class whose instances are classes.
 - Python:
 - *PEP 3115 – Metaclasses in Python 3000* <https://www.python.org/dev/peps/pep-3115/>
 - Python 3.6.0 documentation – Data model – Metaclasses <https://docs.python.org/3/reference/datamodel.html#metaclasses>
 - Groovy:
 - *The Groovy programming language – Runtime and compile-time metaprogramming* <http://groovy-lang.org/metaprogramming.html>
 - `groovy.lang.MetaClass` <http://docs.groovy-lang.org/latest/html/api/groovy/lang/MetaClass.html>
 - UML: a metaclass is a class in the metamodel (e.g. `Element`, `Classifier`, ...).

Syntax (1)

- **Abstract syntax:** A form of representation of language elements that is independent of machine-oriented structures and encoding and also of the physical representation of them.
- **Concrete syntax:** The concrete syntax is defined by mapping the notation onto the abstract syntax.

Syntax (2)

- Abstract syntax:



- Concrete syntax:

- Infix form:

- $(1 + 2) * 3$

- Prefix form:

- $(* (+ 1 2) 3)$

- Postfix form:

- $((1 2 +) 3 *)$

Syntax (3)

- The abstract syntax of UML is given by a UML model, that is called the **UML metamodel**

Semantics

- „*Colorless green ideas sleep furiously.*” /
 - Source: Noam Chomsky. *Syntactic Structures*. Mouton & Co., 1957.
- The fact that something is syntactically correct does not mean that it has a semantic meaning

Meta Object Facility (MOF)

- The Meta Object Facility (MOF) provides an open and platform-independent metadata management framework and associated set of metadata services to enable the development and interoperability of model and metadata driven systems.
 - Source: *OMG Meta Object Facility (MOF) Core Specification, Version 2.5*

Meta Object Facility (MOF)

- **MOF is a DSL (*domain-specific language*):** a computer language that's targeted to a particular kind of problem, rather than a general purpose language that's aimed at any kind of software problem
 - e.g.: BibTeX/LaTeX, CSS, DOT (Graphviz), Gradle DSL, SQL, ...
 - See: Martin Fowler. *DomainSpecificLanguage*.
<https://martinfowler.com/bliki/DomainSpecificLanguage.html>
- MOF is a DSL targeting the definition of metamodels

Meta Object Facility (MOF)

- Reuses the modeling facilities of UML
 - It shares and extends a common meta-model with UML (e.g. with reflection).
- It is used to model itself and other metamodels (e.g. UML, CWM)
 - It can be used for any type of metadata (e.g. software configuration or specification metadata)

Meta Object Facility (MOF)

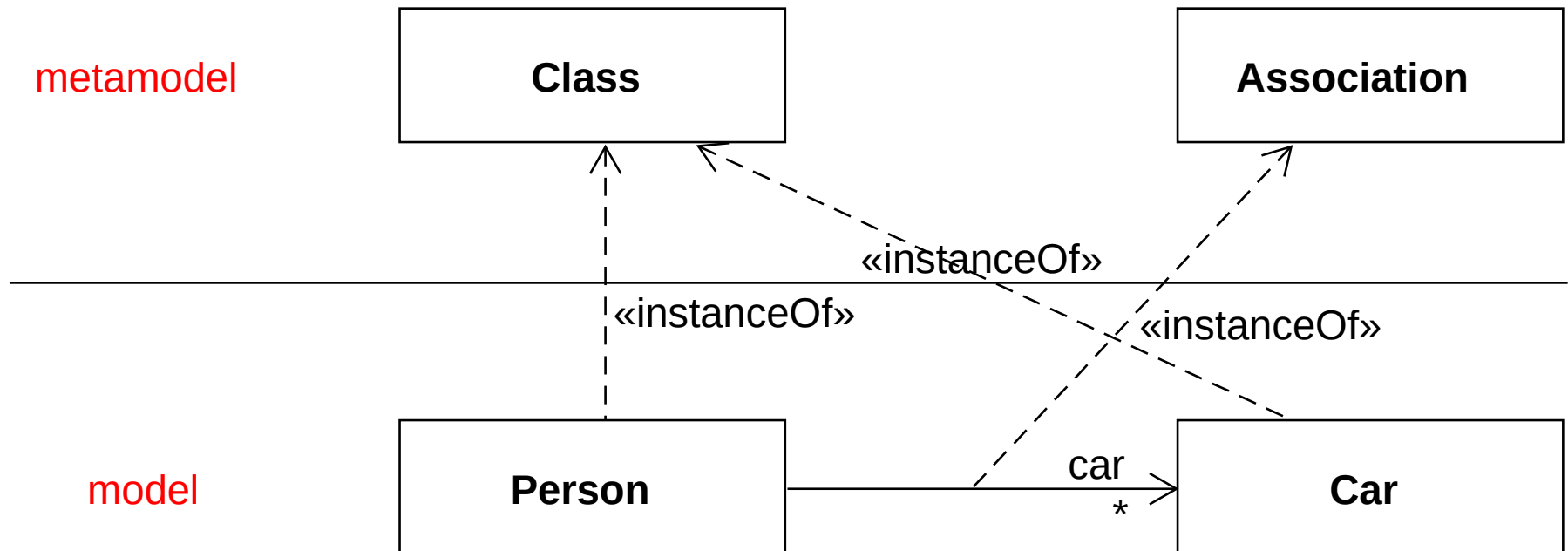
- **UML metamodel:**
 - A UML model defining the abstract syntax of UML
 - A metamodel using the constructions of only a subset of UML, specified by the MOF specification
 - It is a metamodel of metamodels

4-layer metamodel architectural pattern (UML 2.4.1)

- When defining languages usually 3 layers are used:
 - The specification of the language (metamodel)
 - The user specification (model)
 - Objects of the model

4-layer metamodel architectural pattern (UML 2.4.1)

- Metamodeling:



4-layer metamodel architectural pattern (UML 2.4.1)

- **Meta-metamodel:**

- The main goal is to provide a language that can describe metamodels
- Usually a meta-metamodel is more robust than the described metamodel
- Usually it is required that the meta-metamodel and the metamodel have the same design concepts and structures.

-

- An instance of a meta-metamodel, which also means that all elements in the metamodel are instances of the elements of the meta-metamodel
- The main goal is to define a language with which models can be described

- **Model:**

- An instance of a metamodel
- The main goal is to model problems of different areas (e.g. software, business processes, requirements)
- Usually it builds up from model elements

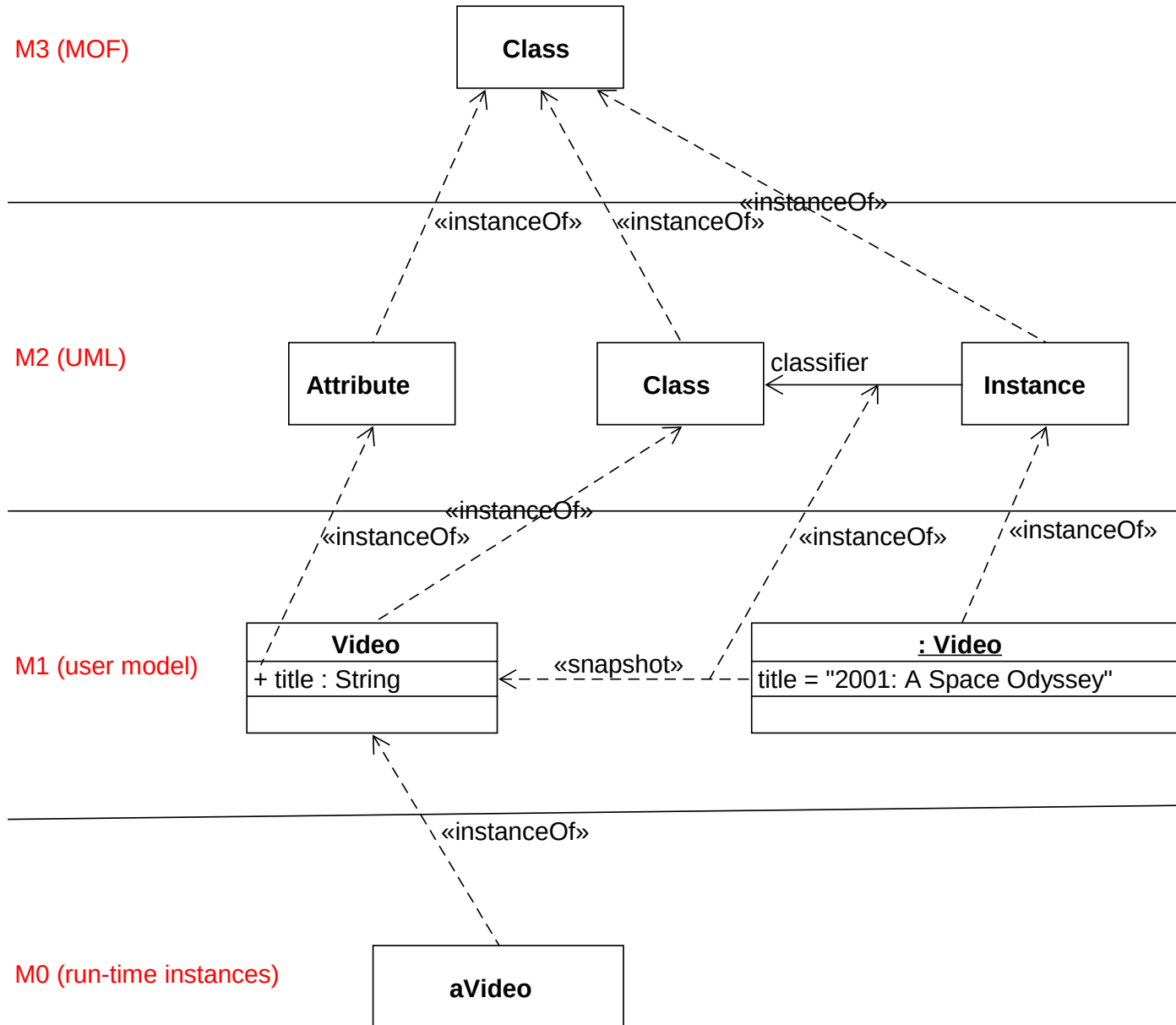
- **Runtime instances:**

- Contains runtime instances of model elements defined in the model

4-layer metamodel architectural pattern (UML 2.4.1)

- A model can be a metamodel and vice-verse in different cases e.g.: UML and MOF.
 - UML and MOF are meta-models, with which users can define their own models
 - From the point of view of MOF UML is a user specification (model) that relies on the MOF metamodel

4-layer metamodel architectural pattern (UML 2.4.1)



Meta Object Facility (MOF)

- The key concepts are *class*, *object* and the capability of *navigation* from an object to its meta-object
 - The latter is done through ***reflection***
- By the use of these any number of meta-layers can be handled
- Most systems use only a small number of layers (4 or less)
 - e.g. general reflective systems use two layers (class and instance)

Meta Object Facility (MOF)

- Two main packages:
 - **Essential MOF (EMOF)**: provides metamodeling features similar to object oriented languages for the creation of simple metamodels
 - **Complete MOF (CMOF)**: full metamodeling toolbox
 - The UML is defined e.g. by this

Model entities (1)

- Major categories of UML model elements:
 - **Classifiers.** A classifier describes a set of objects. An object is an individual with a state and relationships to other objects.
 - **Events:** An event describes a set of possible occurrences. An occurrence is something that happens that has some consequence with regard to the system.
 - **Behaviors:** A behavior describes a set of possible executions. An execution is a performance of a set of actions (potentially over some period of time) that may generate and respond to occurrences of events,

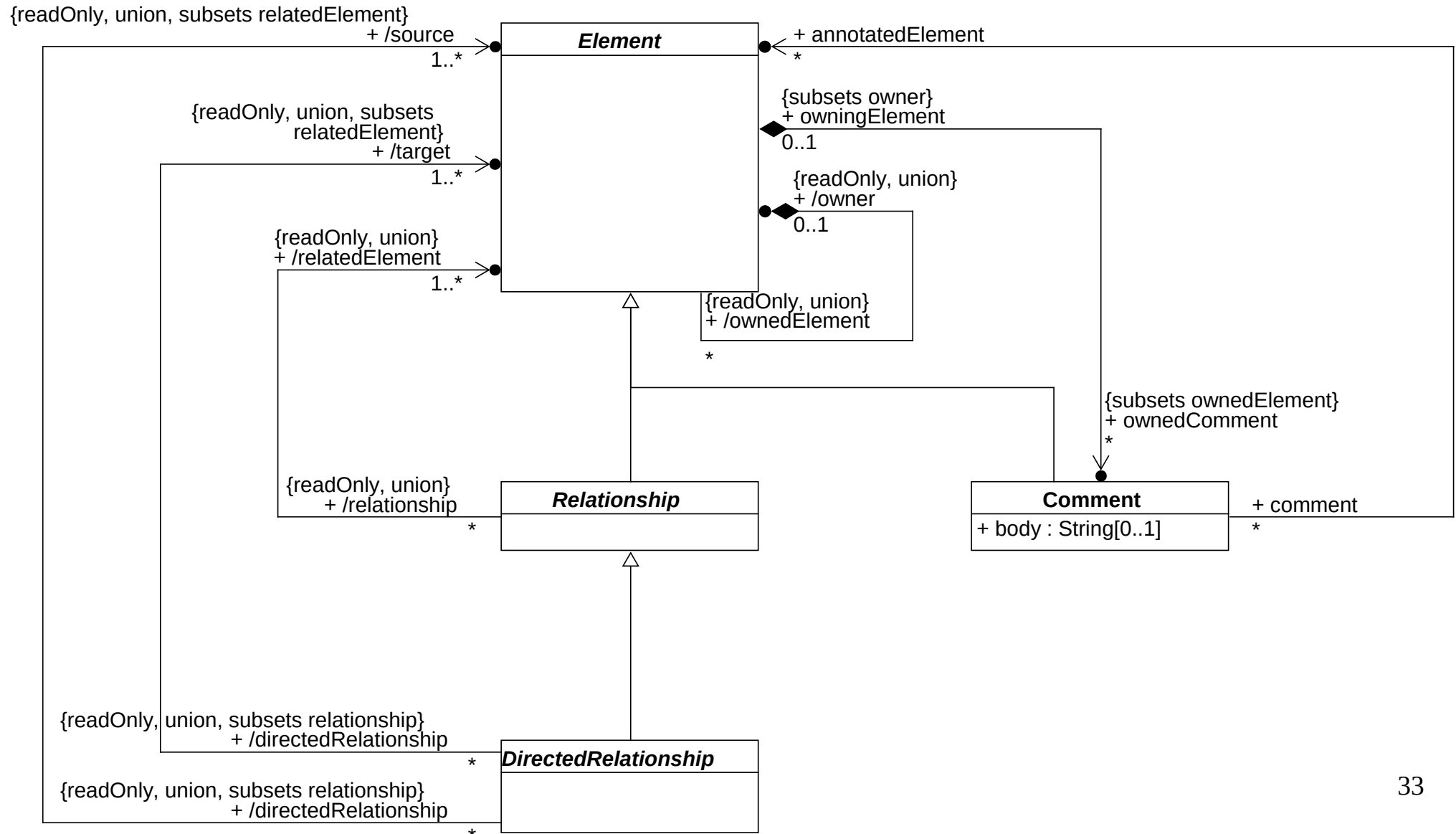
Model entities (2)

- UML models do not contain objects, occurrences, or executions, because such individuals are part of the domain being modeled, not the content of the models themselves!
 - UML does have modeling constructs for directly modeling Individuals however, these are just model elements

Classifier

- A UML model element that groups together a set of instances having common attributes
- A classifier has properties (attributes and methods).
- They can be organized to a hierarchy.
- Specializations: `DataType`, `Association`, `Interface`, `Class`, ...
- Notation: just like classes (use **boldface** letters)

UML definitions



UML diagram types (1)

- The 2 types of UML diagrams are:
 - **Structure diagrams:**
 - They describe the static structure of objects in the system
 - Time independent elements are described
 - e.g. definitions of an application
 - **Behavior diagrams:**
 - They describe the dynamic behavior of objects in the system including cooperation, activities and state-changes
 - The dynamic behavior of the system can be described as a sequence of changes of the system in time
- The two categories are not distinct. Diagram types can be combined.

UML diagram types (2)

- **Structure diagrams:**

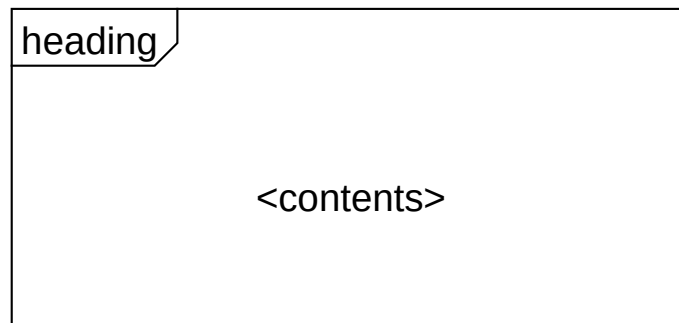
- *package diagram*
- *component diagram*
- *object diagram*
- *class diagram*
- *composite structure diagram*
- *profile diagram*
- *deployment diagram*

- **Behavior diagrams:**

- *activity diagram*
- *state machine diagram*
- *use case diagram*
- *interaction diagram*
 - *timing diagram*
 - *communication diagram*
 - *interaction overview diagram*
 - *sequence diagram*

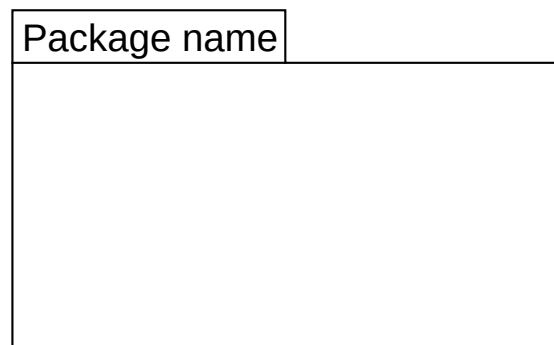
Diagram

- Optionally they can have a frame and heading



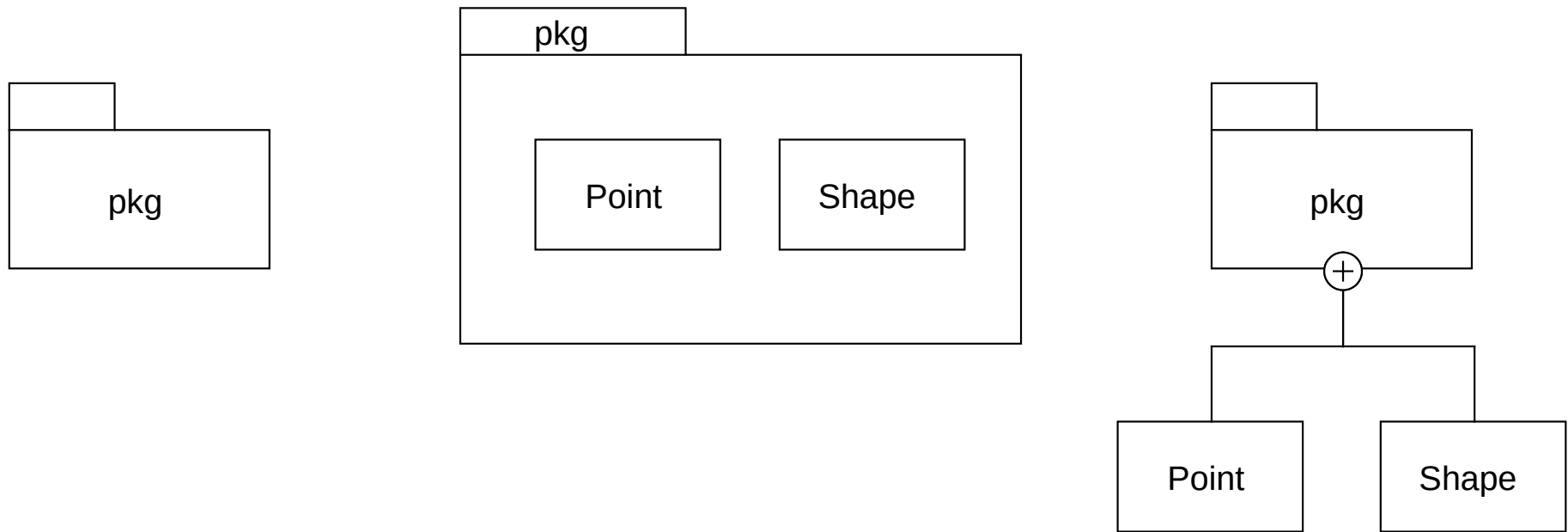
Package (1)

- It is used to group model elements. A package is a namespace
- Notation:



Package (2)

- The contained elements can be referred as *package_name::element_name* (e.g. `pkg::Point`, `pkg::Shape`).



Keyword, Stereotype

- Notation: between « and »
 - If these characters are not present in the character set >> and << are to be used.
- More than one keyword or stereotype can be used on one model element
 - They can be listed after each other (each one separately closed between « and »)
 - More than one can be given separated by commas all together between « and »

Keyword

- A word that is a part of the UML notation
- It is a textual annotation connected to a graphical element or as a part of a diagram
 - For each keyword the places of possible uses are given.
- With it similarly noted graphical UML elements can be distinguished (e.g. «interface»).

Stereotype (1)

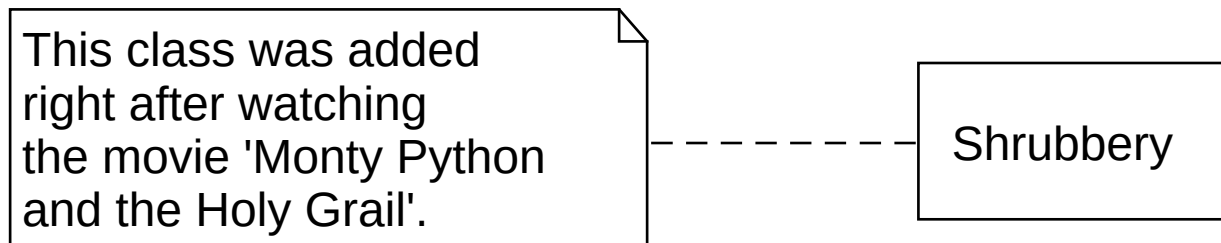
- Defines the extension of one or more metaclasses.
- It makes possible the use of platform-, and domain specific terminology
- A restricted form of metaclasses that cannot be used alone, just together with an extended metaclass.

Stereotype (2)

- The stereotype has to be given above or before (or instead of) the model element when applied..
- A stereotype can change the graphical appearance of the extended model element by the use of icons.
- Names of stereotypes usually start with capital letters
 - E.g.: «Create», «Instantiate», «Metaclass», ...
- The standard provides a lot of pre-defined stereotypes

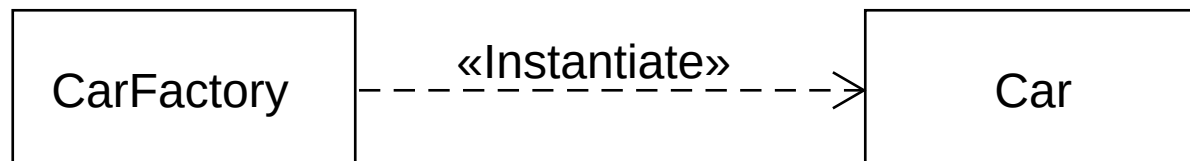
Comment

- Provides information for the reader of the model
- Notation:



Dependency

- Defines provider-client connection between model elements, where the modification of the provider may have an effect on the client
- Notation:
 - A dashed arrow to the client. A keyword/stereotype can be optionally given.

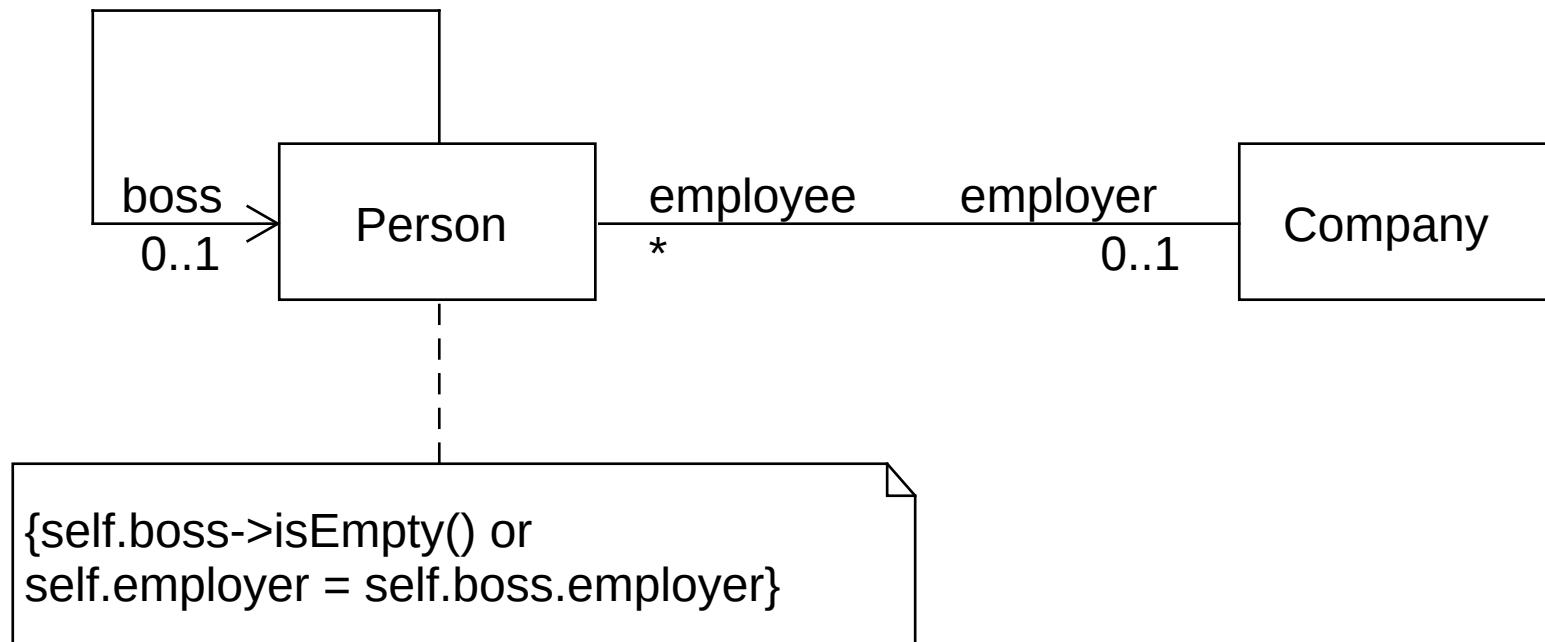


Constraint (1)

- A statement that describes a restriction that has to be fulfilled by all the valid realizations of model containing the constraint
- Notation:
 - { *[name:] logical-expression* }
 - The user constraints can be given in any language (e.g. a formal language like OCL, or a programming language like Java)

Constraint (2)

- Generally a constraint is given as a comment connected to those element by a dashed line on which the constraint has to be true

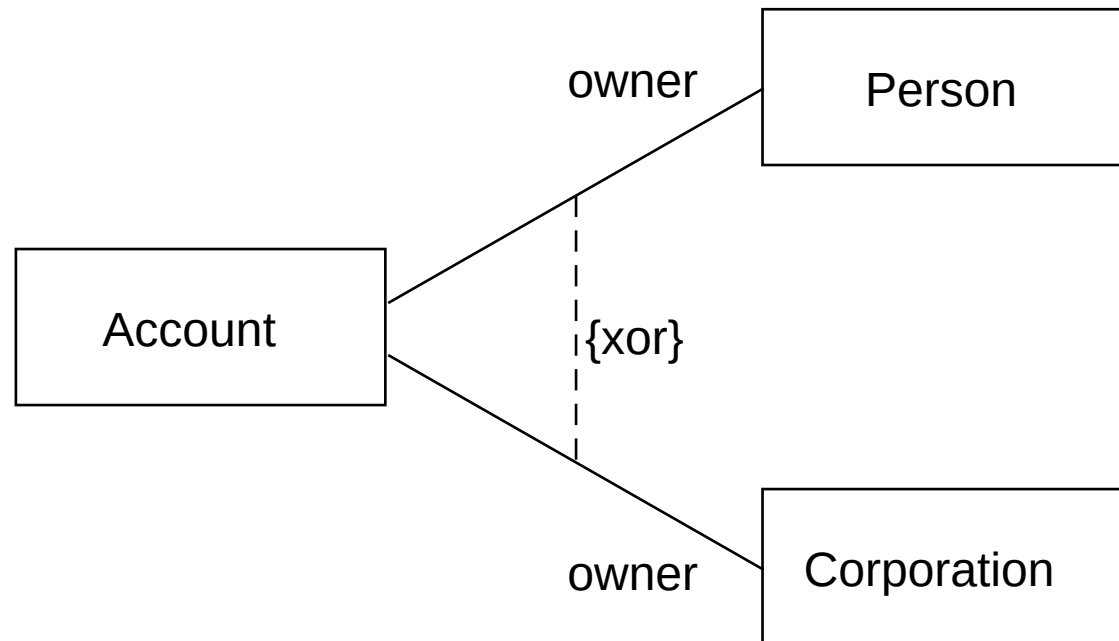


Constraint (3)

- If a constraint is for only one model element then it is enough to place it close to the element (if it has a name then close to the name)
- If the model element is represented by a character sequence then the constraint is to be put after it.

Constraint (4)

- A constraint for two model elements can be marked by writing it on a dashed line that connects the two elements.
 - Example:



Class diagram (1)

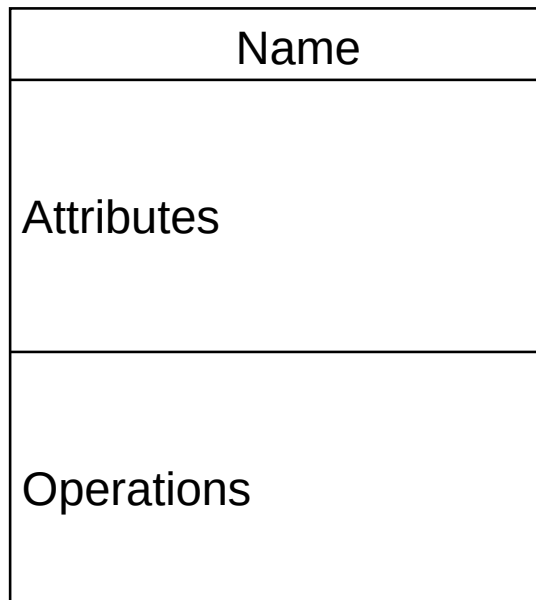
- A **class diagram** describes the types of objects in the system and the various kinds of static relationships that exist among them. Class diagrams also show the properties and operations of a class and the constraints that apply to the way objects are connected. The UML uses the term **feature** as a general term that covers properties and operations of a class.
 - Source: Martin Fowler. *UML Distilled – A Brief Guide to the Standard Object Modeling Language*. Third Edition. Addison-Wesley, 2003.
- „[...] a class diagram is a diagram where the primary symbols in the contents area are class symbols.”
 - Source: *OMG UML Version 2.5*

Class diagram (2)

- Class diagram types (Störrle):
 - **Analysis:** At the analysis level the classes are definitions of the domain. The class diagram models the structure of the domain.
 - **Design:** The methods of realization appear in the classes
 - **Operational:** The classes are equivalent to the constructions of the implementing language (e.g. C++, Java)

Class

- Notation:



Visibility

- + (public)
- - (private)
- # (protected)
- ~ (package private)

Multiplicity

- Describes a constraint on the number of elements of a collection
 - The number of elements cannot be lower than the lower limit
 - The number of elements cannot be bigger than the upper limit, if it is not $*$.
- Notation:
 - *[lower_limit . .] upper_limit*
 - The lower limit is a non-negative whole number, the upper limit is a non-negative whole number or $*$ ($*$ means „unlimited”)
 - If the lower limit is equal to the upper limit, then showing the upper limit is enough
 - e.g. 1..1 means the same as 1 and 5..5 means the same as 5
 - $0..*$ is equivalent to $*$

Property (1)

- An attribute or an association end
- Notation:
 - *[^] [visibility] [/] name [: type] [[multiplicity]]*
[= default_value] [{ modifier [, modifier] }]*
 - ^ means that the attribute is inherited (UML 2.5).
 - / means the the attribute is derived
 - If no multiplicity is present the default value is 1.
 - Modifier: e.g. `readOnly`, `ordered`, `unordered`, `unique`, ...

Property (2)

- Modifiers (not the full list):
 - **id**: it means that the attribute is a part of the identifier of the class
 - **nonunique**: means that the same values may appear multiple times
 - **ordered**: the attribute is ordered
 - **readOnly**: the attribute cannot be modified
 - **redefines** *name*: means that the attribute redefines the attribute with the given name
 - **seq** or **sequence**: means that the attribute describes an ordered multiset
 - **unordered**: means that the attribute is not ordered
 - **unique**: means that there are no multiple appearances of the same value
 - *constraint*: a constraint of the attribute.

Parameter, parameter list

- Parameter list:
 - *parameter [, parameter]**
- Parameter:
 - *[direction] name : type [[cardinality]]*
[= default] [{ property [, property] }]*
 - direction: in (default), out, inout, return
 - property: nonunique, ordered, seq/sequence, unique, unordered

Operation

- Notation:
 - *[^] [visibility] name ([parameter list])
[: type] [[cardinality]]
[{ property [, property]* }]*
 - ^ means that the operation is inherited (UML 2.5).
 - Property: nonunique, ordered, query, redefines name, seq/sequence, unique, unordered, constraint
 - **query**: means that the operation does not change the state of the system

Example class

Person
-title: String[0..1] -name: String -birthDate: Date -/age: int {age >= 0}
+Person(title: String, name: String, birthDate: Date) +Person(name: String, birthDate: Date) +getTitle(): String {query} +setTitle(title: String) +getName(): String {query} +setName(name: String) +getBirthDate(): Date {query} +setBirthDate(birthDate: Date) +getAge(): int {query}

Static attributes and operations

- Static attributes and operations are showed underlined
 - Example:

Singleton
<u>-instance: Singleton</u>
-Singleton() <u>+getInstance(): Singleton</u>

Abstract class (1)

- Notation:
 - Mark the name of the class by italic letters and/or add the annotation `{abstract}` before or after
 - UML 2.5 does not mention abstract operations

Abstract class (2)

- Example:

<i>Shape</i>
-x: int -y: int
#Shape(x: int, y: int) +getX(): int +getY(): int +moveTo(newX: int, newY: int) +getArea(): <i>double</i> +draw()

Association (1)

- Describes a semantic relation that may be present between instances of classifiers
 - It marks that there may be relations between instances of the associated types or implement them.
- They have at least two ends
 - Two-end association: binary association.

Association (2)

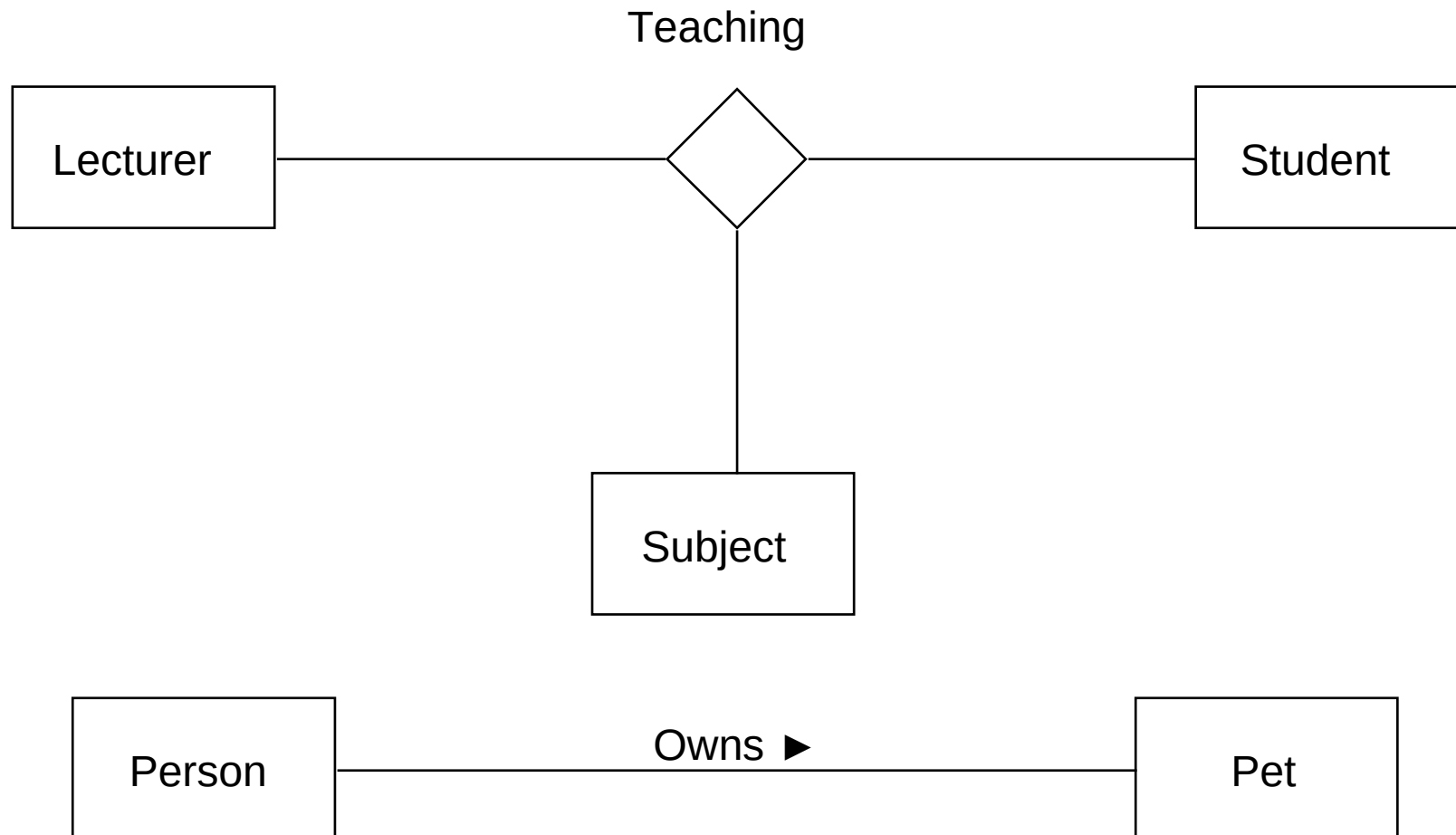
- Notation:
 - Any association can be noted by a diamond that is connected by a continuous line to the classifier that is the type of the association. A more than two end association can be noted just in this way.
 - A binary association is usually noted by a continuous line connecting two classifiers (or one)

Association (3)

- An association can be named (the name is written on the line but not too close to any ends).
 - A triangle may be added to the name of a binary association showing the reading direction of it. This is just for documentation purposes.

Association (4)

- Example:

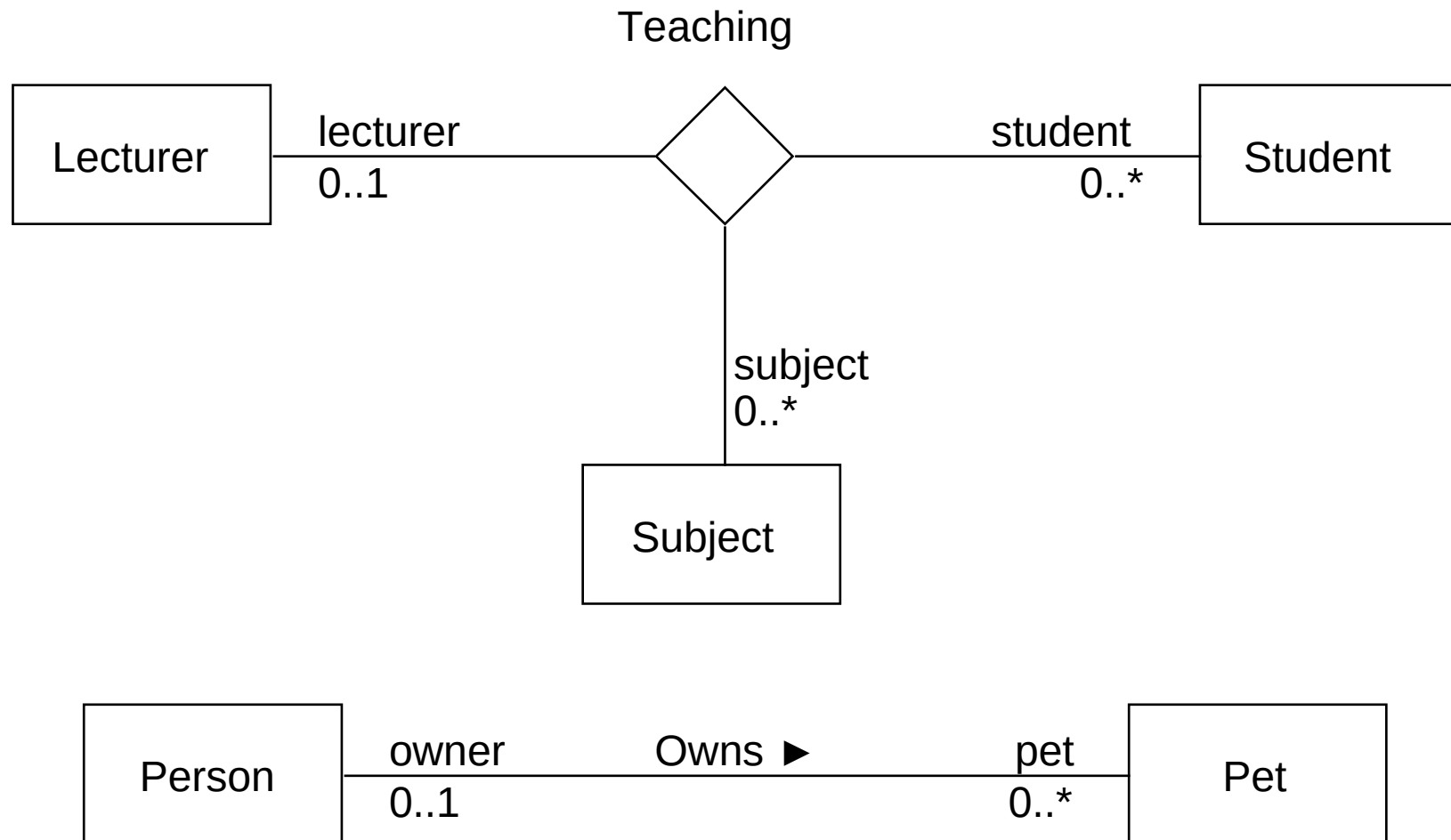


Association (5)

- **Association end:** a connecting point of the line marking the association and the icon of a classifier at the end
 - Close to the end off the line the followings may be put optionally:
 - Name (often called as „role”)
 - Multiplicity (if not given, we have no constraints about it)
 - Modifier (see properties)
 - Visibility
 - > means that the end is navigable, while × means that it is not navigable

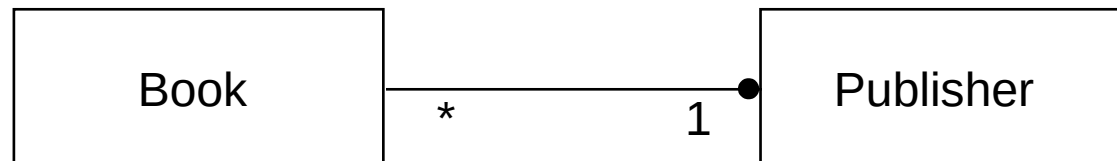
Association (6)

- Example:



Association (7)

- At the connection point of the line and the classifier a little filled circle can be placed (hereafter „dot”).
 - „dot” shows that the model has a property whose type is represented by the classifier. This property is a part of the classifier at the other end of the association. In this case usual the attribute is left out from the „attributes” part of the classifier.
 - If there is no „dot” that means that the end is a part of the association itself.

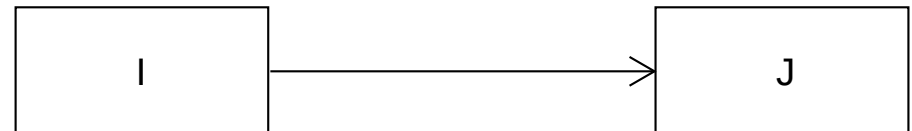
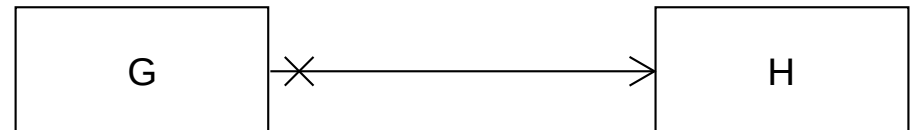
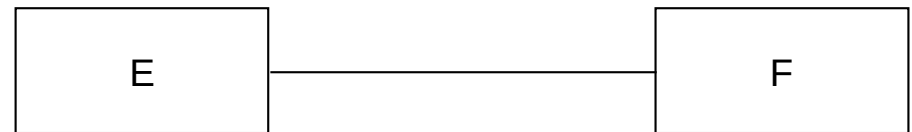
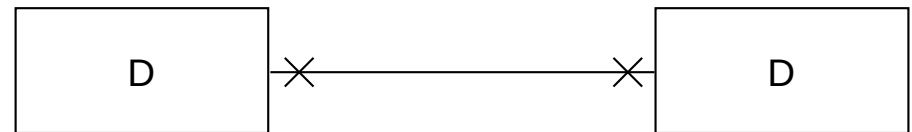
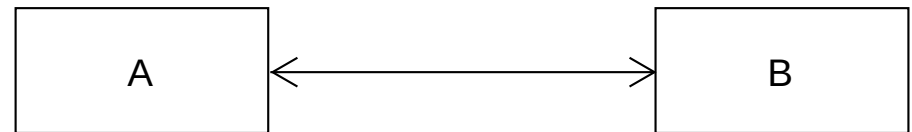


Association (8)

- Navigability means that the instances of the association can be reached in runtime from the instances at the other ends of the association
 - Association ends of classes are always navigable while association ends of associations may or may not be navigable

Association (9)

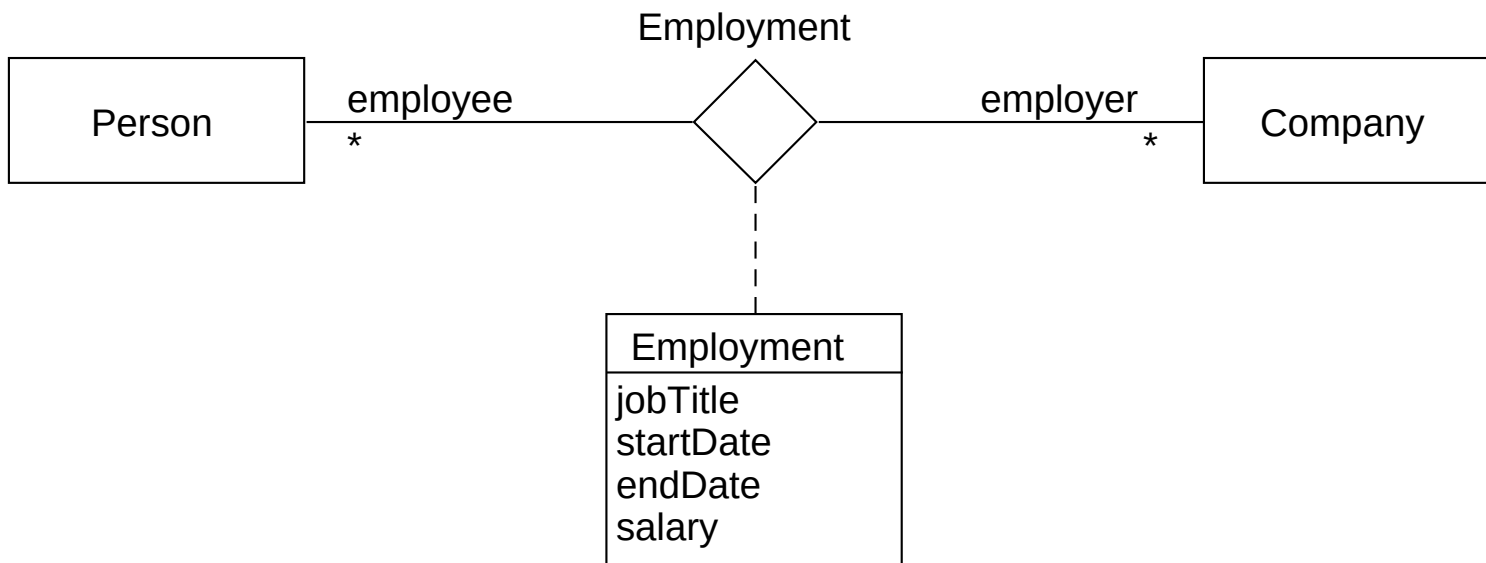
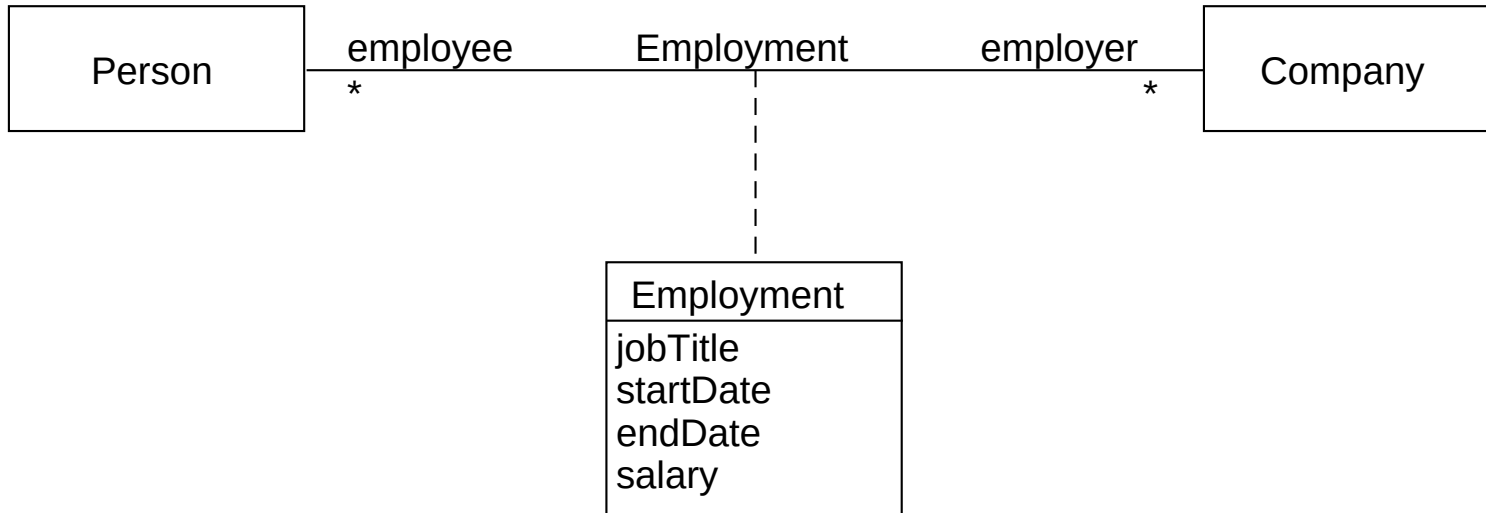
- Both ends are navigable
- No ends are navigable
- Navigability is not defined
- One end is navigable the other is not
- One end is navigable the other is not (In a diagram that does not use crosses and uses > only at one way associations)



Association class (1)

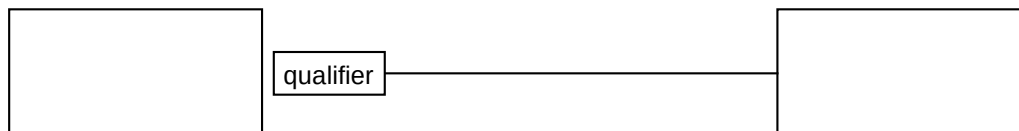
- A model element that is a class and an association at the same time. Defines attributes for the association.
- Notation:
 - A class symbol connected to an association symbol (diamond or continuous line) by a dashed line.
 - The symbol and the class marks the same model element. Their names have to match.

Association class (2)



Qualified association (1)

- The qualifier is a (set of) property(es) of an association end.
- Notation:
 - Marked by a small box at the end of the association
 - One or more qualifiers may be given. Their notation is the same as for attributes but they do not have starting values.

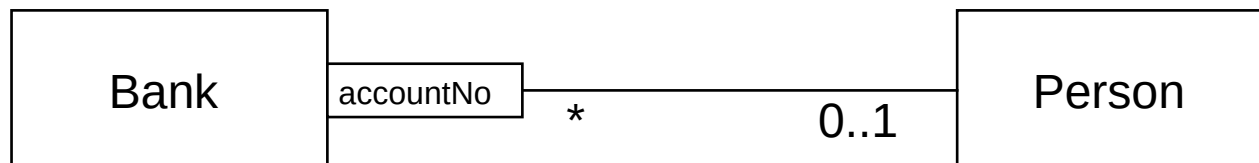


Qualified association (2)

- The qualifier splits the set of instances connected to the qualified instance to distinct partitions.
 - All partitions are marked by a qualifier value that is a vector holding a value for all qualifier attributes
 - Multiplicities at the end of the association show the number of instances in the partitions. E.g. 0..1 means that there is at most one instance per qualifier value

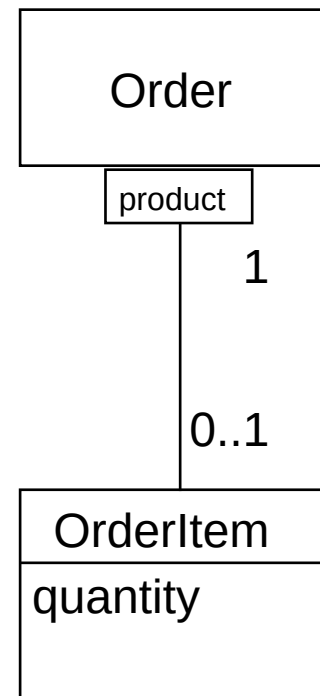
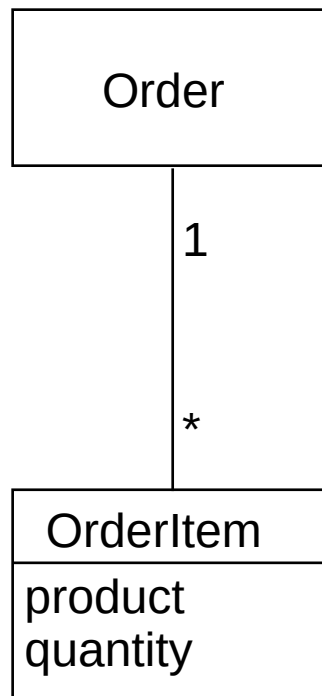
Qualified association (3)

- Example:
 - The diagram shows that for a given bank one account number can identify 0 or 1 person
 - The qualifier is the accountNo attribute, the qualified instance is the Bank. The qualifier is for the not mentioned Bank type attribute of the Person



Qualified association (4)

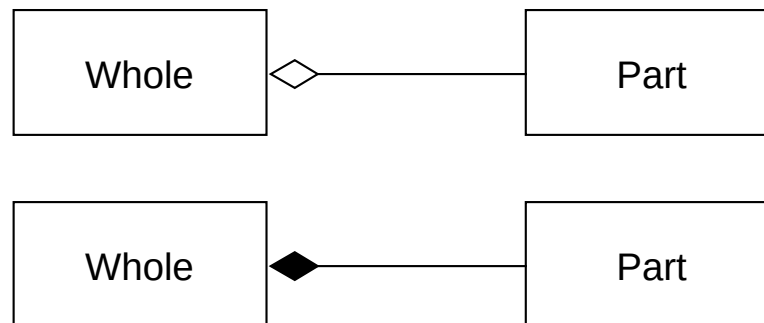
- Example:



Aggregation (1)

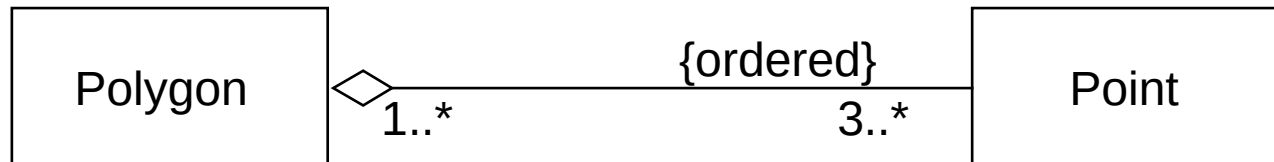
- Types of binary association:
 - **Shared aggregation**: One part object can belong to more than one aggregated objects. The parts and the aggregated object can exist without each other
 - **Composite aggregation, composition**: A more strict version of the aggregation. A part object can belong at most one composite object. At the time of deleting the composite object all the part objects are also deleted.
- Only one end of an association can be marked as aggregation or composition

- Notation:



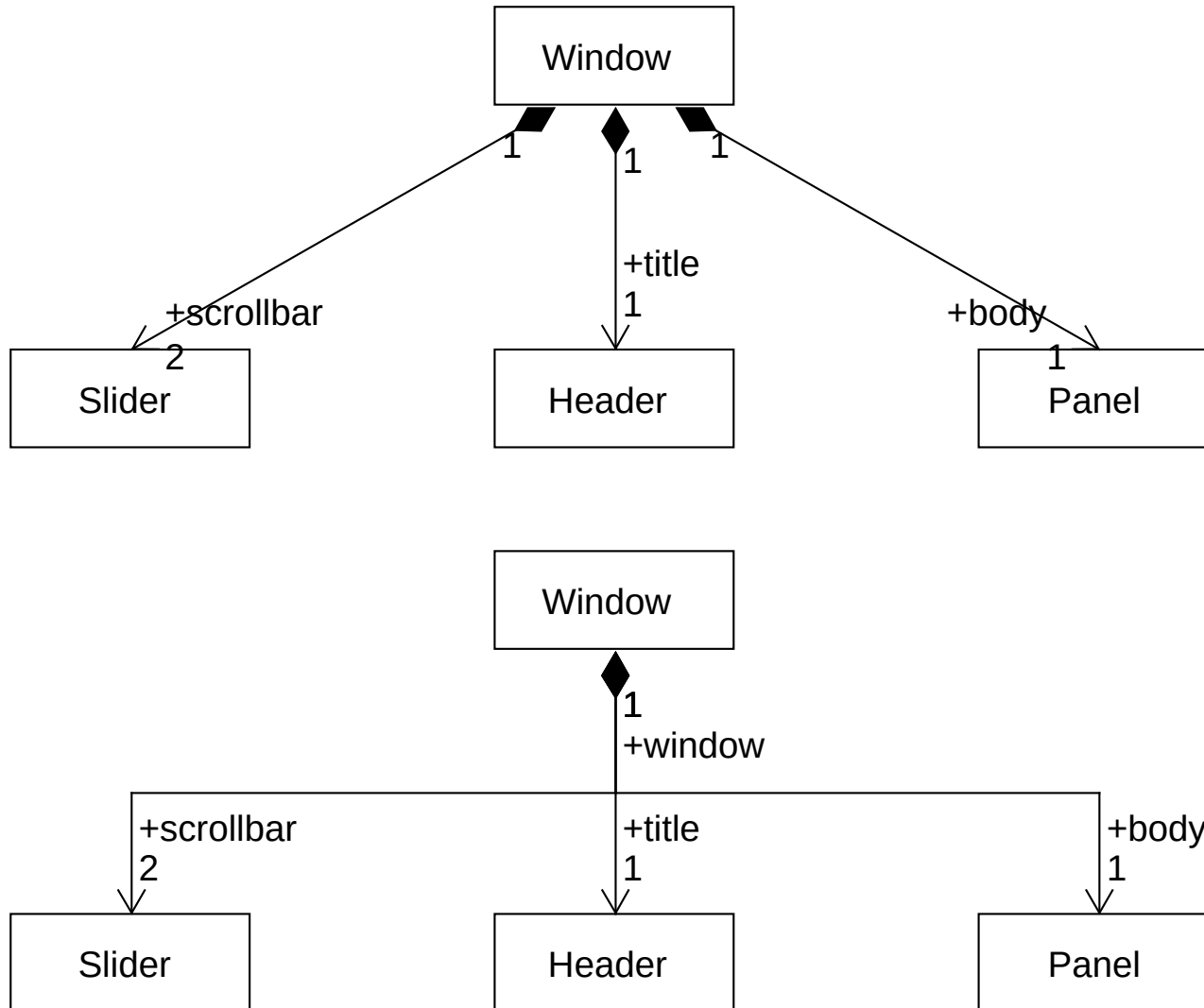
Aggregation (2)

- Example shared aggregation:



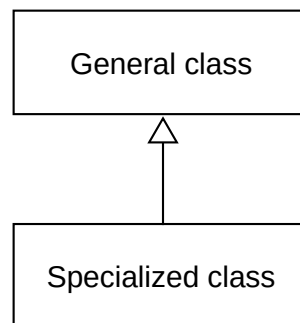
Composite connection (3)

- Example composite connection:



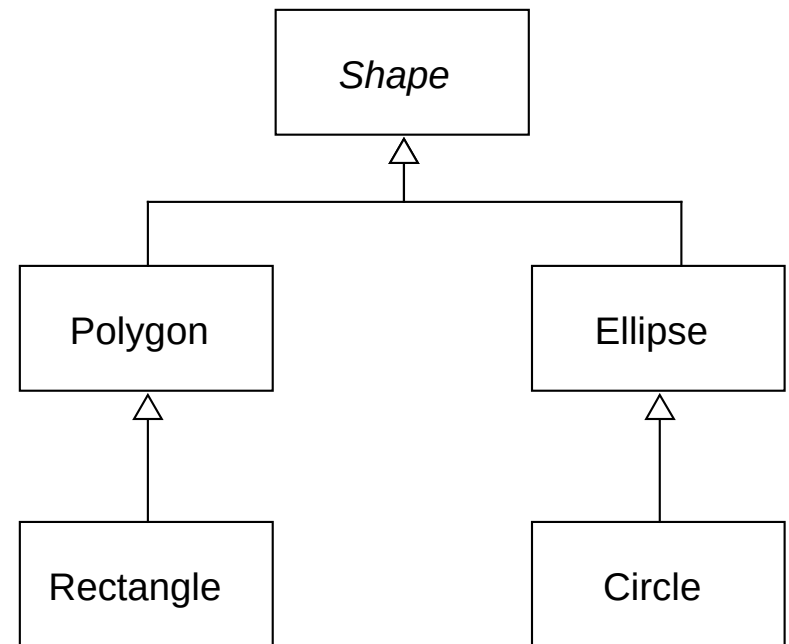
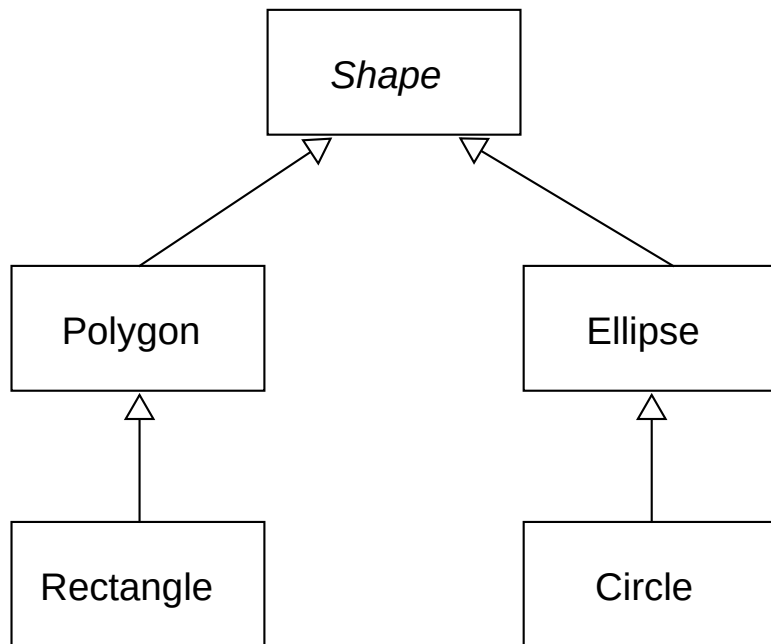
Generalization (1)

- Generalization describes a generalization/specialization connection between classifiers. It connects a more special classifier to a more general classifier.
 - The general classifier is called the parent classifier.
 - The special classifier inherits given properties of the general classifier
 - An instance of a classifier is an instance of all the generalizations of it.
- Notation:



Generalization (2)

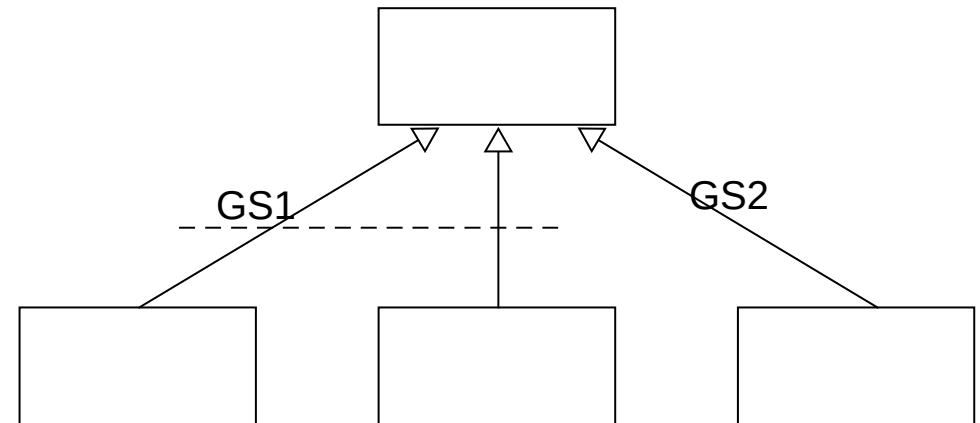
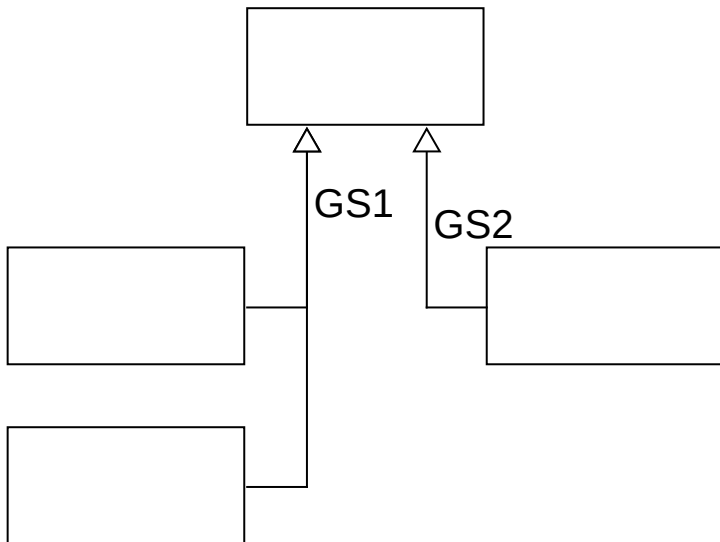
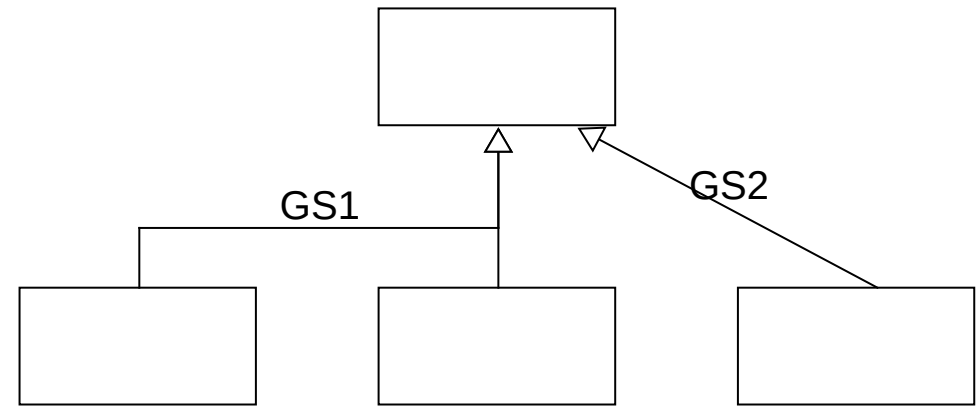
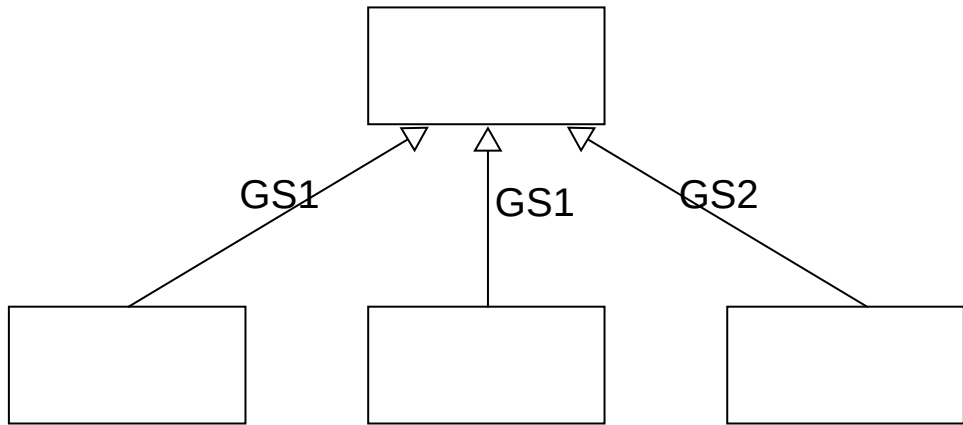
- Example:



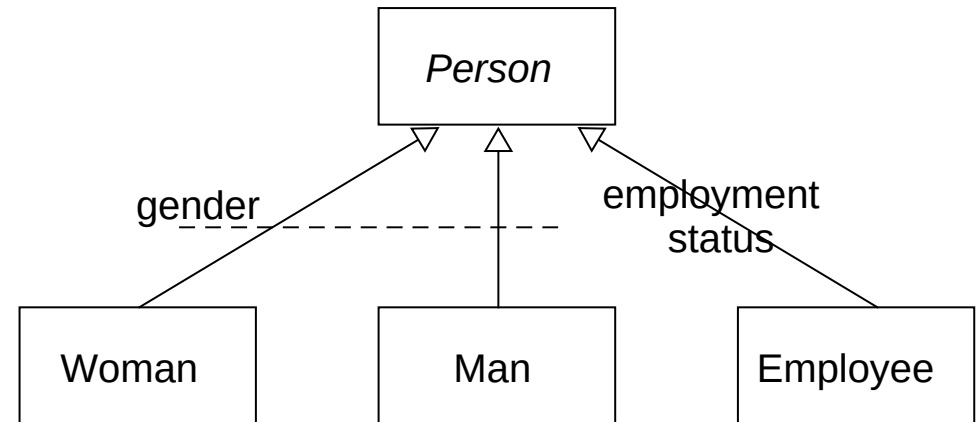
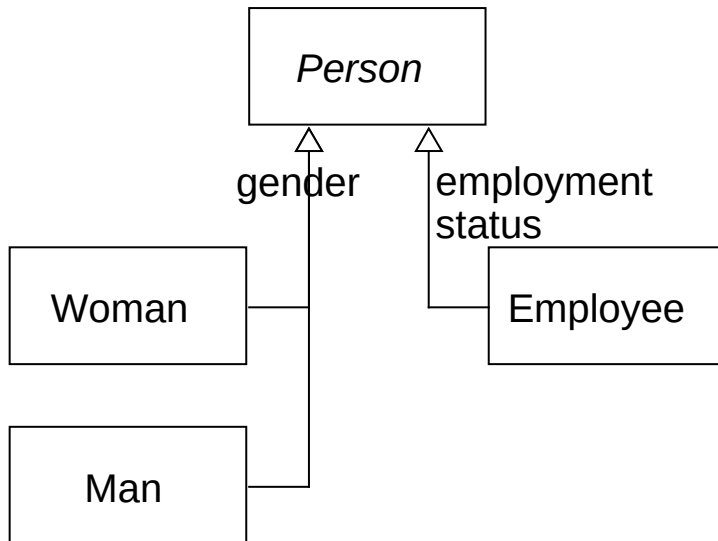
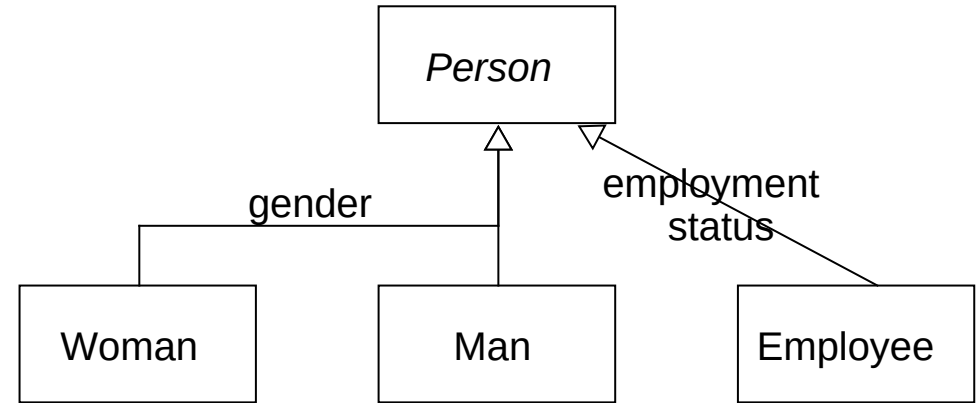
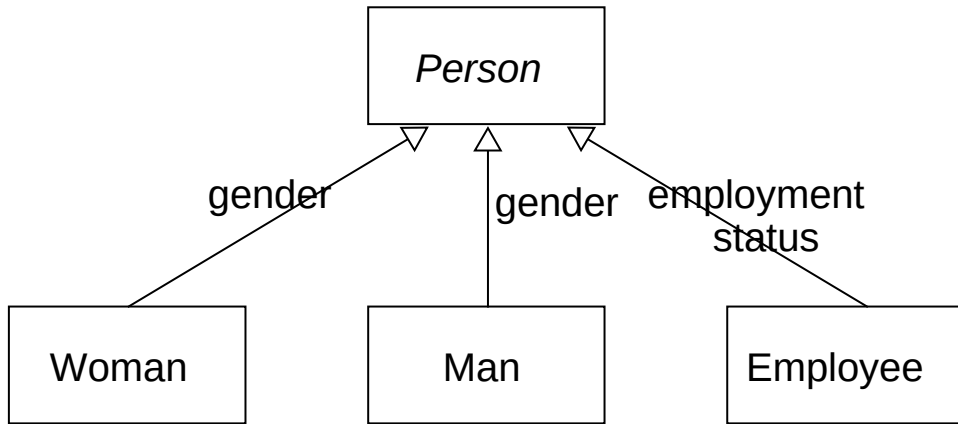
Generalization set (1)

- A tool of grouping generalizations
 - Their aim is to describe orthogonal dimensions off the generalization

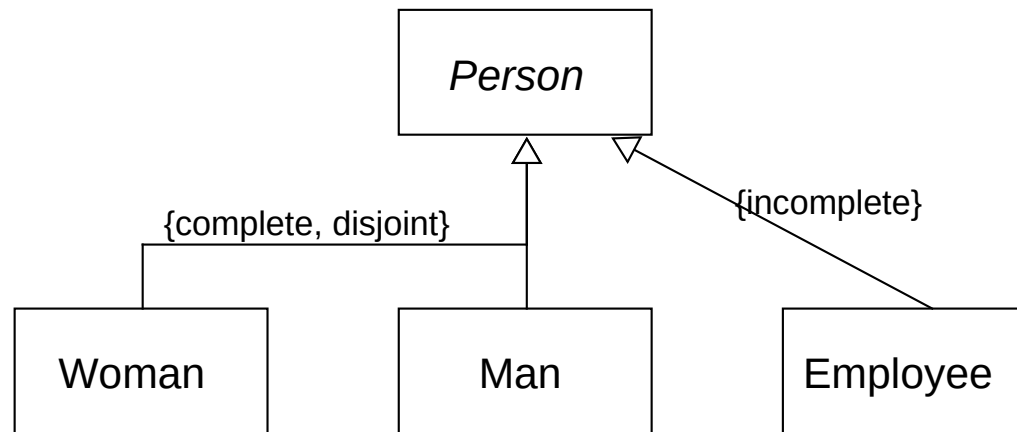
Generalization set (2)



Generalization set (3)

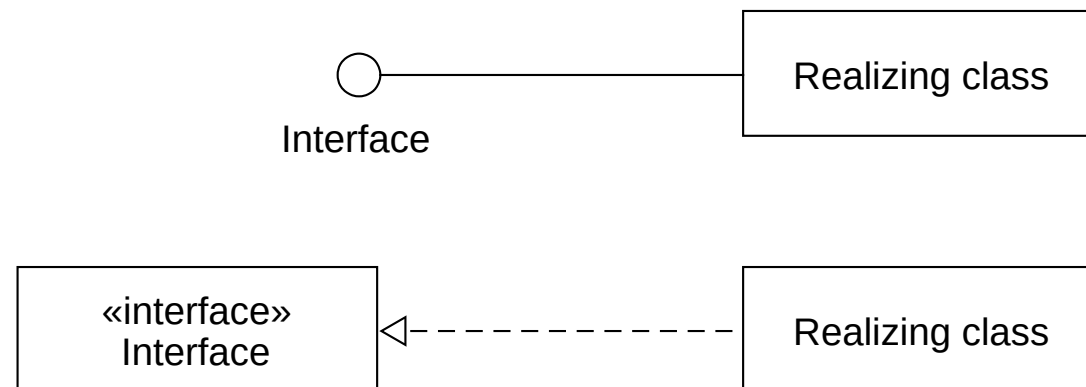


Generalization set (4)



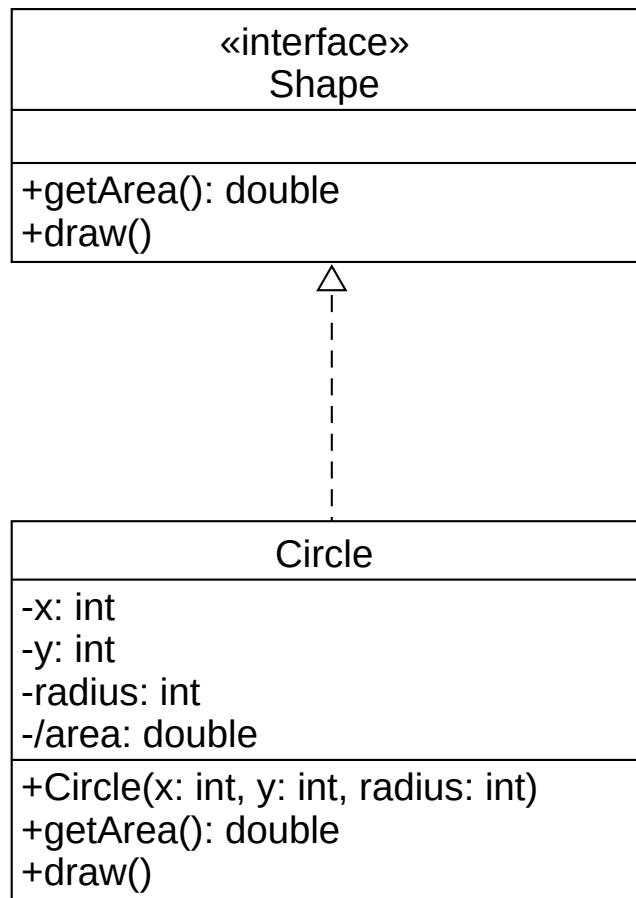
Interface (1)

- An interface is a classifier defining public properties providing a coherent service as a result. An interface can be illustrated as a contract that has to be fulfilled by all the realizing classifiers..
- Interfaces can't be instantiated. Classifiers implement or realize the specifications of the interface meaning that they provide a public connection surface matching to the specification of the interface.
- Notation:



Interface (2)

- Example:



Instance specification (1)

- Instance specifications are possible or already existing instances of classifiers in the modeled system.
- An instance specification can describe an instance at a given time-instance.
- An instance specification is a model element that is not the same as the modeled instance.
 - It may partially describe the properties of an instance. In the model there may be more than one instances fulfilling requirements of the specification.

Instance specification (2)

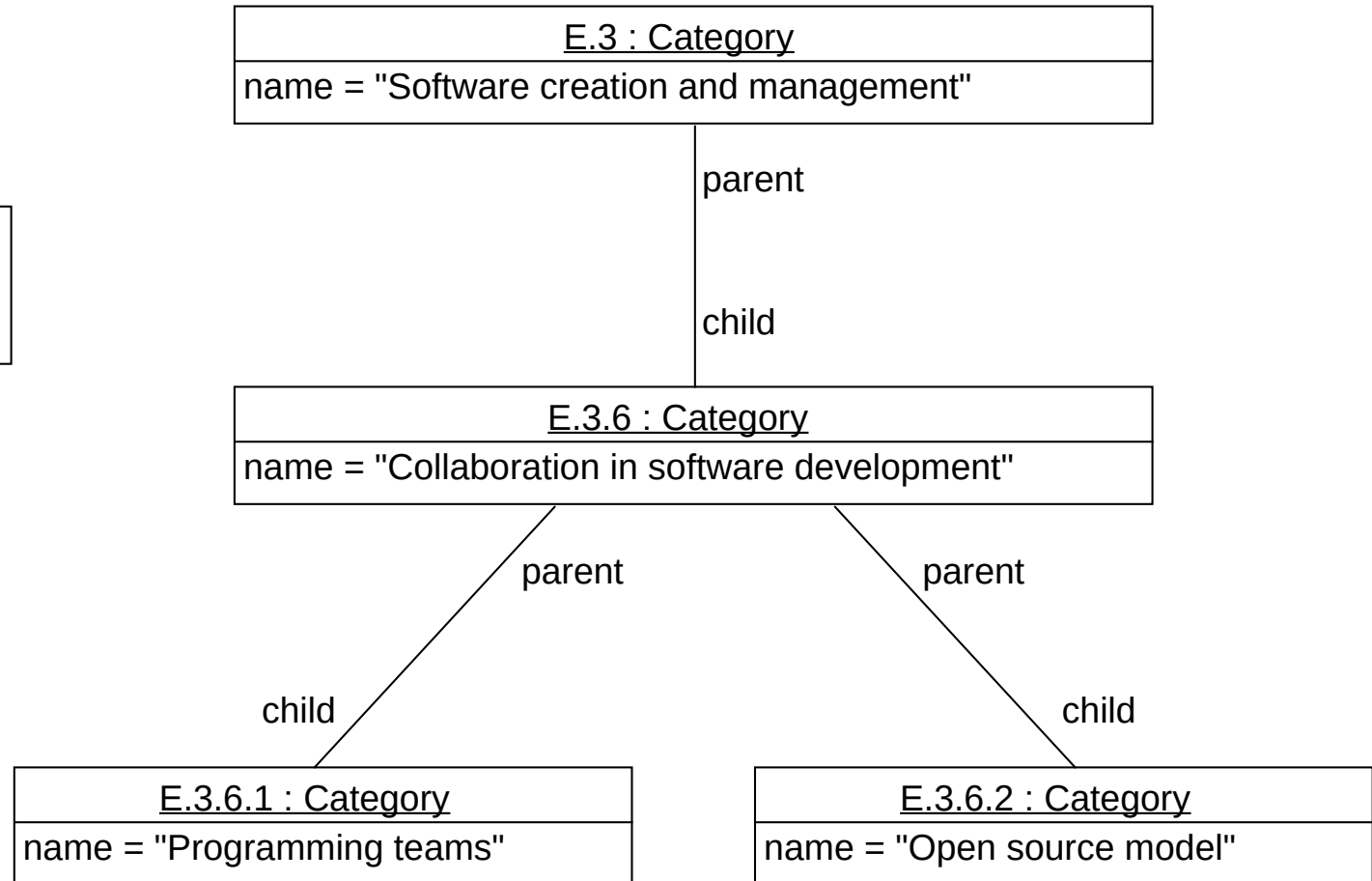
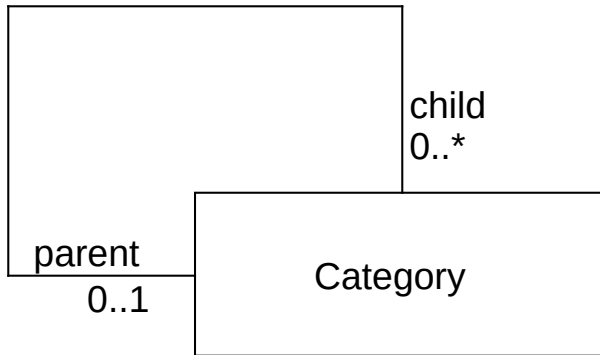
- Notation:

<u>[object_name] : [class_name]</u>

<u>[object_name] : [class_name]</u>
attribute = value
...

<u>deikAddr : Address</u>
streetName = "Kassai út" streetNumber = "26" city = "Debrecen" postalCode = 4028 country = "Hungary"

Object Diagram



Free software

- *Modelio* (operating system: Linux, Mac OS X, Windows; license: GNU GPL v3) <https://www.modelio.org/>
- *Papyrus* (platform: Eclipse; license: Eclipse Public License v1.0) <http://www.eclipse.org/papyrus/>
- *PlantUML* (platform: Java; license: GNU GPL v3) <http://plantuml.com/>
- *UMLet* (platform: Java; license: GNU GPL v3) <http://www.umlet.com/>
 - Eclipse plugin: <http://marketplace.eclipse.org/content/umlet-uml-tool-fast-uml-diagrams>
 - Web service: *UMLetino* <http://www.umlet.com/umletino/>

Closed software

- *Altova UModel* (OS: Windows)
<https://www.altova.com/umodel.html>
- *Microsoft Visio* (OS: Windows) <http://visio.microsoft.com/>
- *Sparx Systems Enterprise Architect* (OS: Windows)
<http://www.sparxsystems.com/>
- *StarUML* (OS: Linux, Mac OS X, Windows) <http://staruml.io/>
- *Visual Paradigm* (OS: Linux, Mac OS X, Windows)
<https://www.visual-paradigm.com/>
 - *Visual Paradigm Community Edition*
<https://www.visual-paradigm.com/download/community.jsp>

PlantUML (1)

- Free and open source software for the creation of UML diagrams
<http://plantuml.com/>
 - Programming language: Java
 - License: GNU GPL v3
- Supported diagrams: class diagram, object-diagram, sequence-diagram, use-case diagram, ...
- Provides a simple language to define diagrams
- Some diagrams are provided by using:
 - Graphviz (platform: Linux, Mac OS X, Windows; programming language: C; license: Eclipse Public License 1.0) <http://www.graphviz.org/>
- Used and integrated by a lot of development tools.
 - See: *Tools using the PlantUML language* <http://plantuml.com/running>

PlantUML (2)

- Support:
 - Atom (platform: Linux, OS X, Windows; license: MIT License) <https://atom.io/>
 - *PlantUML Viewer*
<https://atom.io/packages/plantuml-viewer>
 - *PlantUML language package*
<https://atom.io/packages/language-plantuml>
 - PlantText <https://www.planttext.com/>

PlantUML (3)

- Support:
 - **Eclipse:** *Integration with Eclipse*
<http://plantuml.sourceforge.net/updatesite/>
 - **NetBeans:** *PlantUML-NB – Netbeans Plugin for PlantUML*
<http://plugins.netbeans.org/plugin/49069/plantuml>
<https://sourceforge.net/projects/plantumlnb/>
 - **Apache Maven:** *UML Reverse Mapper*
<https://github.com/markusmo3/uml-reverse-mapper>

Further reading

- Martin Fowler. *UML Distilled – A Brief Guide to the Standard Object Modeling Language*. Third Edition. Addison-Wesley, 2003.
- Robert A. Maksimchuk, Eric J. Naiburg. *UML for Mere Mortals*. Addison-Wesley, 2004.
- Russ Miles, Kim Hamilton. *Learning UML 2.0*. O'Reilly Media, 2006.