UML – Structural Modelling

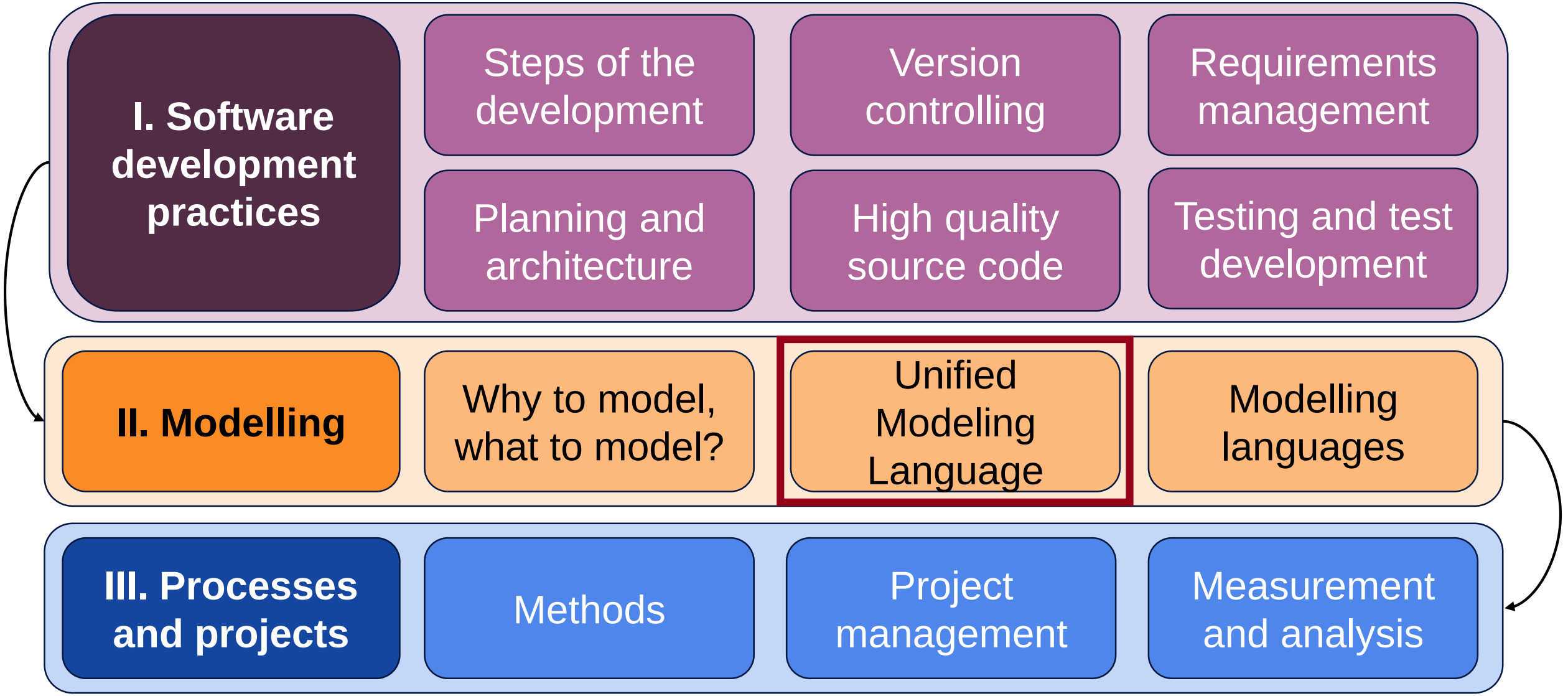
HUSZERL Gábor
huszerl@mit.bme.hu

MŰEGYETEM 1782
BME VILLAMOSMÉRNÖKI ÉS INFORMATIKAI KAR
Méréstechnika és Információs Rendszerek Tanszék
ftsrg Critical Systems Research Group

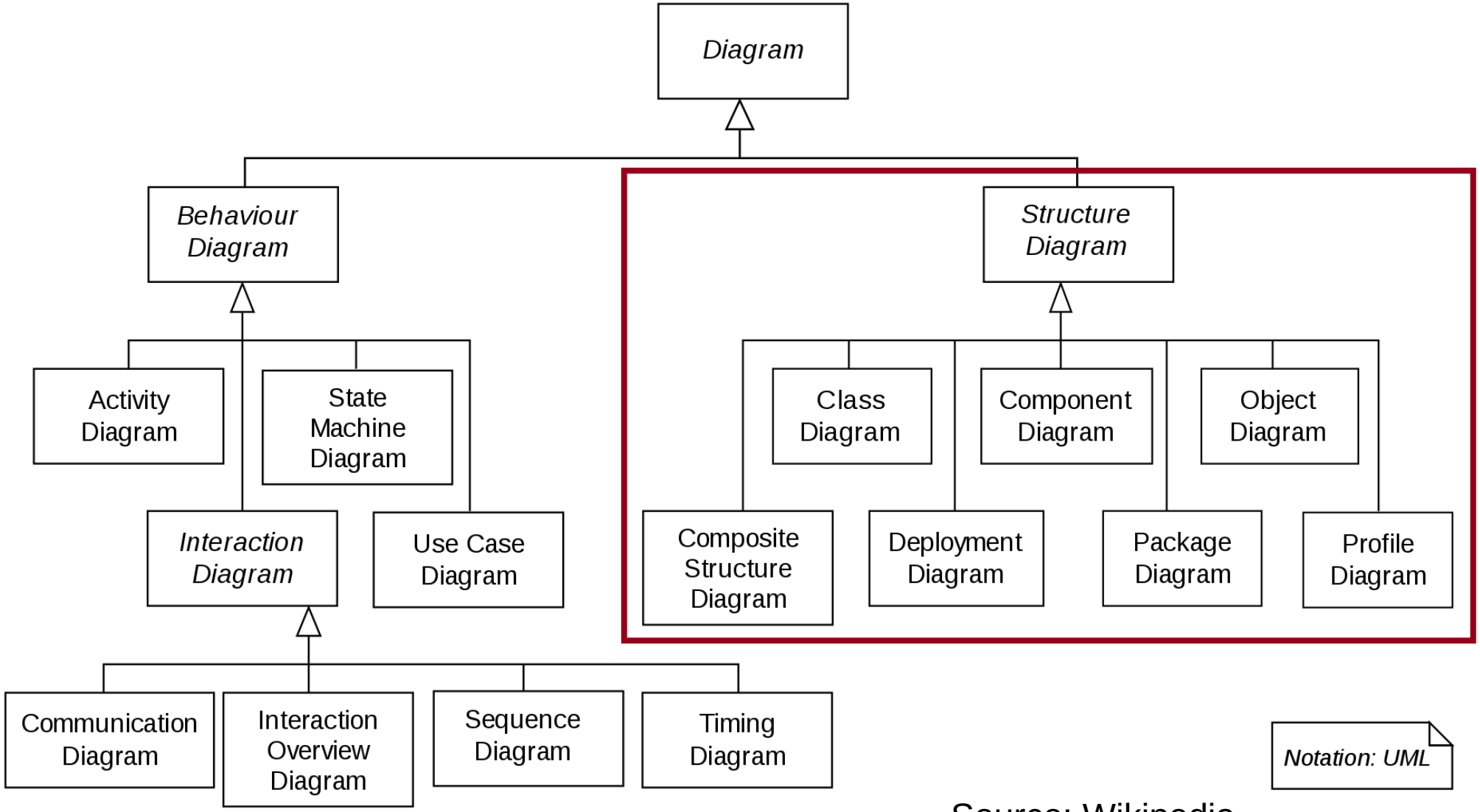
Learning Outcomes

- At the end of the lecture the students are expected to be able to
- (K1) recall the objectives of structural modelling,
- (K3) use the UML language to model components,
- (K3) use the UML language for conceptual and OO modelling.

Further Topics of the Subject



UML Diagram Types



Source: Wikipedia

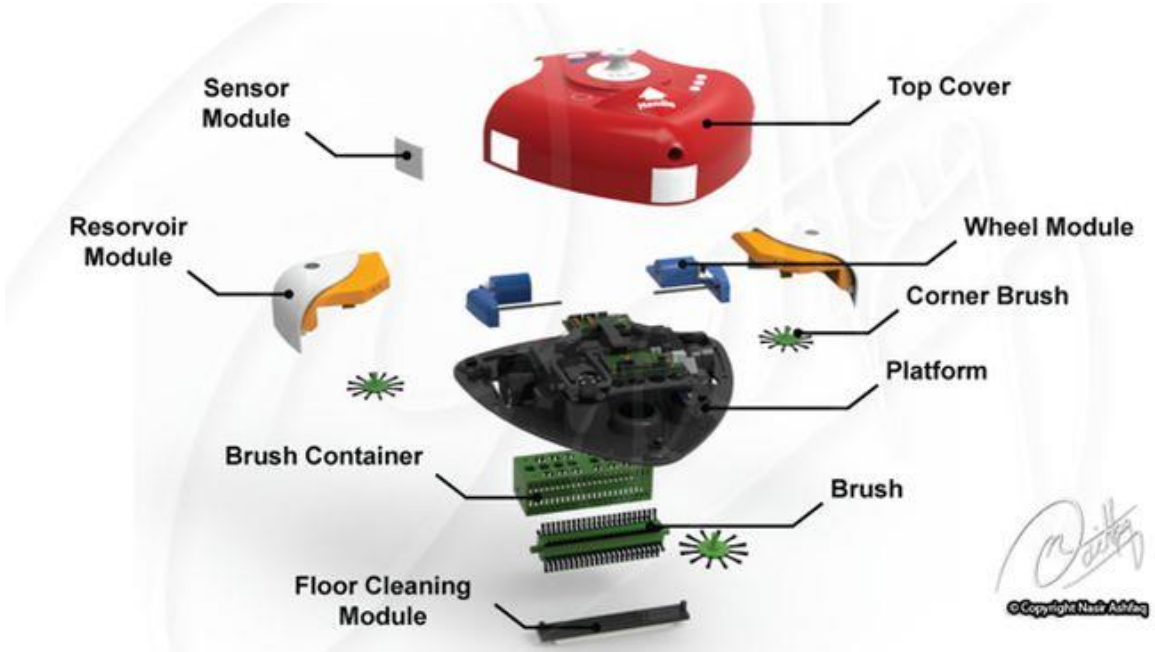
Goals of Structural Modelling

- Division of a system into its components („Decomposition”)
  - Smaller units are easier to design
  - Use of existing subsystems
  - Use of general purpose components
- Documenting existing systems
  - „System map”
- Designing data structures
  - What information are we working with?

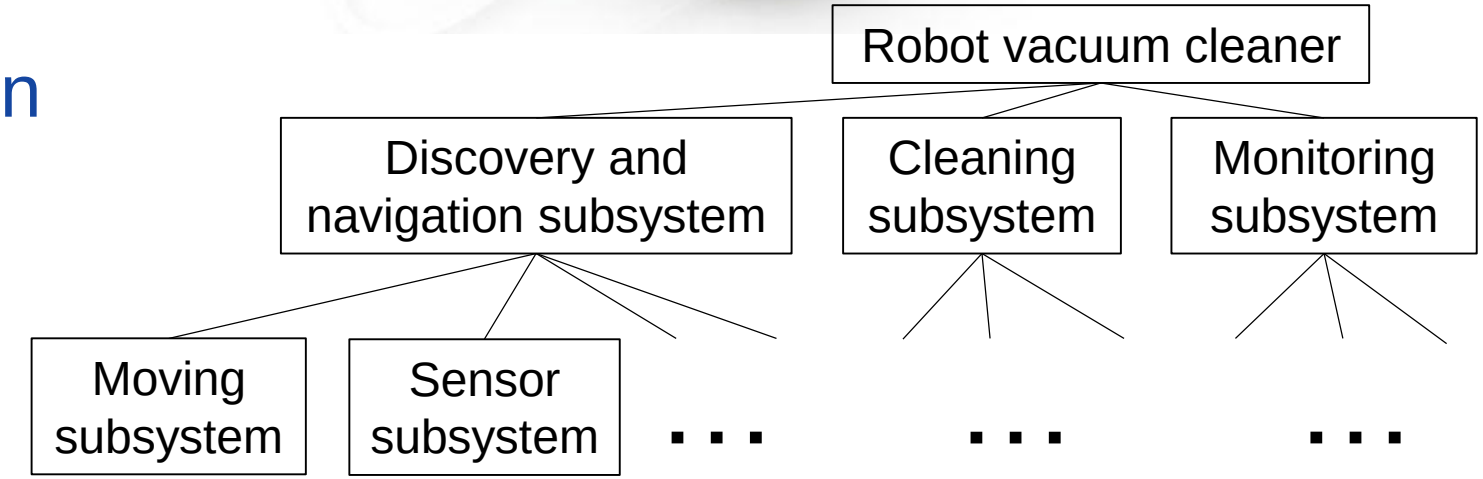
Types of Decomposition

- Physical Decomposition
  - What pieces does it consists of?

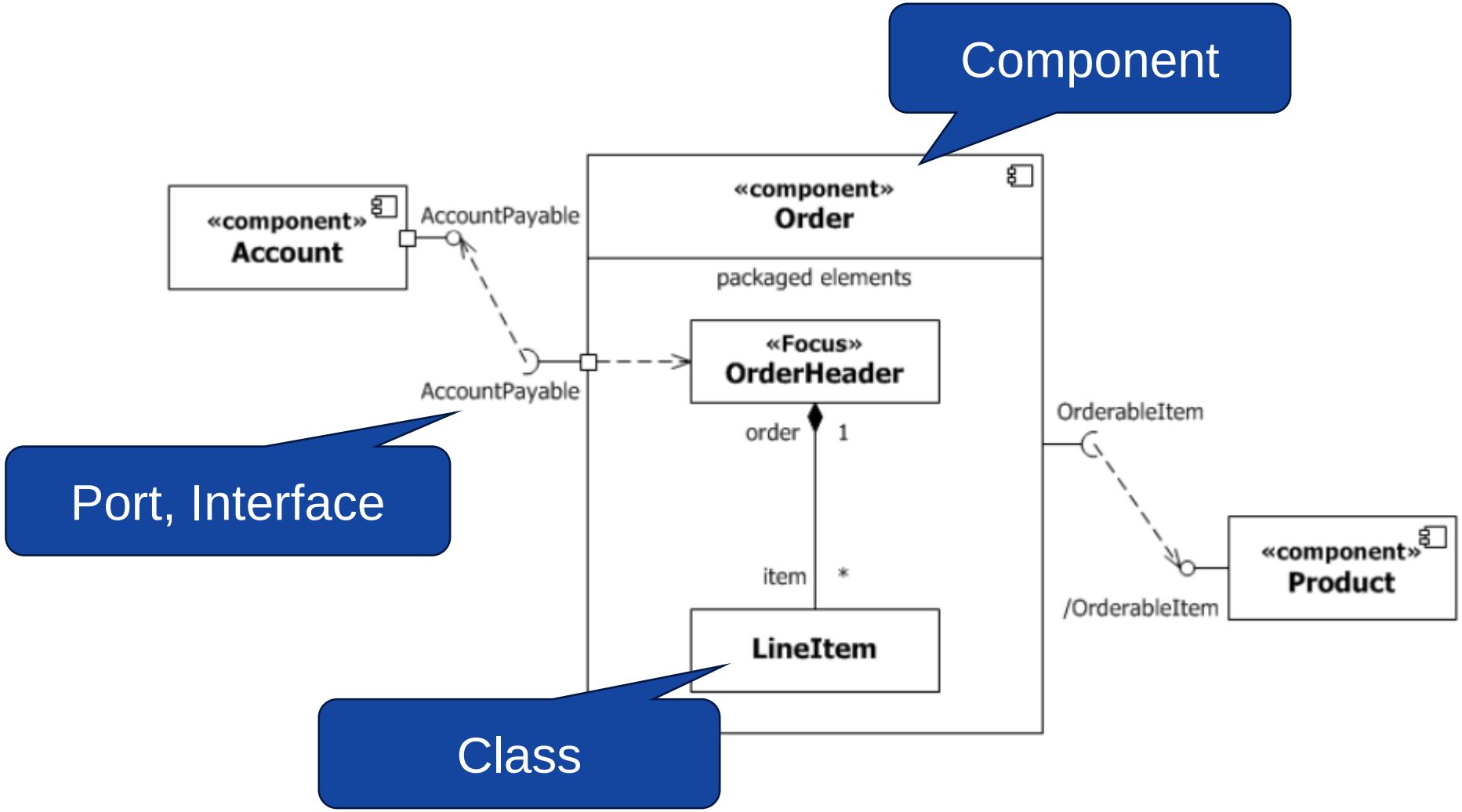
If possible,
do not mix the two!



- Logical Decomposition by *Functionalities*
  - What does it do?



UML Structural Models

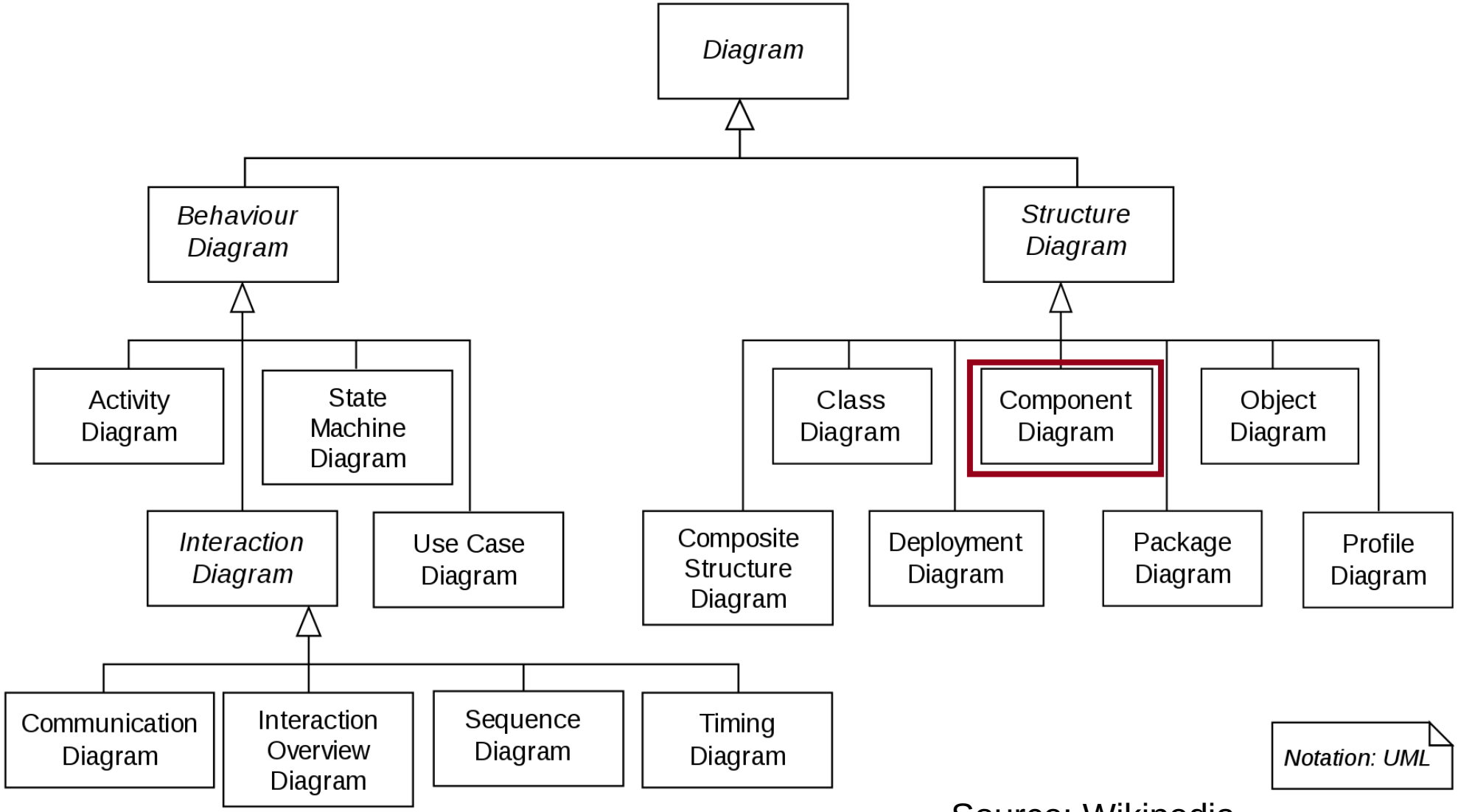


Source: UML v2.5.1

UML Component Models

Component, Port, Interface

UML Diagram Types



Source: Wikipedia

What Is a Component?

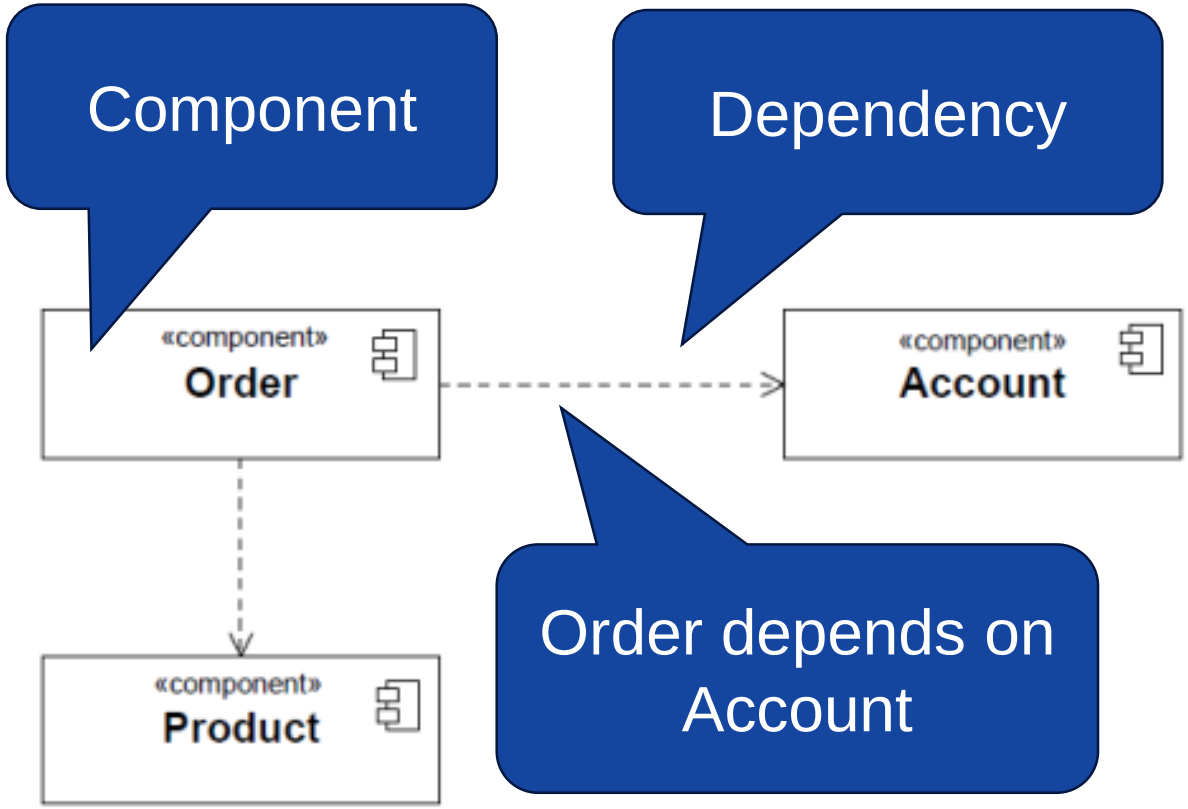
Mechanical, SW, HW, …

"A Component represents a modular part of a system that encapsulates its contents and whose manifestation is replaceable within its environment."

"A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties."

Source: C. Szyperski. Component Software - Beyond Object-Oriented Programming, 1999

Components and Their Relations



- Dependency: change of the supplier may influence the behaviour of the client
- The exact nature of the dependence and its effect is not (yet) specified

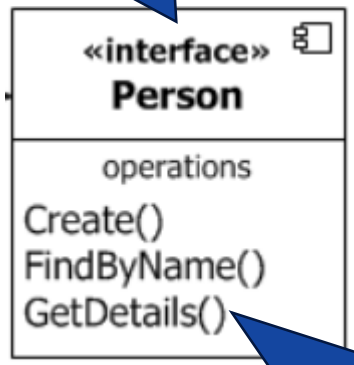
Definition of the Component by Its Interfaces

Component

provided interface [lollipop]

required interface [socket]

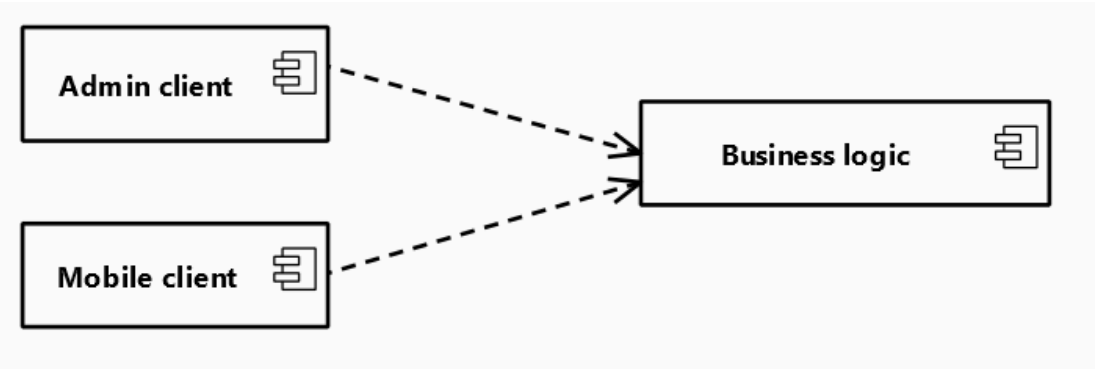
Interface: public, coherent "contract"

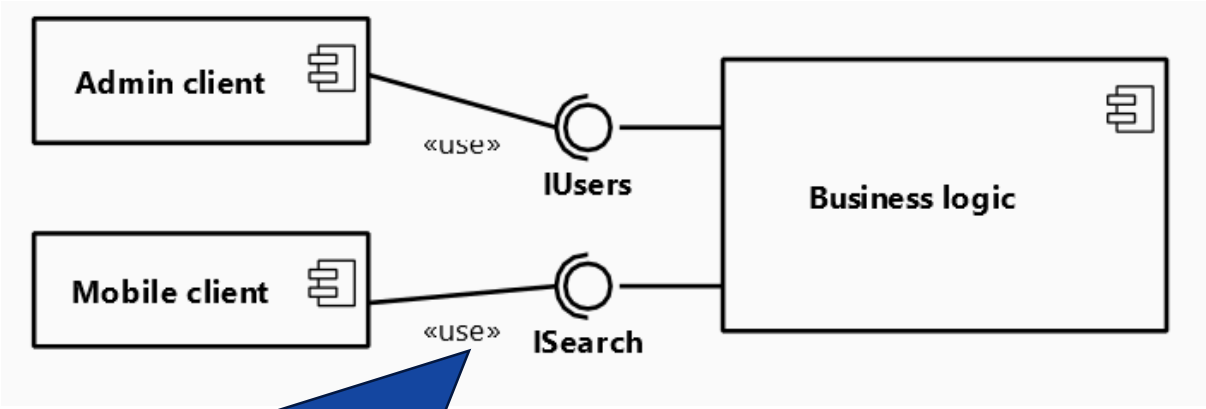


operations, properties, behavioural patterns, …

Reusability and loose coupling!

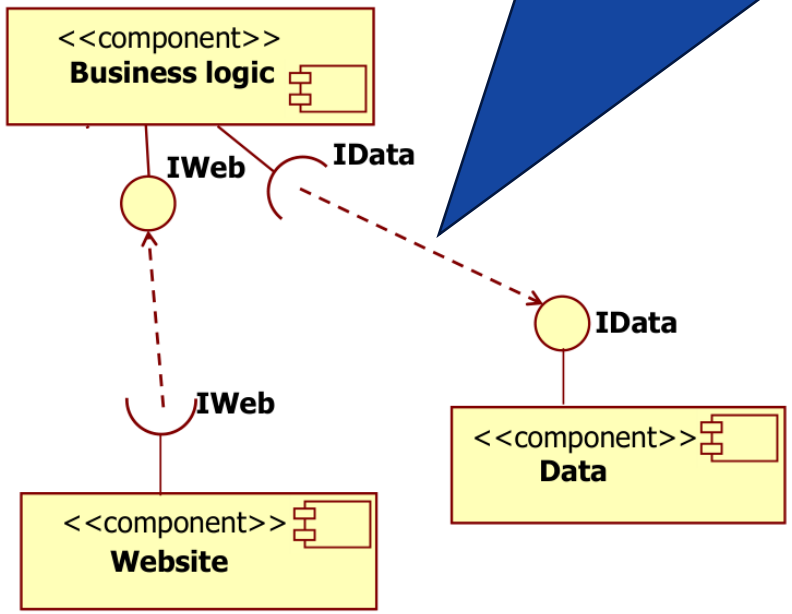
Connecting Components through Interfaces



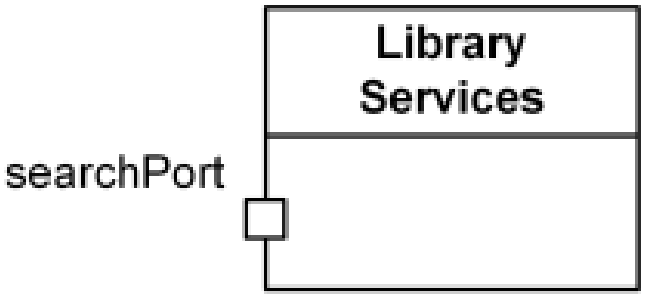


Clarification of the dependency relationship

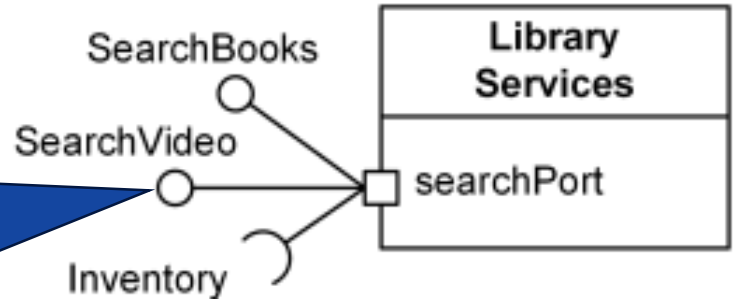
Use of dependencies: If additional information is also important when using the interface

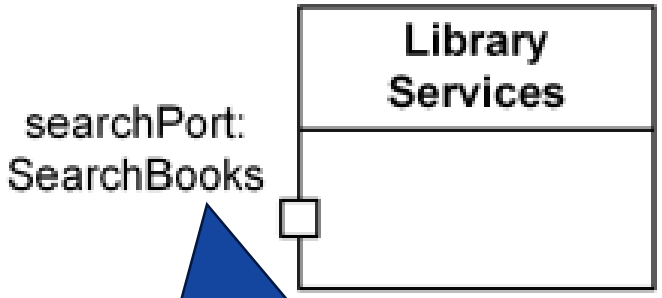


Port: Named Interaction Point

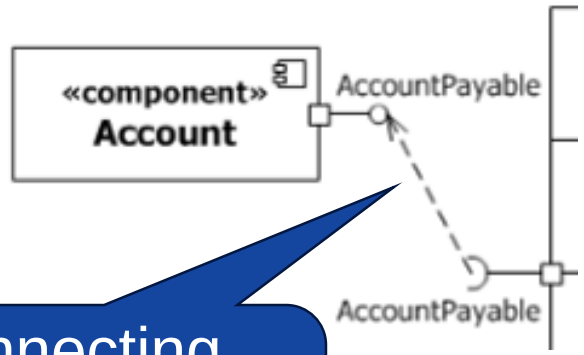


Port with interfaces (composite port: has several interfaces)

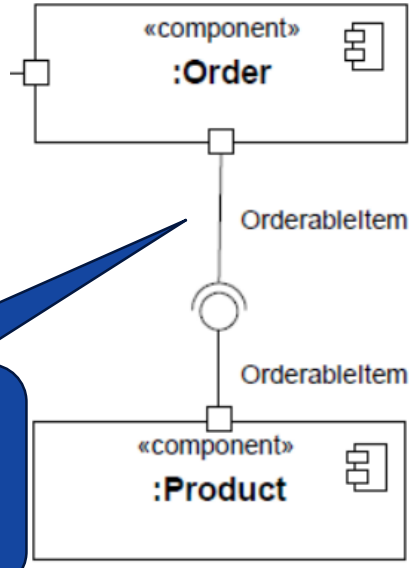




Port with type

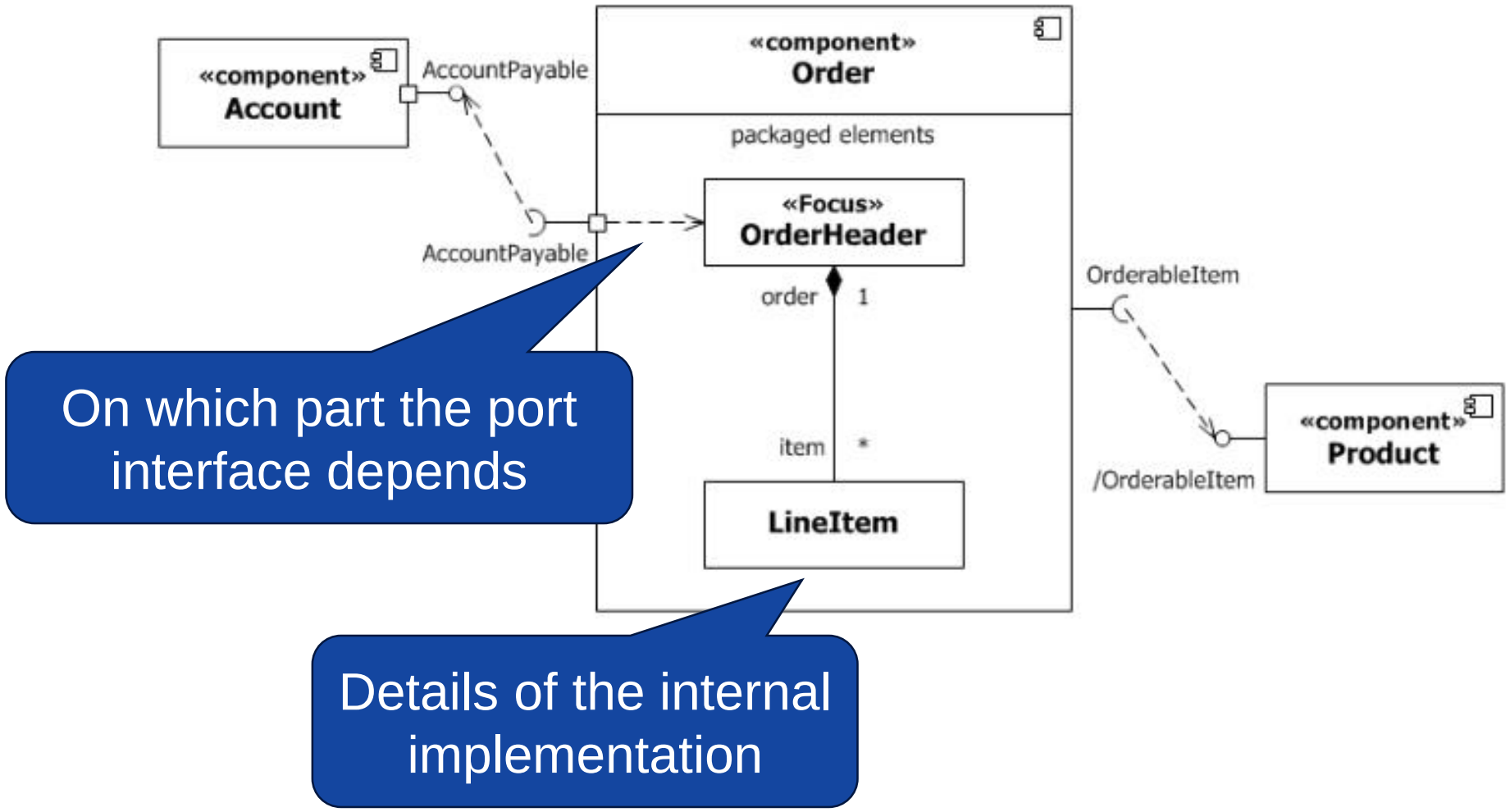


Connecting ports with dependencies

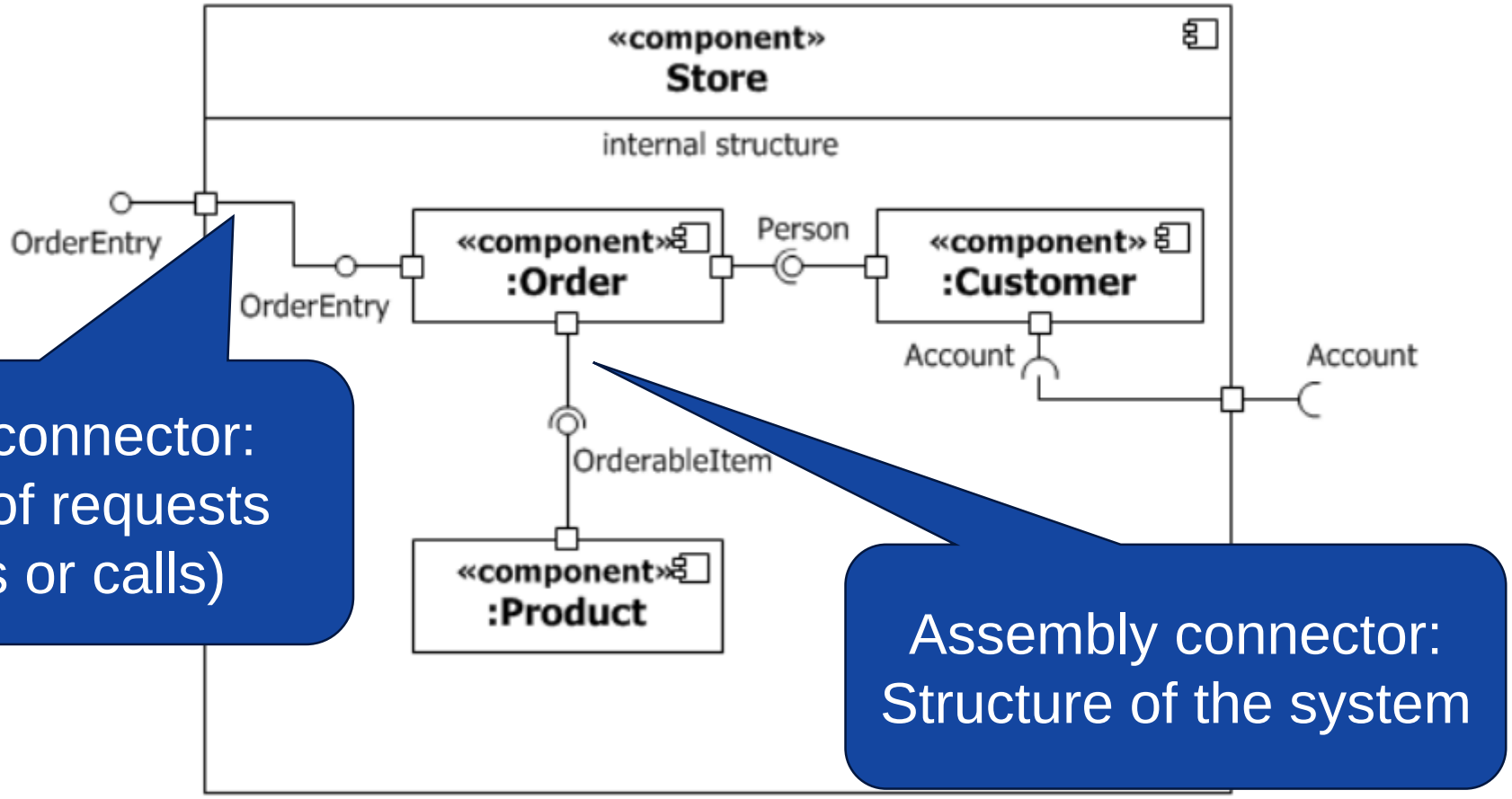


Connecting ports with relations

Representation of the Internal Structure



Internal Structure with Components

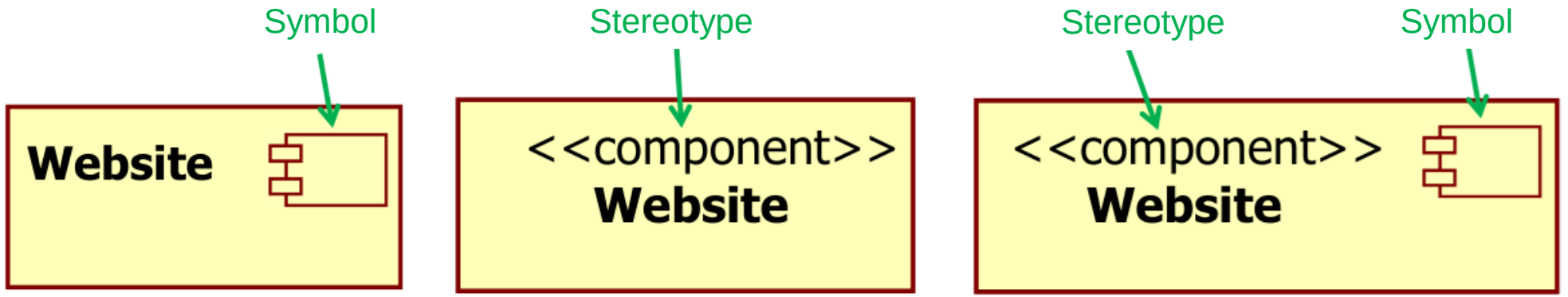


Delegation connector:
Forwarding of requests
(messages or calls)

Assembly connector:
Structure of the system

Alternative and Further Notations (1)

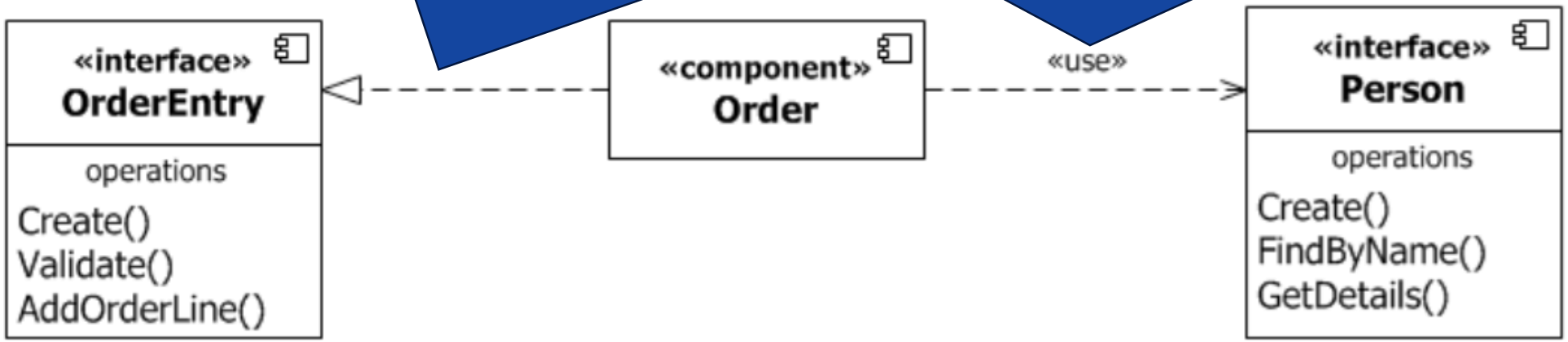
Component:



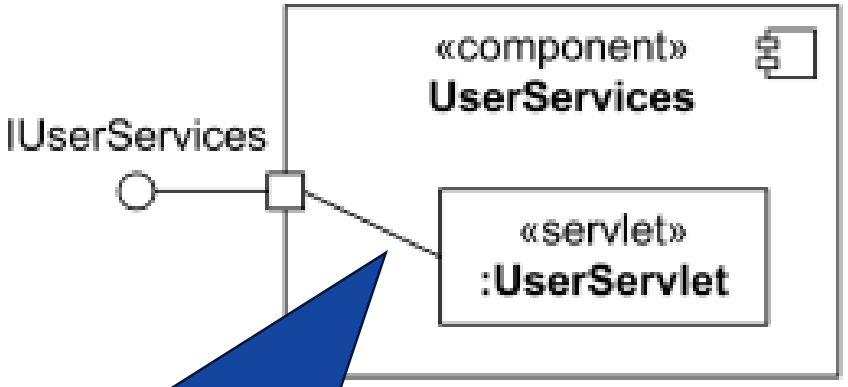
Inter-face:

InterfaceRealization = provided interface, „lollipop”. Component implements this interface

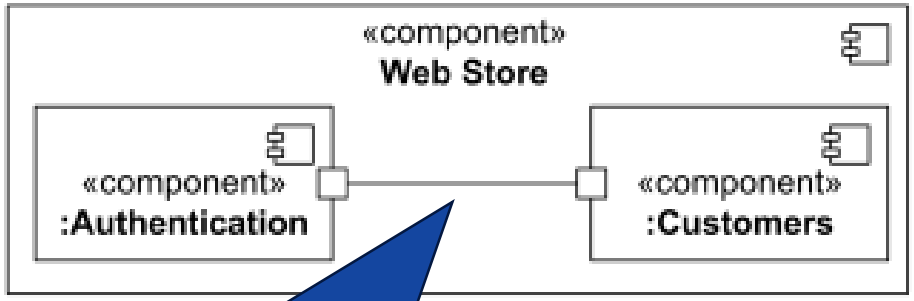
Usage Dependency = required interface, “socket“. Component uses this interface



Alternative and Further Notations (2)

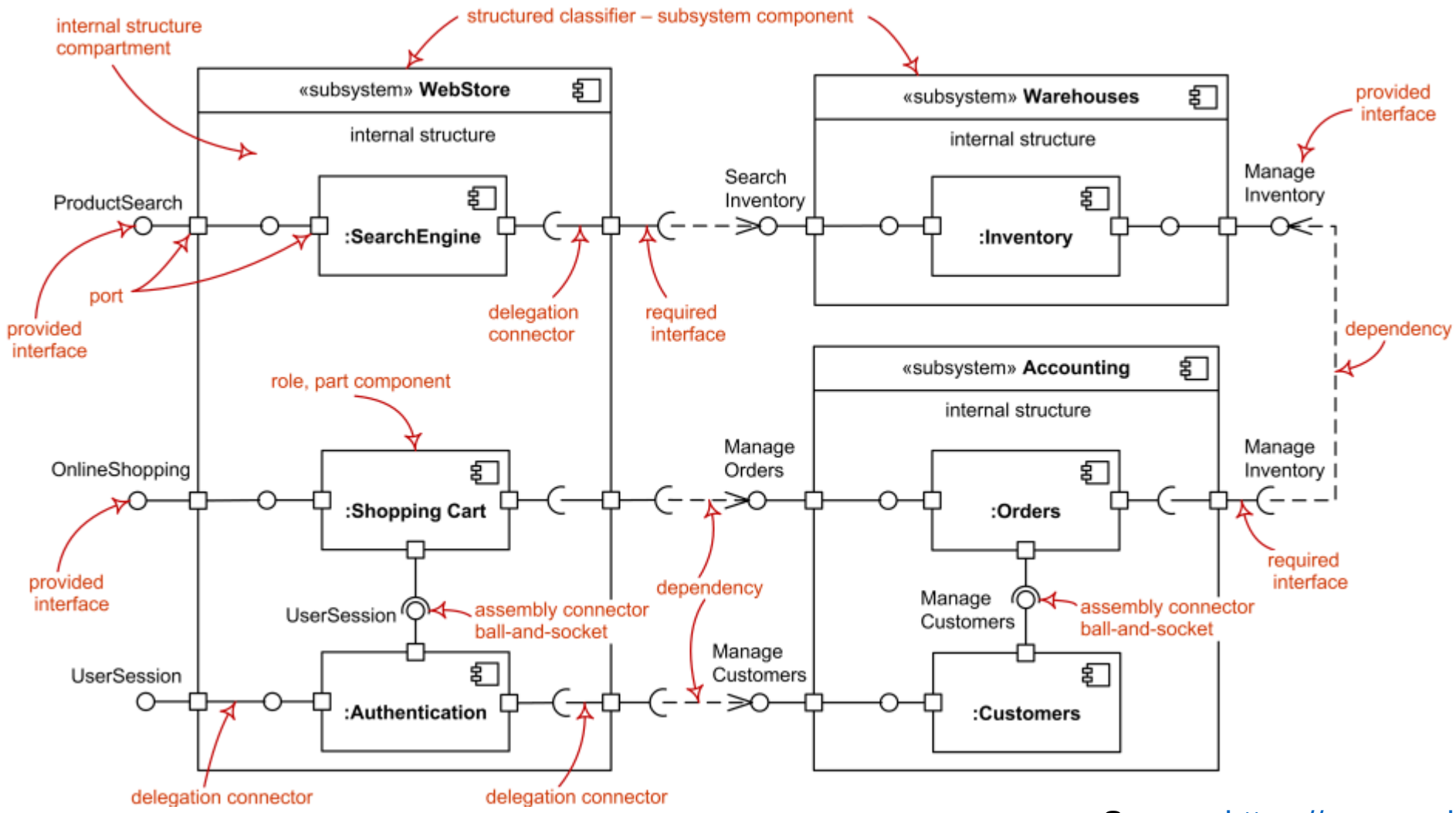


Delegation connector,
if the port is a composite one
and not simple one



Assembly connector,
if the port is a composite one and
not simple one

Port – Summary

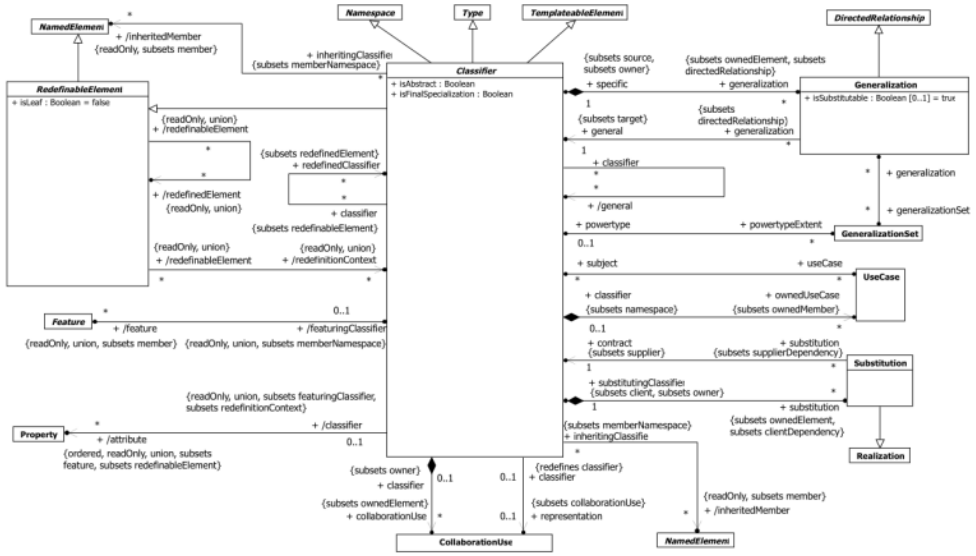


Source: https://www.uml-diagrams.org/

Exercise: High Level Architecture (MTMT)

- "The system can be divided into three main parts: a central server, which ... consists of components that implement the central functionalities, an MTMT2 system that contains not only local but also institutional ... modules, and clients and extensions that are external to the system but closely linked to it."
- Modules of the MTMT2 server:
  - **Add, change, delete publications**. The module ensures that users can successfully enter publications into the system and that users can make changes to these records.
  - **Import and export data in the appropriate format**. It helps standardise data so that importing and exporting records between different systems is not a problem.
- Modules of the MTMT2 system:
  - **Import**. With the import module you can automatically import publication data from large bibliographic databases (WoS or Scopus) via standard interfaces.
  - **Data cleaning**. With the data cleaning component, only the data corresponding to the MTMT2 database is entered into the system.
- External elements:
  - **„Web of Science, Scopus, ...** Databases with Hungarian and international bibliographic data from which data can be imported.”

UML Class Models



Class, Interface, Attribute, Method, …

Classes in Different Languages

- Classes in OO programming languages
  - C++, Java, C#, …
  - Object: A unit for the common storage of the data and the methods for its processing
  - Class: A template definition of the methods and variables in a particular type of objects
- Classes in modelling languages
  - OMT, UML, …
  - Class: A group of model elements that have similar structural characteristics (attributes, associations)
- There are similarities, indeed

Simple & Structured Classifiers

- “Class diagram”

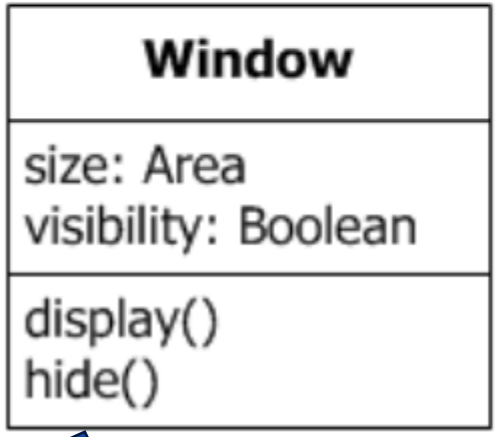
- The most frequently used UML element
  - In general: Classifiers classify instances according to their characteristics

- Concepts (+their properties, operations) and the connections among them

- Can be used at different levels of abstraction
  - Conceptual models, analysis, detailed implementation, ...

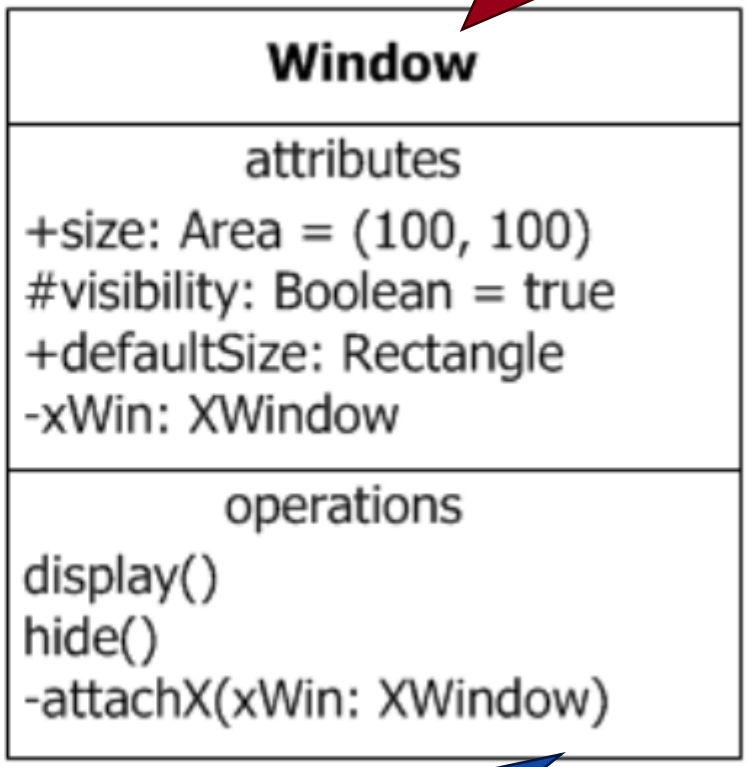
Various Degrees of Detail

Concepts



Early analysis

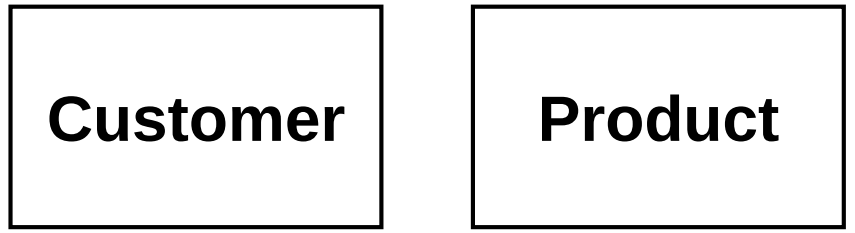
This is often not the level at which you model!



Detailed implementation

Class

- „*A concept from the modeled system*”

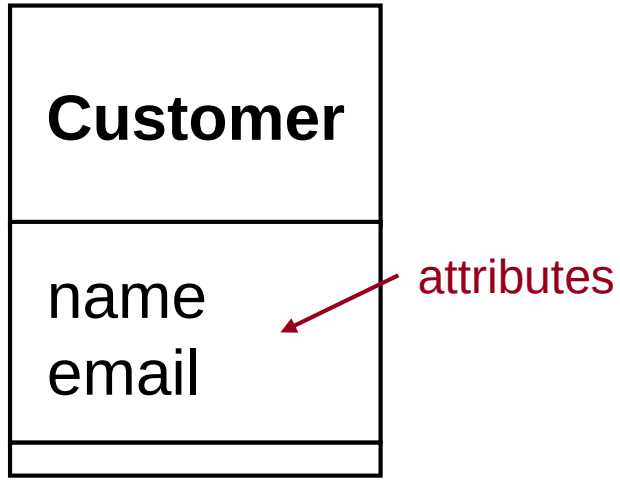


- Common state, behaviour and constraint of Things (~objects)

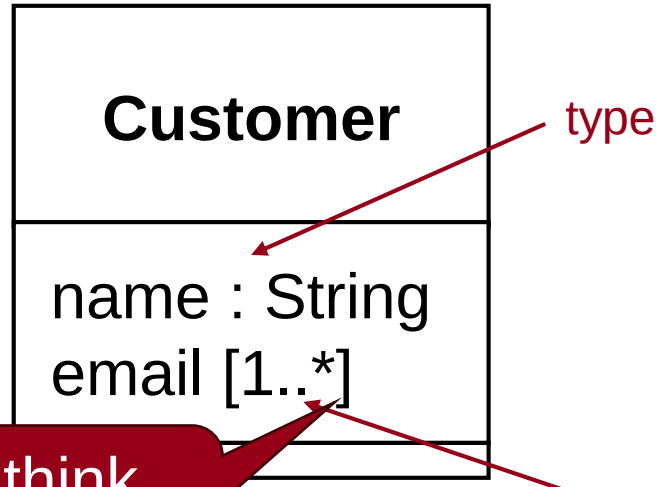
- Naming: noun in the singular
  (from the concepts of the domain)
  - Avoid abbreviations!
  - The UML allows the use of spaces
    (only recommended in analysis models)

Attributes [property]

- Property (attribute): assigns values to instances
  - This value may change “at runtime“
- For classifier: „state variables" of the instances
  - In the case of classifiers they are called: attribute



- Additional to name they may have:
  - Type (built-in type, other classifier, …)
  - Multiplicity [lower limit..upper limit]
    - How many values can belong to it?
    - The meaning of * : unlimited
    - Typically: 1, * (0..*), 0..1, 1..*



You don't need to think about implementation yet!

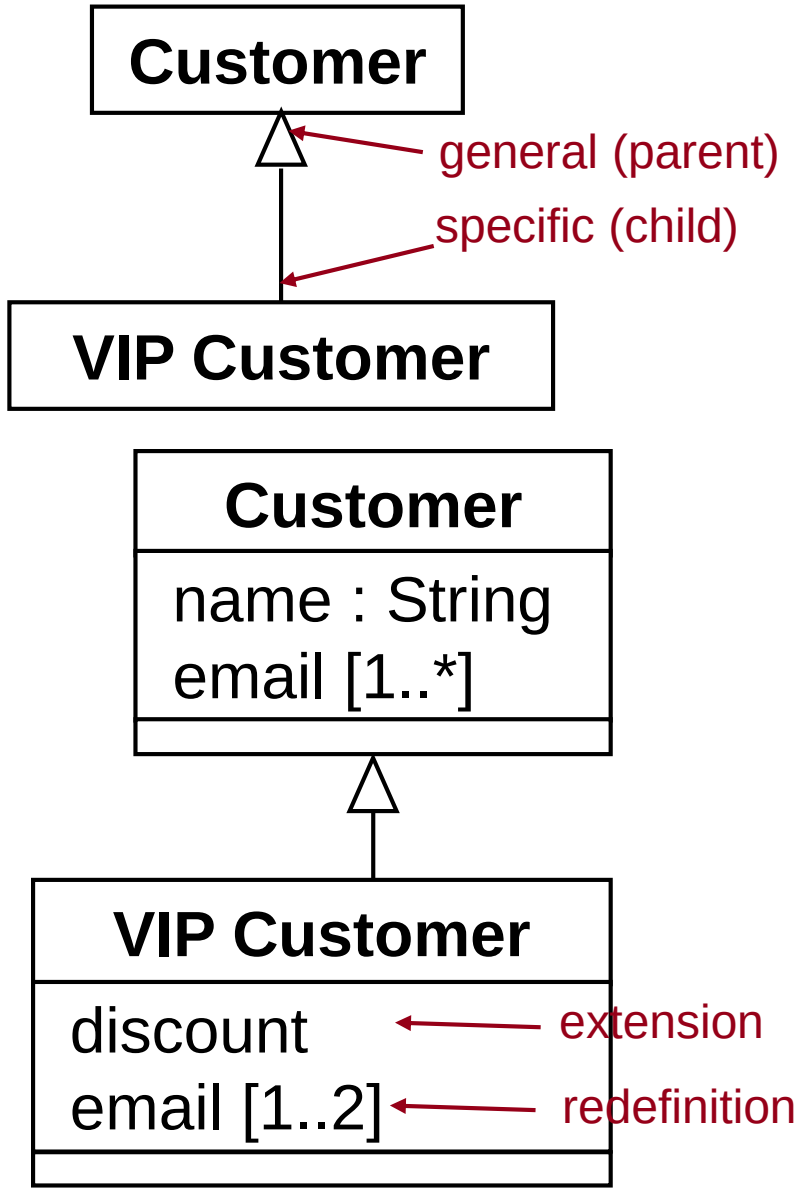
UML Built-In Types [primitive type]

Figure 21.1 Primitive Types

- General, from implementation independent types
- Mathematical integer, natural, real numbers
  - Unlimited cardinality
- Boolean: *true* and *false* values

Generalization

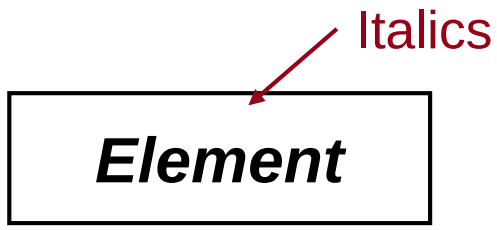
- Generalization/specialisation
  - Inherits the members of the ancestor
    - More transparent, easy to maintain model
  - Substitutability
    - Not always!
- There can be several generalizations of the same classifier
- May expand or redefine its ancestor
  - Both the structure and the behaviour
  - Redefinition: in a non-contradictory way



Abstract Classifier

- Must not have instances
  - Typically its successors will have instances

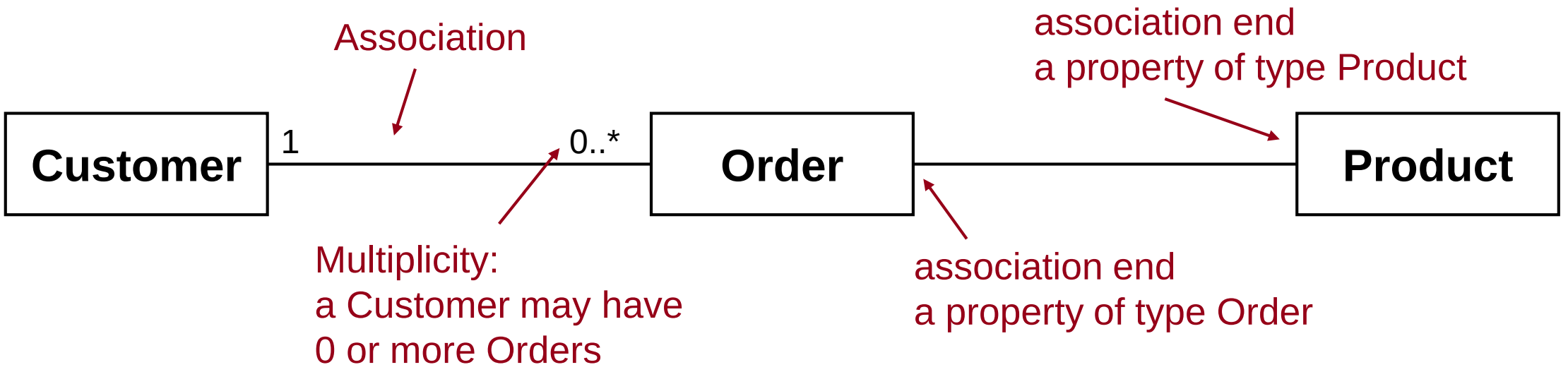
- Notation:



- (Not only a class may be abstract, but also a behaviour …)

Association

- Semantic relationship between instances (of certain types)
- „Permanent” relationship (not only for the duration of a call)



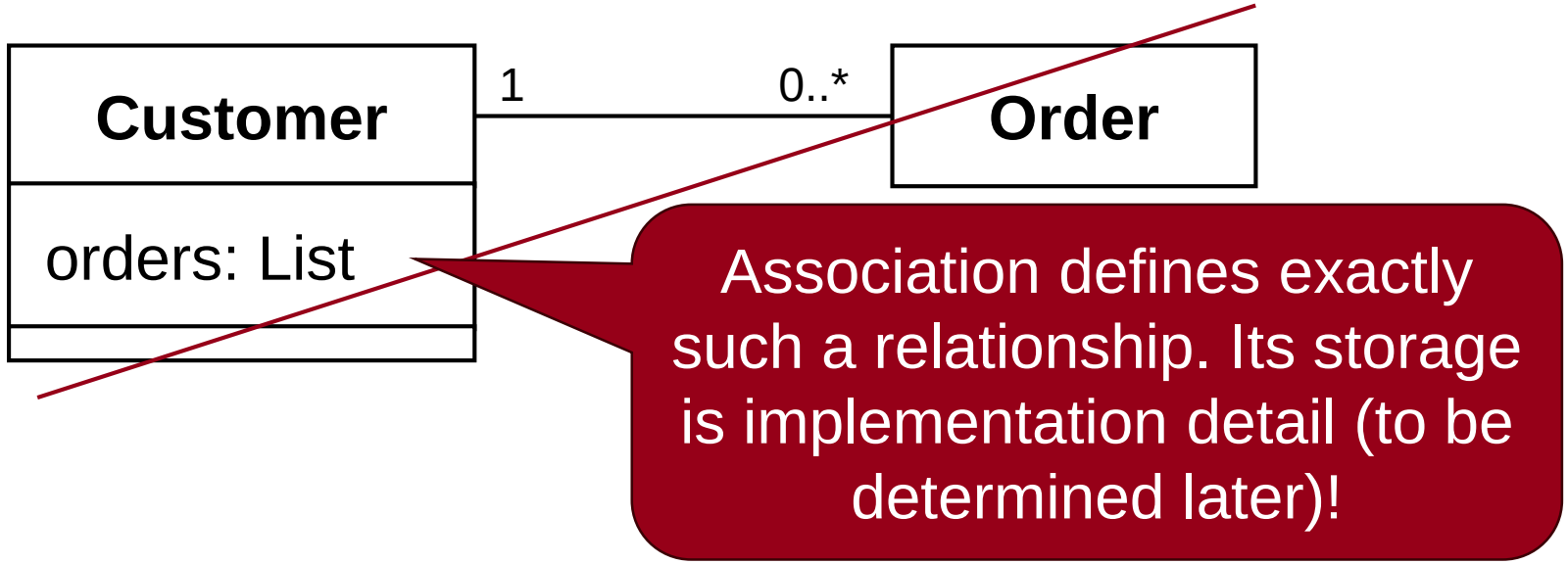
- Association end: it is also a property
  - Usually the association stores these attributes.

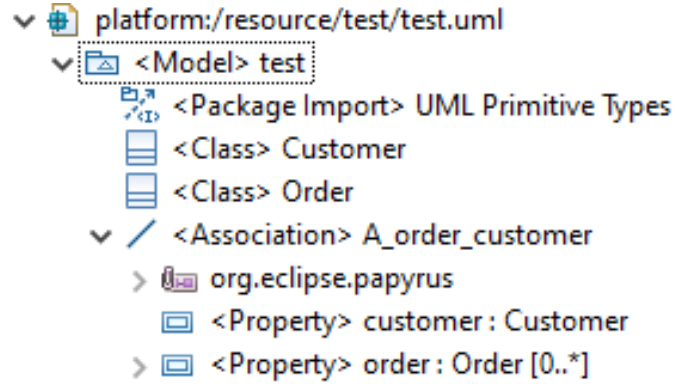
Attributes assign values to instances

“You don't need to think about implementation yet!”

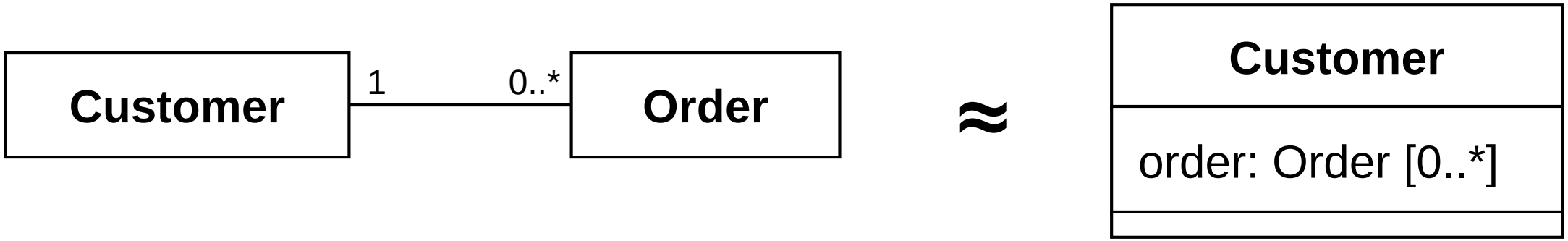
- A separate attribute does not necessarily have to be included for an association.
- This is not the task of modelling.
- We may not even need a separate attribute later on.
- Sometimes associations are implemented differently:
  - as an attribute
  - with relation classes
  - ...

Associations and Attributes





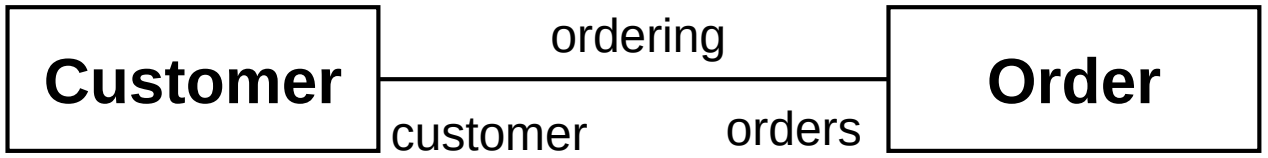
ATTENTION: the attribute and the association model similar concepts, just emphasise something different, they can be extended in different ways



Naming Associations

- The association and its ends can be named separately
  - The names of the ends are the corresponding classifiers in lower case
- It is worthwhile if they cannot be guessed unambiguously

Unnecessary written out names



Good example: The name contains additional information

Source: uml-diagrams.org

- This is especially important when there are multiple associations between the same classifiers

Exercise: MTMT Concept Model (1)

MTMT stores publications. An example of a publication is a book, a part of a book or a journal article. A book is non-periodical, but a publication in one or more volumes that form a complete publication. A journal is a uniformly edited publication that appears at regular intervals. Conferences may also have publications, which may be either a journal article or a book part. The publication has at least one author. The order of authors is important, the first and the last author may play a prominent role. Authors may be attributed to the institution(s) in which they work. A communication may cite another communication.

An Important Goal of Modelling

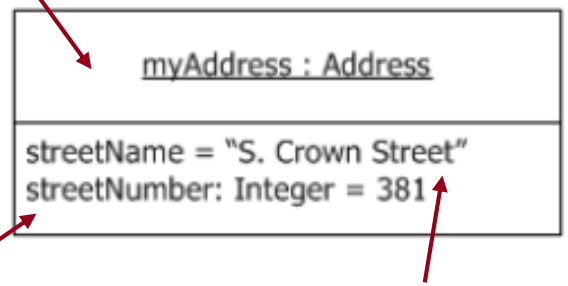
- An important goal is the questions that arise
  - Where should an association be drawn in?
  - What are the multiplicities?
  - Can typical examples be instantiated?
  - Which terms are identical?
  - Which terms have different subtypes?
  - ...

Instances: InstanceSpecification

In UML there is NO such static model element as an object

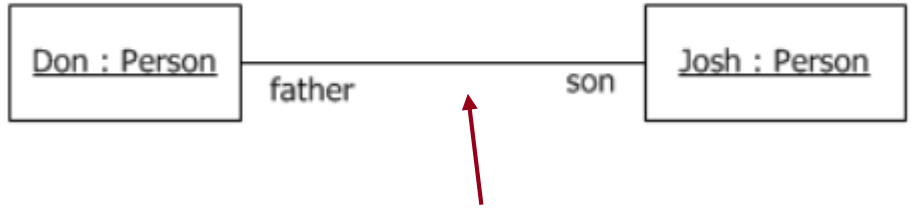
- Instances can be modelled: InstanceSpecification
  - Examples of possible or existing cases
  - Mostly incomplete, abstract descriptions
- The specification of the instance name and the types are optional

InstanceSpecification



Slot

LiteralSpecification
(a specified value)



Link
(instance of an association)

Exercise: MTMT Concept Model (2)

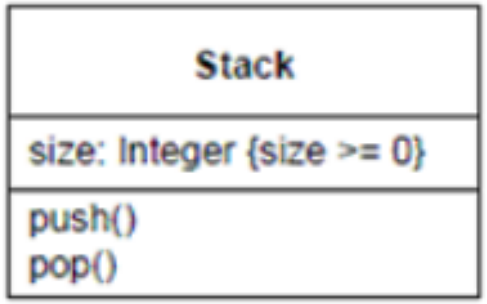
- Draw example models to understand the model!

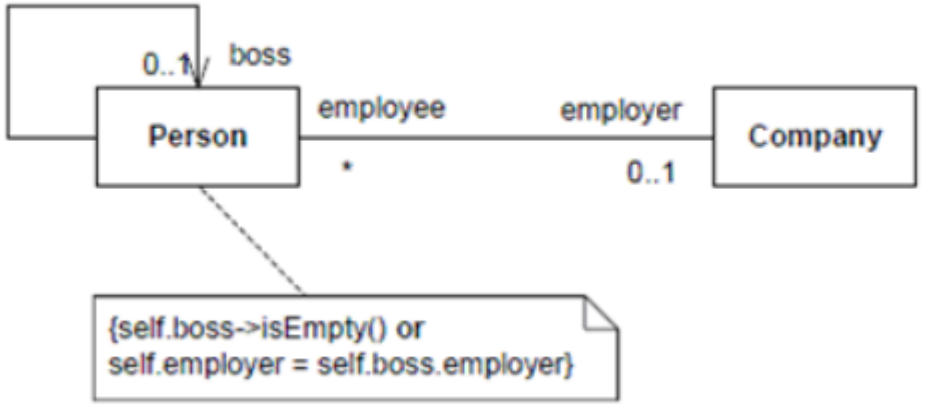
- For example:
  - A journal article with one author
  - A conference publication with two authors, published as a book part
  - A book whose author has two jobs
  - ...

- Do you have new questions about modelling after the examples?

Constraints

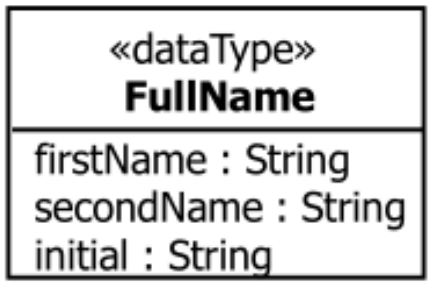
- Additional constraints that the valid models must fulfil
- Many different elements may get them
  - Class, attribute, operation, ...
- Notation: between { } or in separate comments

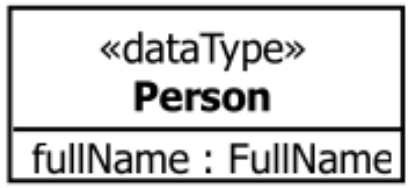


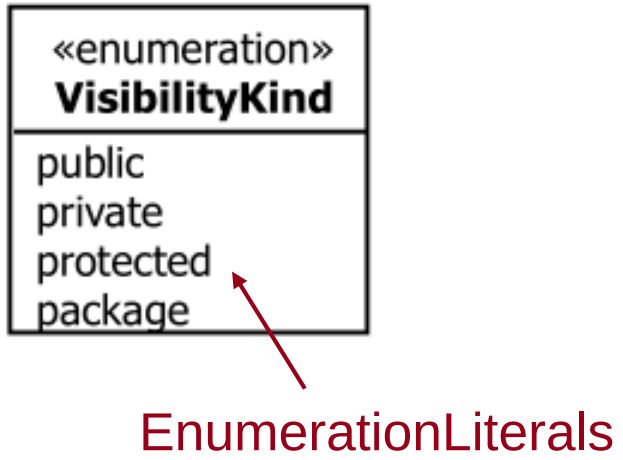


DataType, Enumeration

- DataType: its instances differ in their values only







Multiple Associations

- Not only binary associations can be modelled

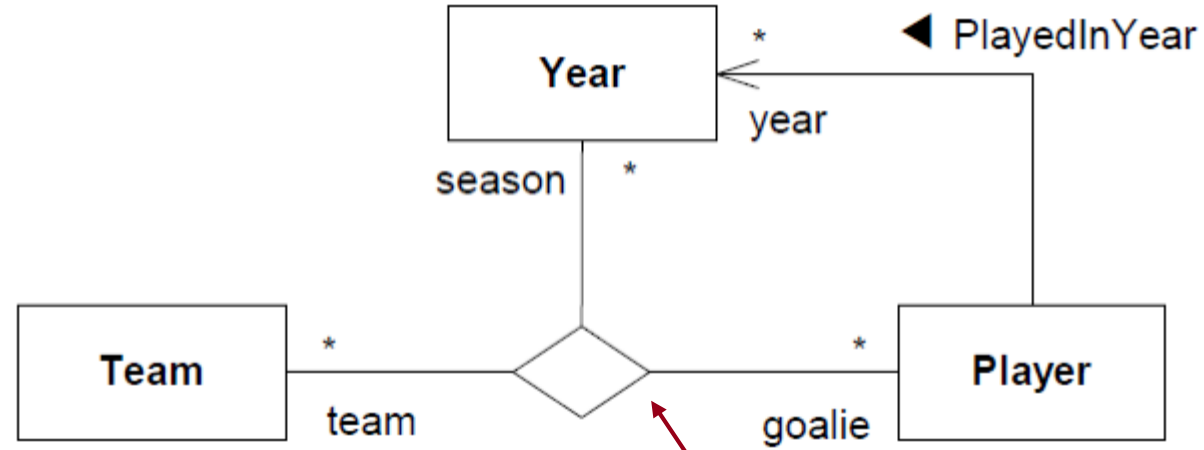
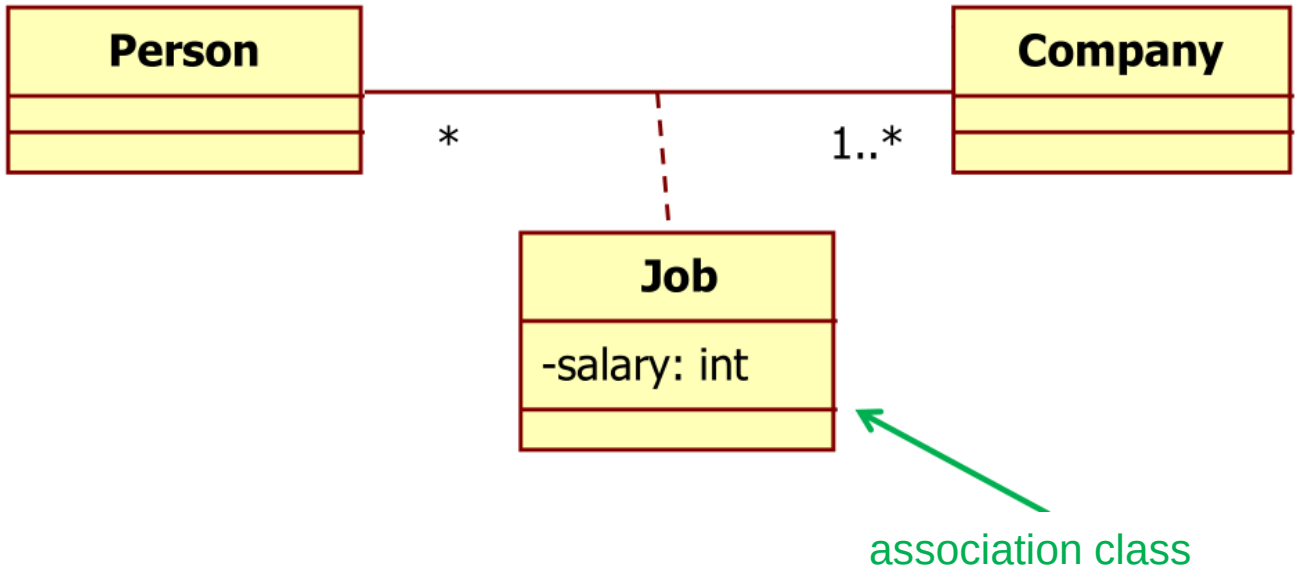


Figure 11.27 Binary and ternary Associations

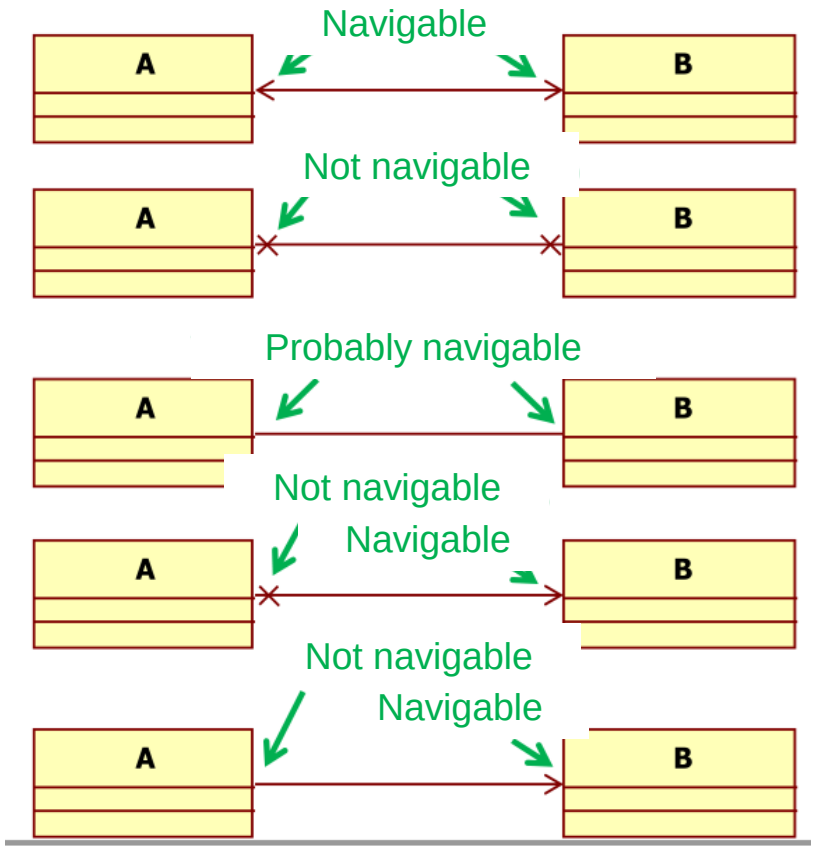
Association Class

- Some information belongs to the association
  - Useful for early modelling
  - Can be implemented later in different ways



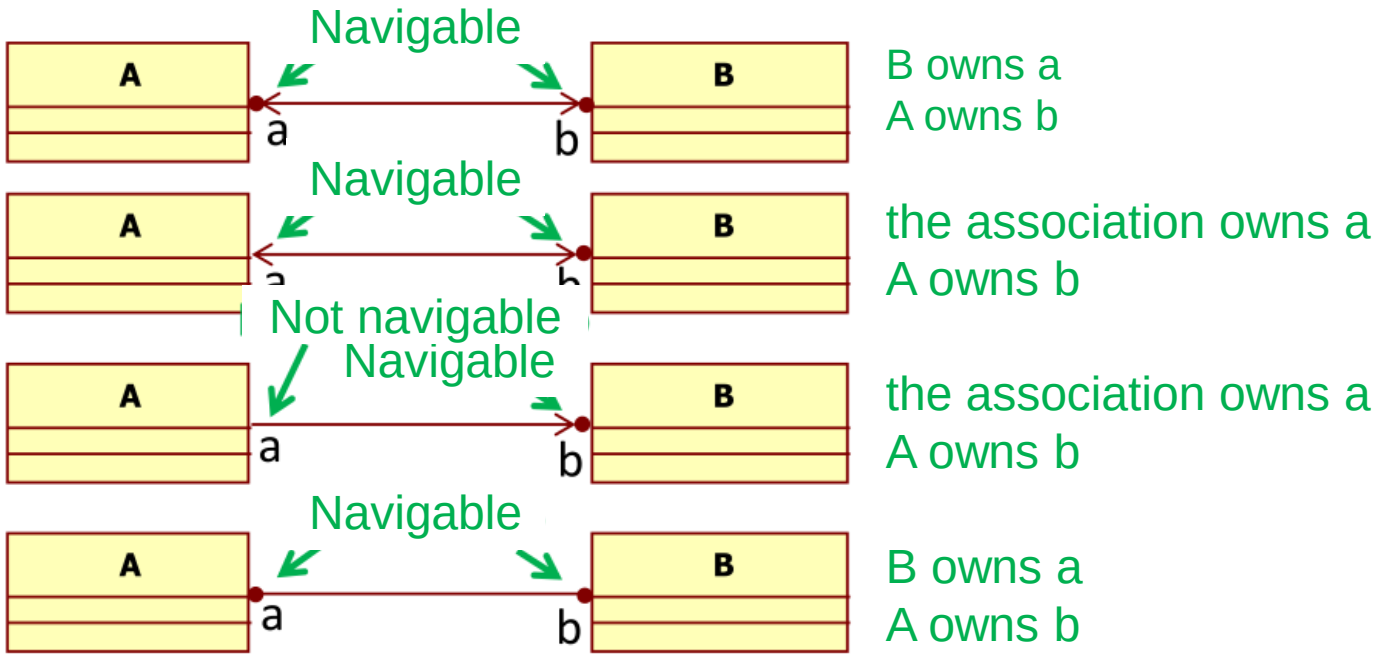
Navigability of Associations

- The instances can be reached from the ones on the other end "effectively"
  - The meaning of efficiency depends on the implementation (e.g. there are references)



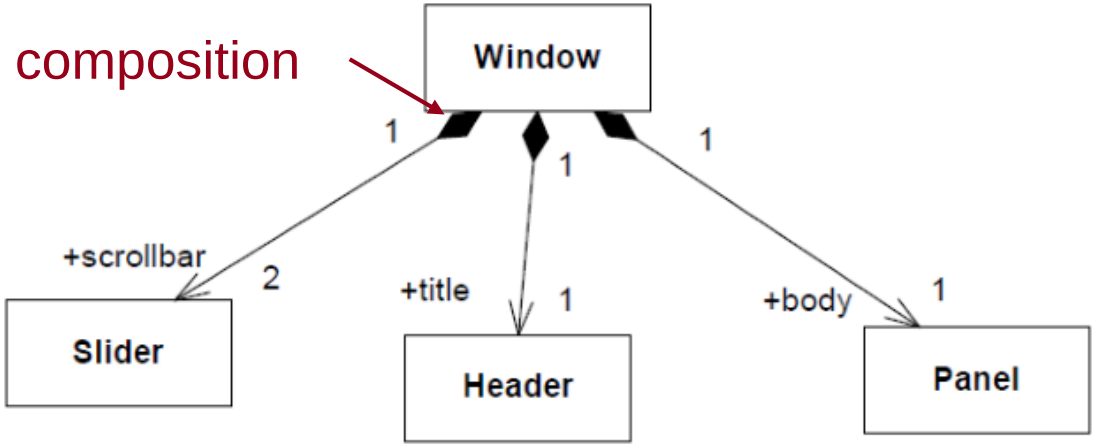
Owning the Association Ends [owned end]

- The property representing the association end may be owned:
  - The association itself
  - The classifier on the other end (notation: dot on the end of the line)
- Not each tool supports it, it is seldom used



Composition, Aggregation

- Aggregation (shared): grouping of instances
  - One instance can participate in more than one grouping
- Composition aggregation (Composition): part/whole relation
  - Stronger form: one instance can only participate in one composition
  - If the composite instance is deleted, the parts are also deleted
- Always a binary association, asymmetric, must not build cycles



„The distinction between aggregation and association is often a matter of taste”

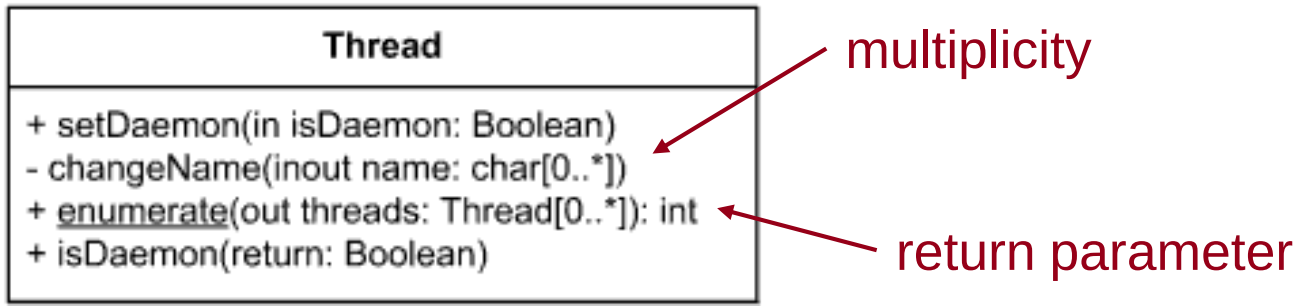
Exercise: MTMT Concept Model (2)

Publications may be of a scientific, educational or informative nature. A journal article can be a scientific paper, an abstract or a conference paper. An abstract may be up to 2 pages long. A book section may be a book chapter (part of a book divided into thematically closely related sections), a conference contribution or a foreword. A citation can be independent or dependent (when the citing and cited book have a common author). For the author, it should be stored from when to when he/she worked in an institution (the end date can be blank for the current workplace). For a publication, it should be stored who created it. In addition to the authors, an administrator can also create a publication. A conference proceedings has editors.

Definition of the Behaviour

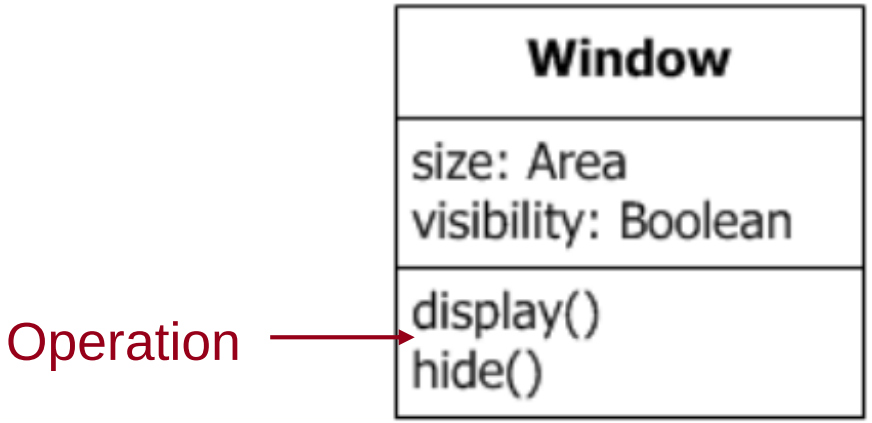
BehavioralFeature, Parameter

- BehavioralFeature: reacts to the invocation of a Feature
  - Specification of the behaviour: text, state machine, activity, …
- Can be static (notation: underlined) and/or abstract
- Parameter: it may have parameters
  - Direction: in, inout, out, return
  - Some parameters may have multiplicity (0 as lower limit means: optional)



Operation

- Operation: behaviour of a Class, DataType or Interface
- Can be invoked synchronously or asynchronously
- Method: the behaviour implementing a BehavioralFeature
  - UML supports modelling polymorphism
  - The method to be invoked may belong to a descendant

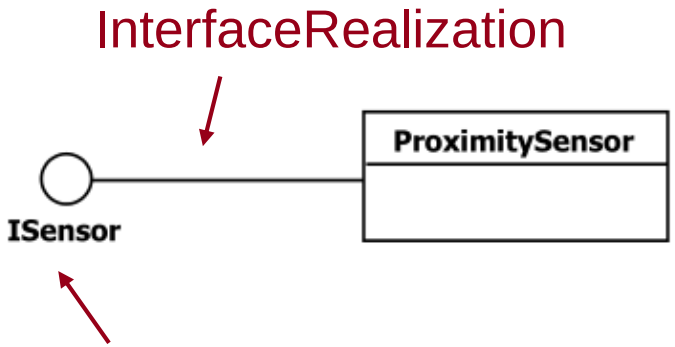


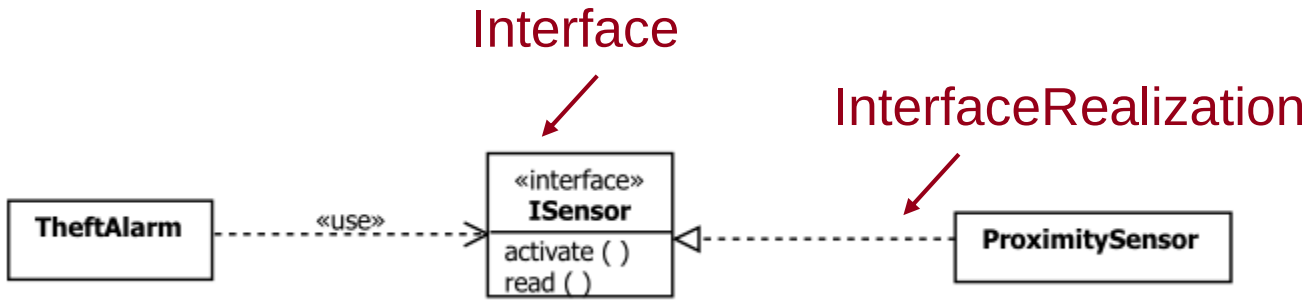
Signal, Reception

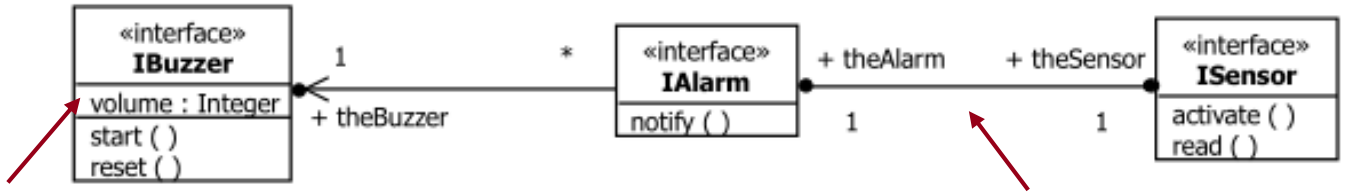
- Modelling asynchronous communication
  - The caller does not wait for a reply (does not block)
- Signal: specification of the kind of the (asynchronous) communication
  - May have attributes
- Reception: the class can receive Signals of the given type

Interface

- Public, coherent “contract”
- Notation is the same as for the interfaces of the components





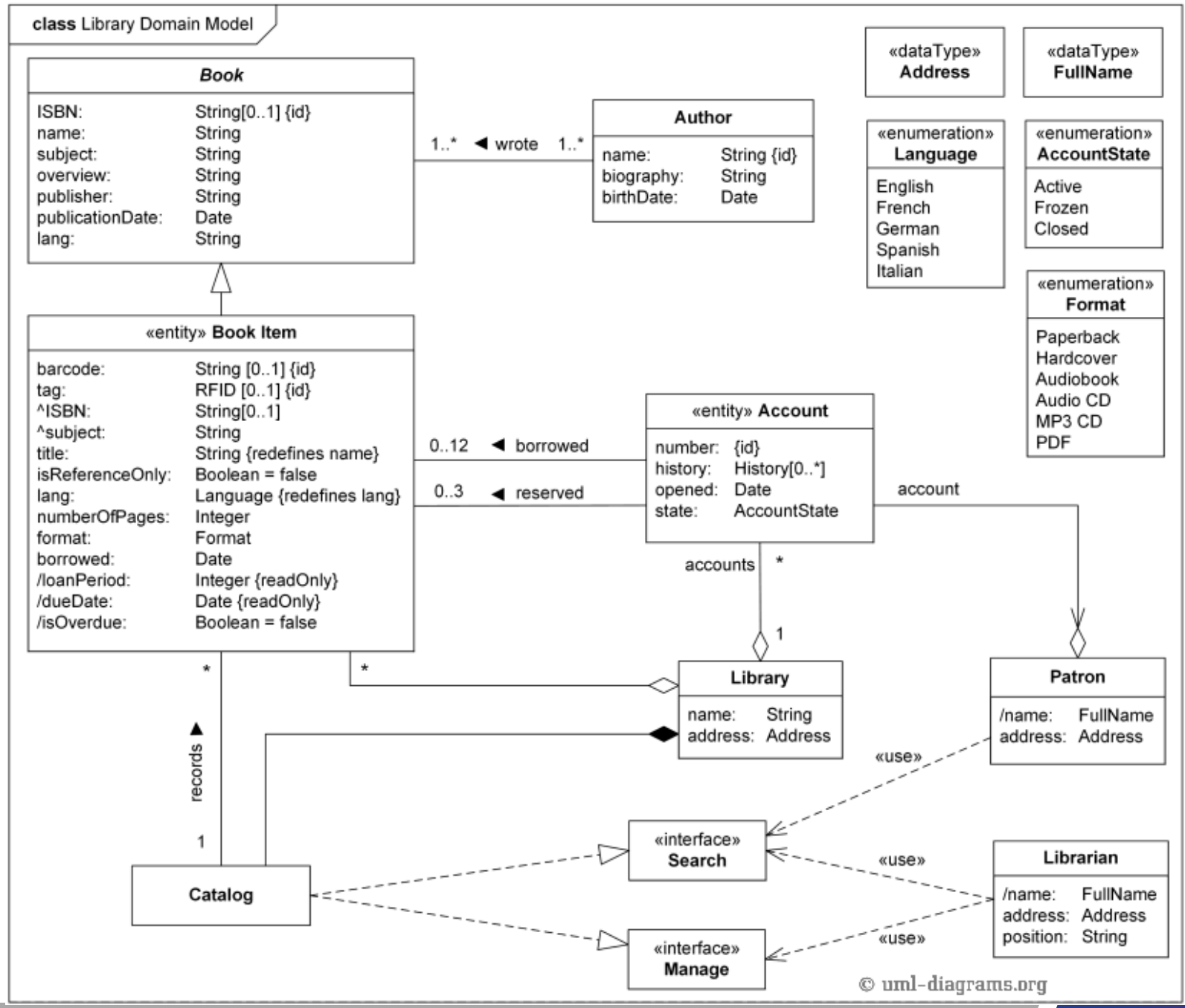


Attribute on an interface (carefully!)

Association between interfaces

Example: Complex UML Class Diagram

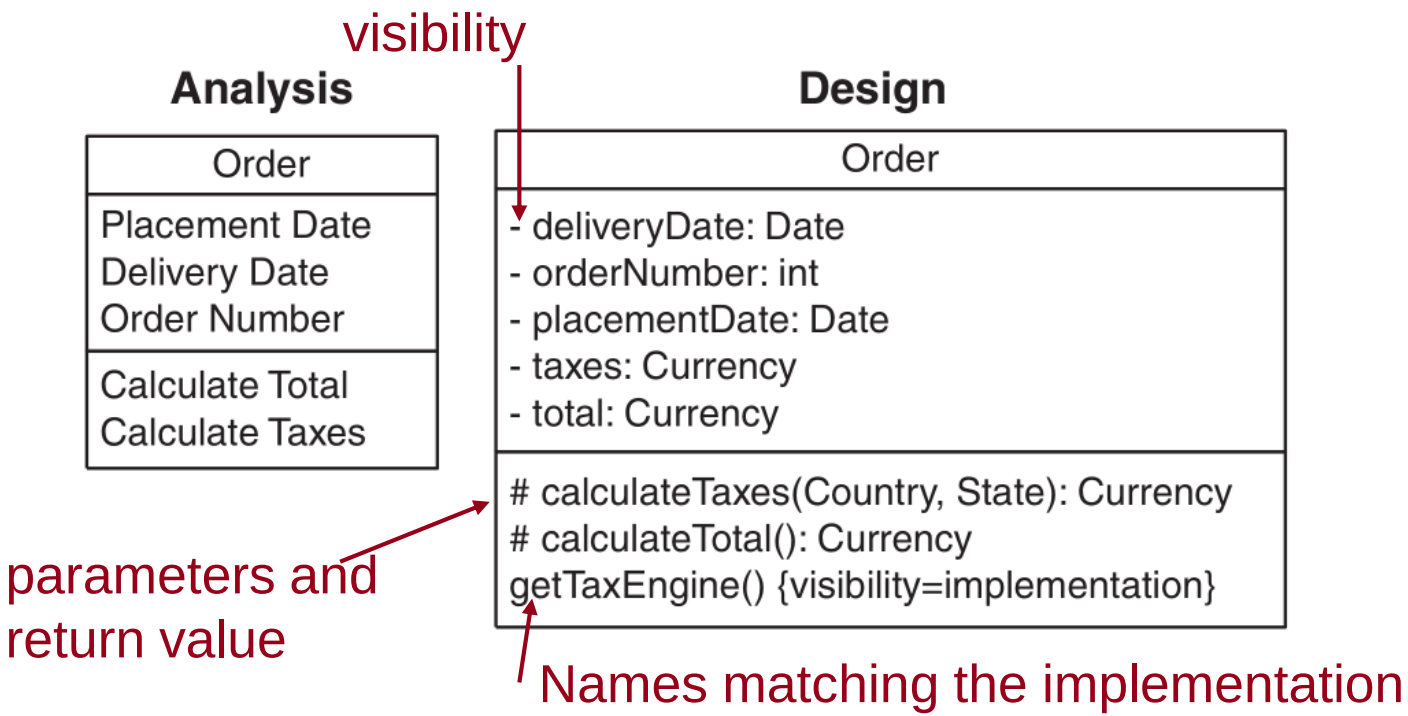
Can you summarise in 3-4 sentences what domain it describes and what are the main responsibilities?

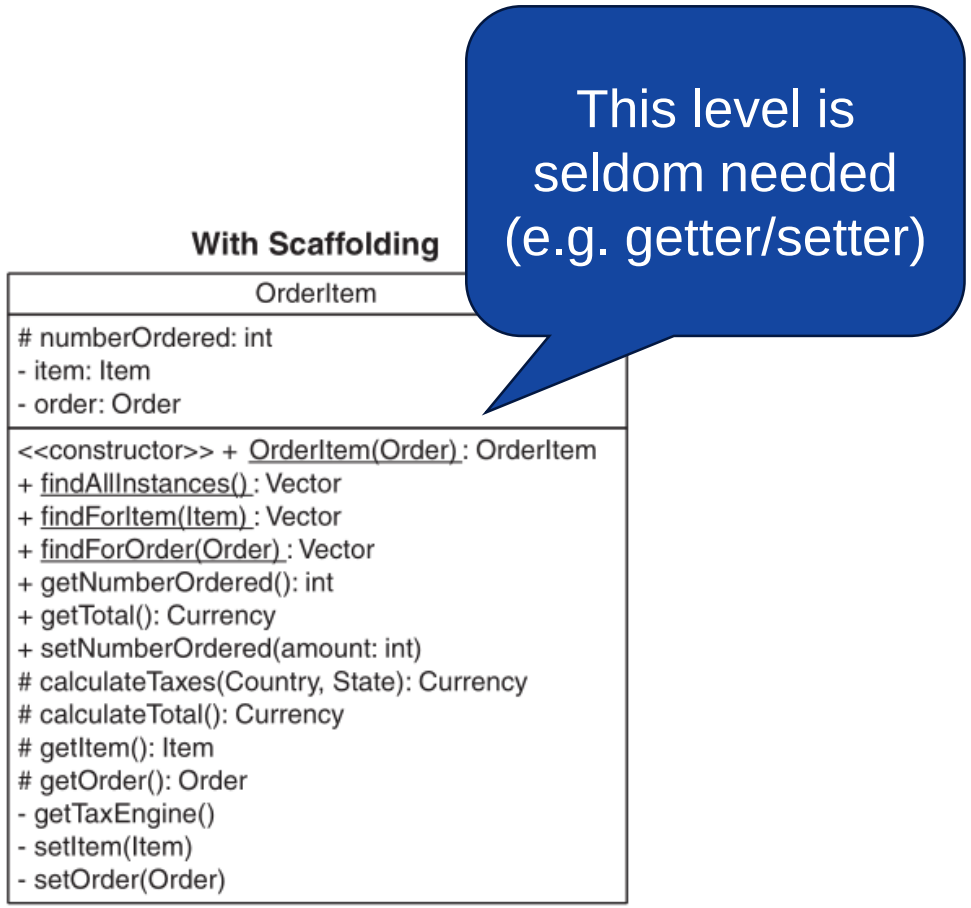


Implementation View and Possible Implementations

Analysis and Detailed Views

- As development progresses, more and more details are added
- At which level should we stop?





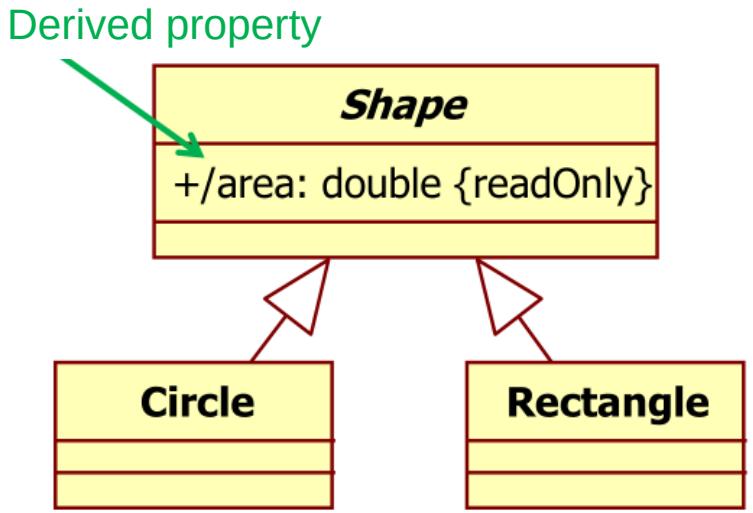
Visibility

- Can be specified for operations and properties
- Levels of visibility:
  - **private** (**-**): visible only for the members of the given class
  - **protected** (#): visible only for the members of the given class and for their descendants
  - **public** (**+**): visible for anyone who can access the class
  - **package** (**~**): visible for anyone in the same package

- CAUTION: the exact semantics may differ from the ones of some programming languages

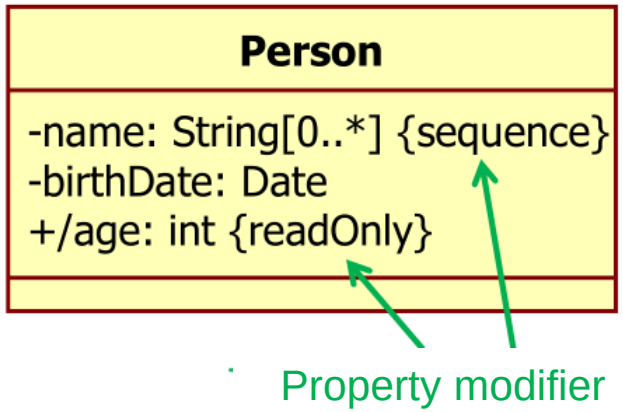
Derived Property

- The value of the property is the result of a calculation
- The calculation itself is not always specified
- Notation: a slash (/) before the name



Property Modifier

- Clarification of the meaning of a property
- Notation: in brackets after the property



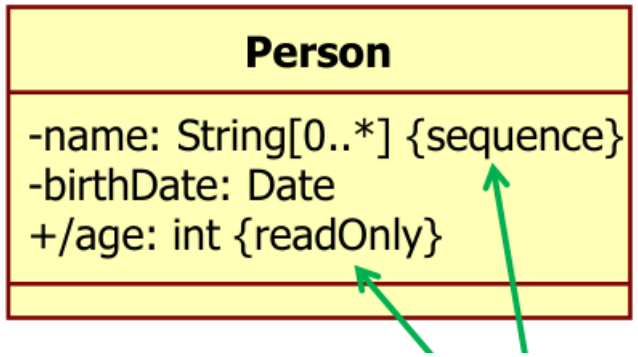
Modifier	Description
readOnly	Property is read only
union	Property is a derived union of its subsets
subsets <propname>	Property is a subset of the <propname>
redefines <propname>	Property redefines an inherited <propname>
ordered	Property is ordered
unordered	Property is unordered
unique	Multi-valued property has no duplicate values
nonunique	Multi-valued property may have duplicate values
sequence (or seq)	Property is an ordered bag (nonunique and ordered)
id	Property is part of the identifier for the class

Also can be used on association ends!

How Would You Implement it? (Example)

Java mapping:

```
public class Maze {
  private ArrayList<Field> fields;
}
public class Field {
  private Maze maze;
}
```



Property modifier

Java mapping:

```
public class Person {
    private List<String> name;
    private DateTime birthDate;

    public List<String> getName() { return name; }
    public DateTime getBirthDate() { return birthDate; }
    public void setBirthDate(DateTime value) { birthDate = value; }
    public int getAge() { /*...*/ }
```

Further Elements and Their Application

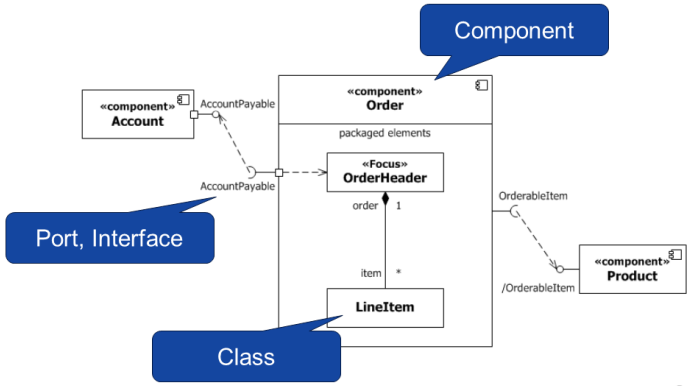
- Nested Classes
- Qualifier: the qualified association end partitions the instances on the other side
- Package: hierarchical organisation of the other elements
  - Like in programming languages: `namespace`, `package`, …

- Generalization can also be used on associations
- Dependency can also be used on classes

Summary

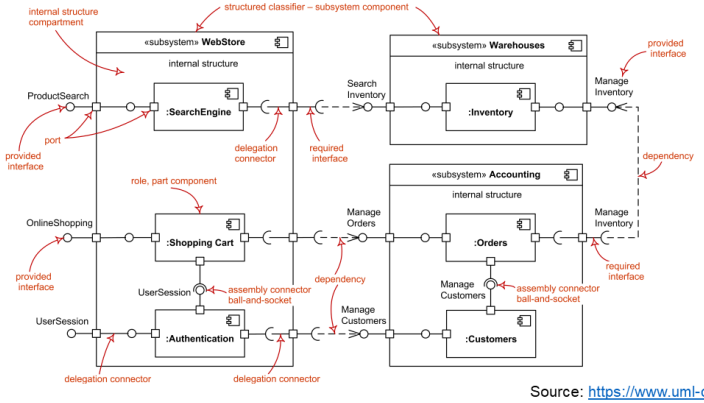
Summary

UML Structural Models



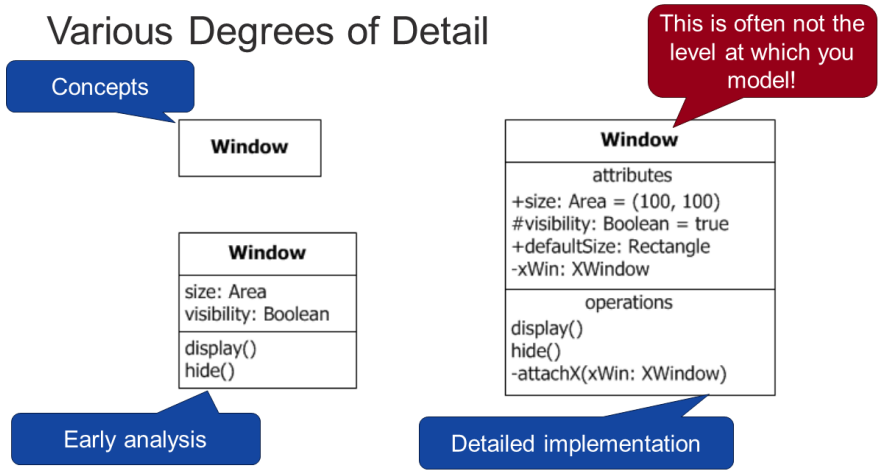
Source: UML v2.5.1

Port – Summary



Source: https://www.uml-diagrams.org/

Various Degrees of Detail



Example: Complex UML Class Diagram

Can you summarise in 3-4 sentences what domain it describes and what are the main responsibilities?