ES Software Engineering

Lecture 3

Unified Modeling Language

KATEDRA INŻYNIERII KOMPUTEROWEJ

intel

Sponsor specjalności

In the previous lecture

1. Requirements engineering
2. System Requirements Specification
3. Requirements defects
4. Search technique to find requirements defects
5. Examples

Plan of the lecture

1. What is UML and its history
2. UML applications
3. UML diagrams
4. Class model
5. Class model – misunderstanding
6. Class diagram
7. Relations in UML class diagram

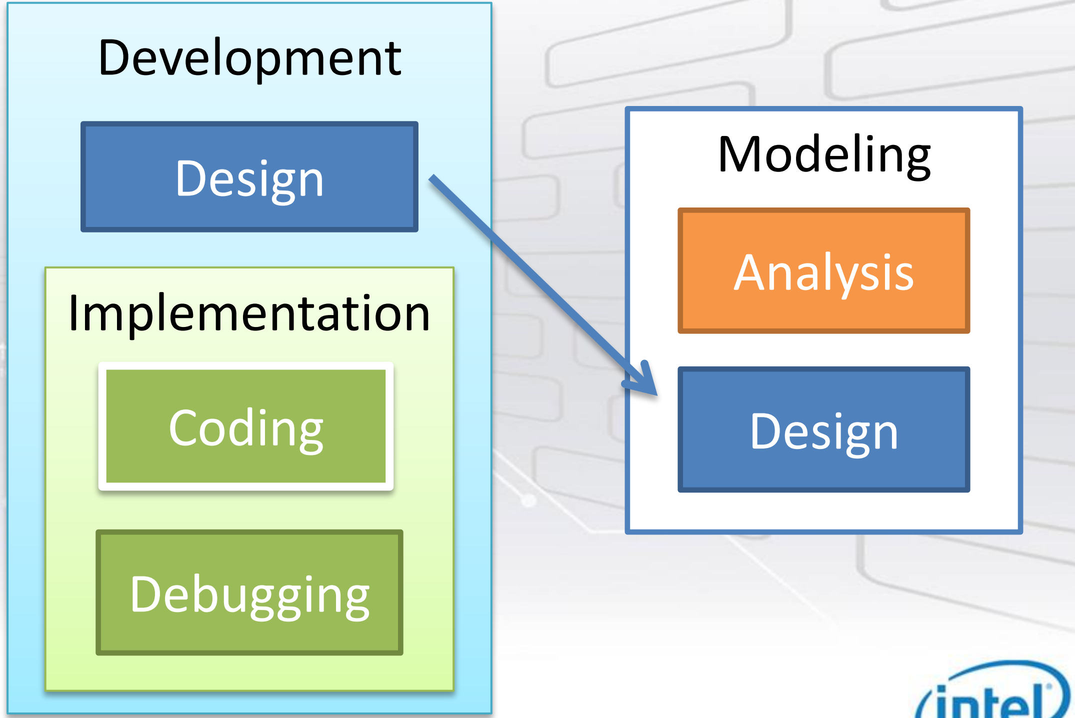
History of UML

- The history of UML goes back to the 70’s, the time of the first object-oriented programming languages
- 1996 – the first documentation of version 0.9 Unified Method (Rational Software); creating the UML Consortium (HP, IBM, Oracle, Microsoft) and the emergence of UML 1.0.
- Since 1997, until today, UML is developed by the Object Management Group (OMG)
- 2005 – more than 100 organizations have developed UML 2.0; mainly modeling for embedded systems was improved.
- 2012 – standardization of UML 2.4.1 (ISO/IEC 19505-1 & 19505-2)

Unified Modeling Language

What is UML?

- used for modeling
- used for design
- too high for implementation



Unified Modeling Language

UML (Unified Modeling Language)

Semi-formal modeling language for various systems developed by the Object Management Group. It is used for modeling fragments of an existing reality or one to be developed, mostly for modeling IT systems. Other applications: modeling business processes, systems engineering and organizational structures engineering.

UML is used, as a rule, along with a graphical representation – symbols connected on diagrams.

UML is officially defined in the UML metamodel, i.e. Meta Object Facility (MOF).

UML metamodel and UML models are serialized in the XML Metadata Interchange (XMI) language.

To define limitations in UML, the Object Constraint Language (OCL) developed by IBM is used.

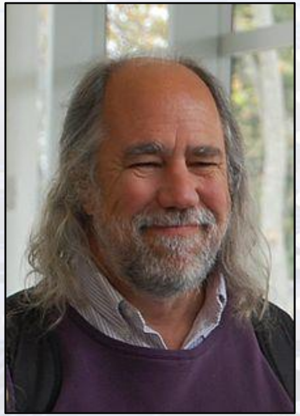
Unified Modeling Language

What UML is not?

- a programming language – however, it is possible to generate source code from diagrams
- a tool – it but specifies tools
- It is not a method of analysis and designing computer systems

Unified Modeling Language

Fathers of UML:



Grady Booch

(Booch method)

Ivar Jacobson

(Use Cases)

James Rumbaugh

(OMT)

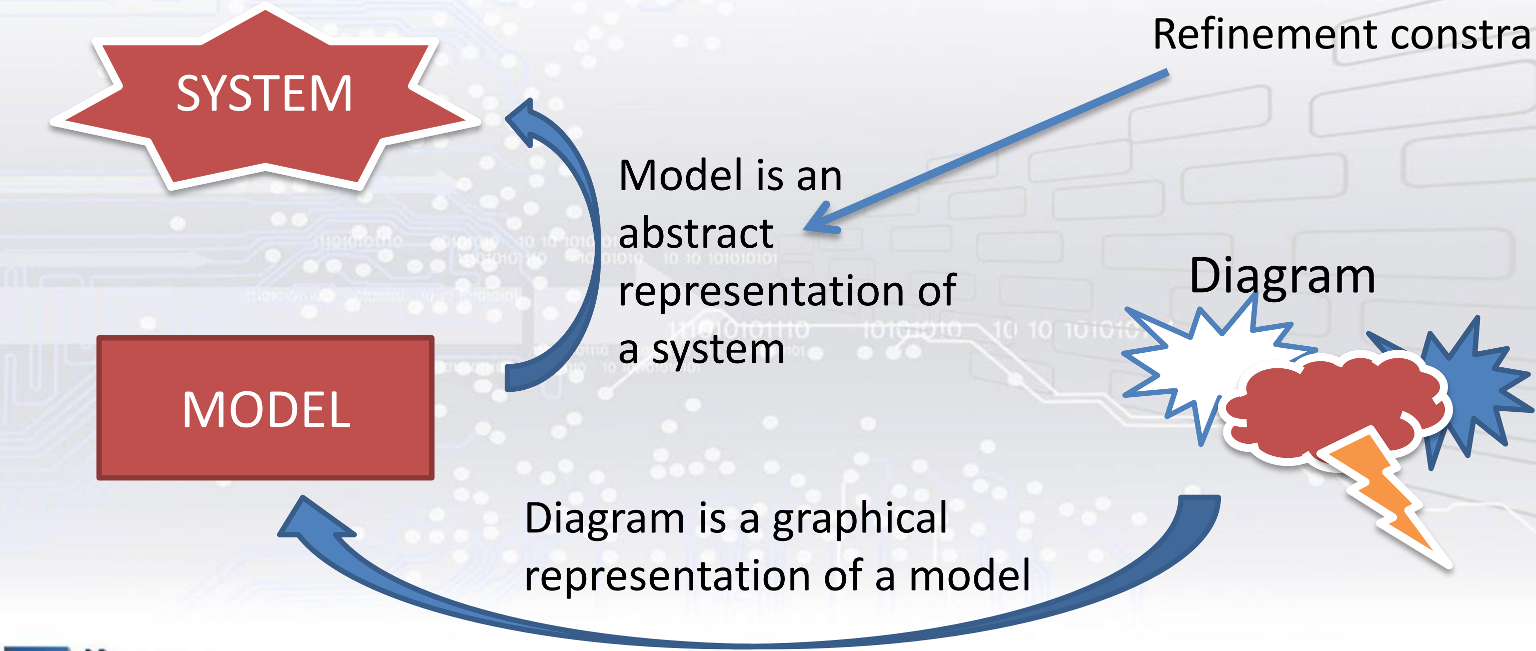
Unified Modeling Language

Fields in which UML is used:

- Banking & finances
- Information sharing systems in corporations
- Computer Science
- Electronics
- Medicine
- Science
- Web services, retail sale
- Transport
- Telecommunication
- And everything else one can imagine.

Unified Modeling Language

Model, Diagram, Abstraction



Refinement constraint problem

SYSTEM

Model is an abstract representation of a system

MODEL

Diagram

Diagram is a graphical representation of a model

UML uses diagrams to model a software system

UML Diagrams

Static structure diagrams

- class diagrams
- component diagram
- deployment diagram

Functionality diagrams

- use case diagram
- interaction diagram
- sequence diagram

Behavioral diagrams

- state transition diagram
- activity diagram

UML Diagrams

The basic division of UML diagrams:

Structural diagrams of:

- Classes
- Objects
- Packages
- Components
- Implementations
- Complex structures

Dynamic diagrams of:

- Use Cases
- States
- Workflow
- Actions
- Cooperation
- Interactions
- Timing conditions

Class model

Class model purpose

- Static analysis of a problem domain
- Basis for functional and behavioral analysis
- Application logic design
- Help for implementation
- Walking down through various levels of abstraction

Class model

Class diagram misunderstanding

1. Developers treat diagram classes as software structures (data & functions)
2. Modelers treat classes as abstraction of real world entities

Modeling classes – no need for:

- Many classes
- Many class attributes
- Class functions

ONE MODEL CLASS == MANY SOFTWARE CLASSES

Class model misunderstanding

Modelers	Developers
Entity – sth that exists in the real world and has a large set of features	**Object** – sth that exists in the model and has a limited set of features
Class – An object or set of objects	**Object** – A concrete instantiation of a class
Property – an informative feature of a class	**Attribute** – an information held by object
Relationship – a logical binding between classes	**Link** – a concrete binding between objects
Generalization-specialization – an ontology relationship between two classes	**Inheritance** – a mechanism on passing features from a generalized to a specialized class
Multiple inheritance – when a class has multiple generalizations	**Single inheritance** – when a class has only one direct generalization

Class model misunderstanding

Modelers	Developers
Aggregation – a relationship between classes that allows joining two or more objects together and creating a new object	**Containment** – a relationship between classes that allows including one or more objects in another object
Association – any other relationship between two or more classes; associations between more than two classes are hard to implement	**Pointer, reference** – a specific implementation of an association; only two objects are undirectionally linked, (bidirectional association requires two pointers/references)
Association role – a name of an object when it is linked to another object	**Class name** – can be used as a role substitute but only when classes at opposite sides of the association are different

Class model misunderstanding

Modelers	Developers
Association direction – a direction in which an association should be read	**Association navigation** – determines the class where a pointer/reference should be implemented
Collection, list – a container of objects	**Array** – a specific implementation of a collection
Multiplicity – a potential count of contained or aggregated objects or associated objects playing the same role	**Array size** – a concrete count of objects in an array
Operation – an abstract action that can be performed on an object of the class	**Function** – a concrete implementation of an operation; needs parameters specification; can be overloaded and overridden

Class model misunderstanding

Modelers	Developers
Class name – a name of a class	**Object name** – can be different from or the same as the class name
Class name – a name of a class when it is used in a model; it can consist of two or more words and national letters	**Class identifier** – a name of a class when it is implemented in a program; only one word is allowed, national letters in some languages
ID property – not needed in analysis	needed in database design
Visibility (private, protected, public) – not needed in analysis; all features are public	can be used in design

UML construction

Notation:

- Graphical elements
- Modeling language syntax
- The essence of sketching models

Notation of diagram elements. The notation is more important for the analyst because it allows to understand the model by others.

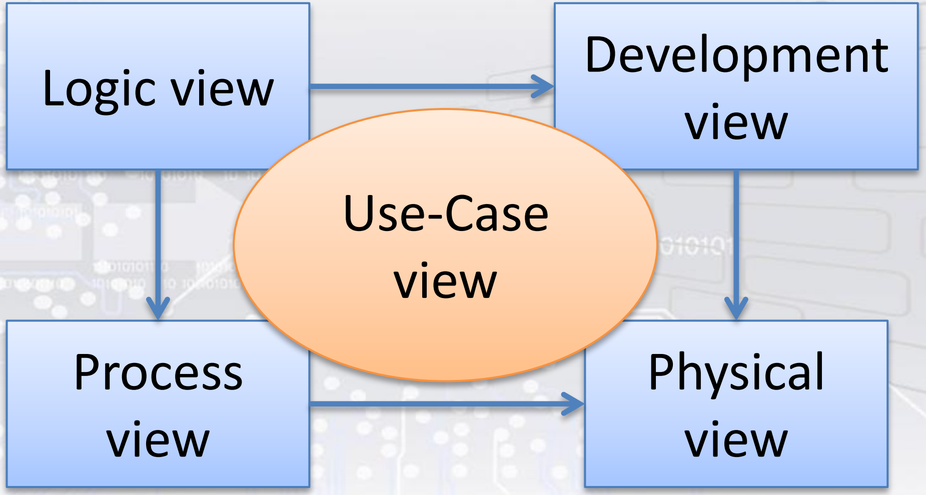
Metamodel:

- Definitions of language concepts and connections between them

Metamodel is the semantic of elements. During implementation it is more important to understand the meaning of elements.

Model views

UML diagrams are divided into views. One of the ways to implement such a task is the Kruchten’s view model 4+1.



Model views

Logical View – modeling of system parts and ways in which they work together. It includes:

- Class diagrams
- Object diagrams
- State machine diagrams
- Interaction diagrams

Process View – describes and visualizes processes and cases occurring in the system. Consists of Action diagrams.

Model views

Design View – models the way of organizing system parts into modules and components. Consists of a:

- Package diagram
- Component diagram

Physical View – explains the way in which the system design, described in logical, process and design views, works in the form of real objects. The view is the closest to the process of the actual deployment of the system and includes deployment diagrams.

Model views

Use Cases View – describes the functionality of modeling such a system from an external perspective. This description also includes the purpose of the system. All 4 views refer to the Use Cases view. This view contains:

- Use Cases diagrams
- Overview diagrams

Class diagram

Class diagram:

- Is one of the most important UML diagrams
- Contains information of static relations between classes
- Classes are closely linked to object-oriented programming techniques

Class diagram

The basic element of a diagram is a **class**. It is defined as a rectangle containing 3 sections:

- a name

```
stereotype class_name label_value_list
```

- attribute

```
stereotype accessibility attribute_name : type = start_value
label_value_list
```

- operations

```
stereotype accessibility method_name (arg_list) :
return_value_type label_value_list
```

All elements of a class specification, except the name, are optional.

Class diagram

Accessibility of method:

- **+** public
  - private
  # secured
  ~ package scope

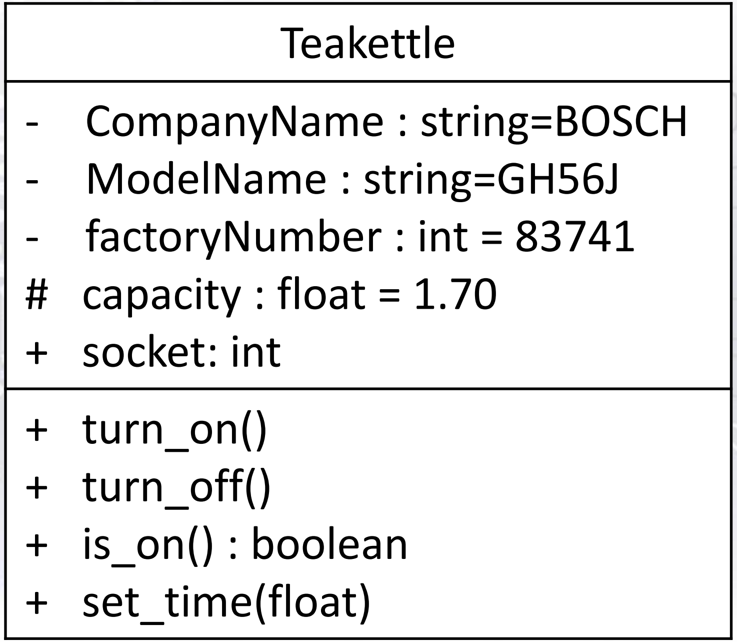
arg_list: type arg_name : type = start_value

A type defines the way in which the method uses a given argument:

- in – the method can read an argument but cannot modify it
- out – can modify, cannot read
- inout – can read and modify

Class diagram

Example:



Teakettle

- CompanyName : string=BOSCH
- ModelName : string=GH56J
- factoryNumber : int = 83741
# capacity : float = 1.70
+ socket: int

+ turn_on()
+ turn_off()
+ is_on() : boolean
+ set_time(float)

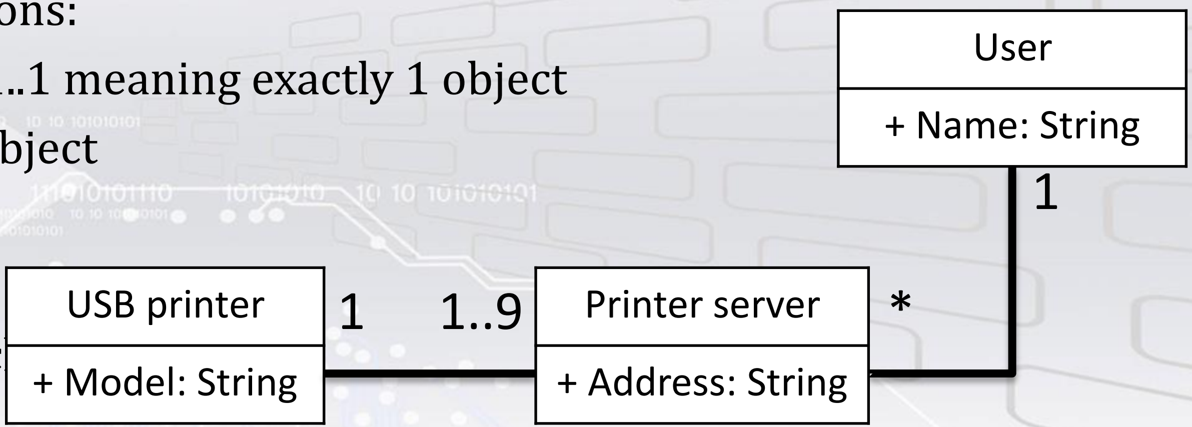
Class diagram

Multiplicity – defines the minimum and maximum number of objects which can be associated with a given class. Saved as:

```
start_value .. end_value
```

Example multiplicity notations:

- 1 – simplified notation 1..1 meaning exactly 1 object
- 0..1 – a single optional object
- 1..* – at least one
- * – any number
- 2, 6, 7 – exactly this muc



Class diagram

Attribute declaration:

[visibility] name [size] [: type] [=start_value] [property]

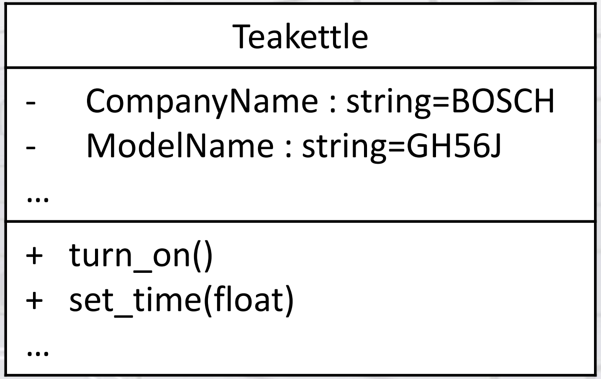
Property:

- Changeable
- addOnly
- readOnly, frozen

Examples:

- ModelName : string = BOSCH readOnly

+ capacity : float = 1.70 frozen



Teakettle

- CompanyName : string=BOSCH
- ModelName : string=GH56J

…

+ turn_on()
+ set_time(float)

…

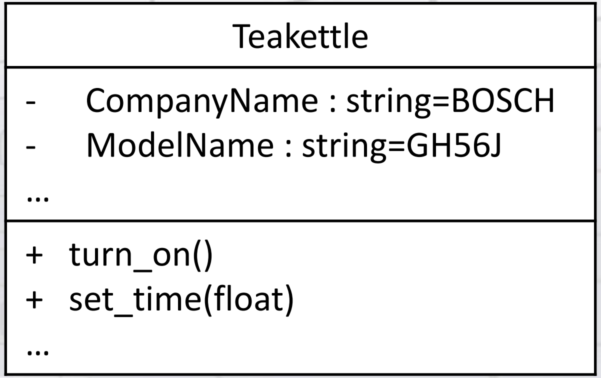
Class diagram

Operation declaration:

[visibility] name [parameters] [: type_of_result][property]

Where parameters are defined as follows:

[mode] name : type [=default_value]



Teakettle

- CompanyName : string=BOSCH
- ModelName : string=GH56J

…

+ turn_on()

+ set_time(float)

…

Mode:

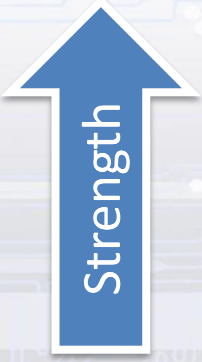
- modifiable: out, inout
- unmodifiable: in

Property:

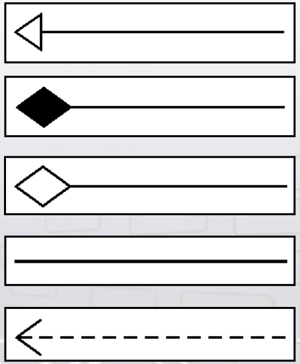
- Leaf – feature which cannot be overwritten (non-polimorfic)
- isQuerry – function not changing the object
- Sequential, concurrent, guarded – concurrent operations

Class diagram

Relations between classes:



- Inheritance
- Total aggregation
- Partial aggregation
- Association
- Relation



Relations

Dependency

One class uses objects from a second class. Changes in one class may cause changes in the second class. A dependency is usually used case of one class using another class as a parameter.

Types of dependencies:

- <<use>> – implementation of the first class needs using a second class
- <<create>> – first class creates an instantiation of a second class
- <<instantiate>> – object *x* is an instantiation of class *Y*
- <<call>> – operations in class *X* cause operations in class *Y*

Relations

Association

Describes a temporary relation between objects of two classes.

Association is a stronger relation than a dependency and objects with associations are independent on each other i.e. removing one object doesn’t cause needs for removing another one.

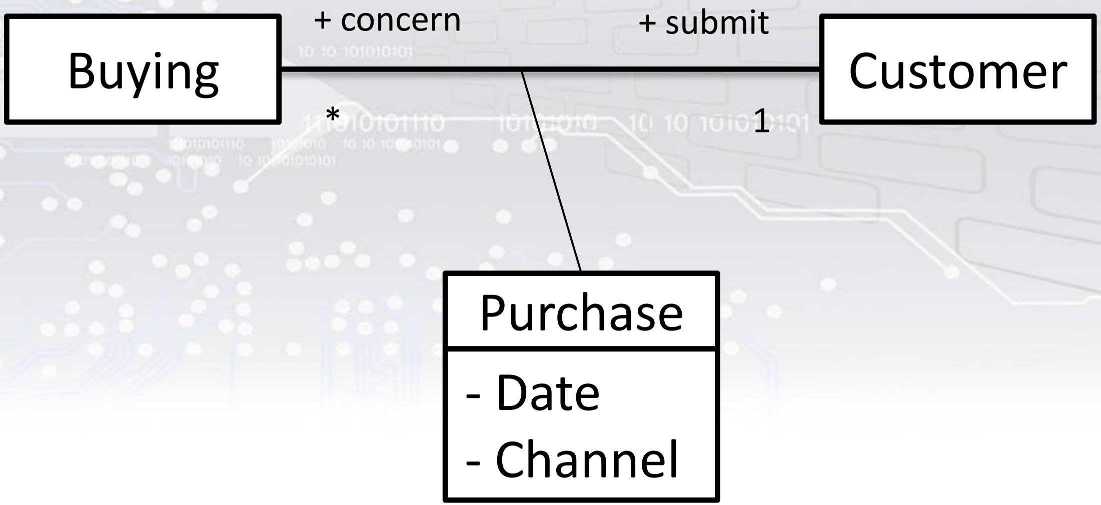
Notation of association contains phrases: {from when < to when} and some verb like "contains", "consists", "is owner", "is contained".

Relations

Association

UML allows to define an association class related to an association of classes.

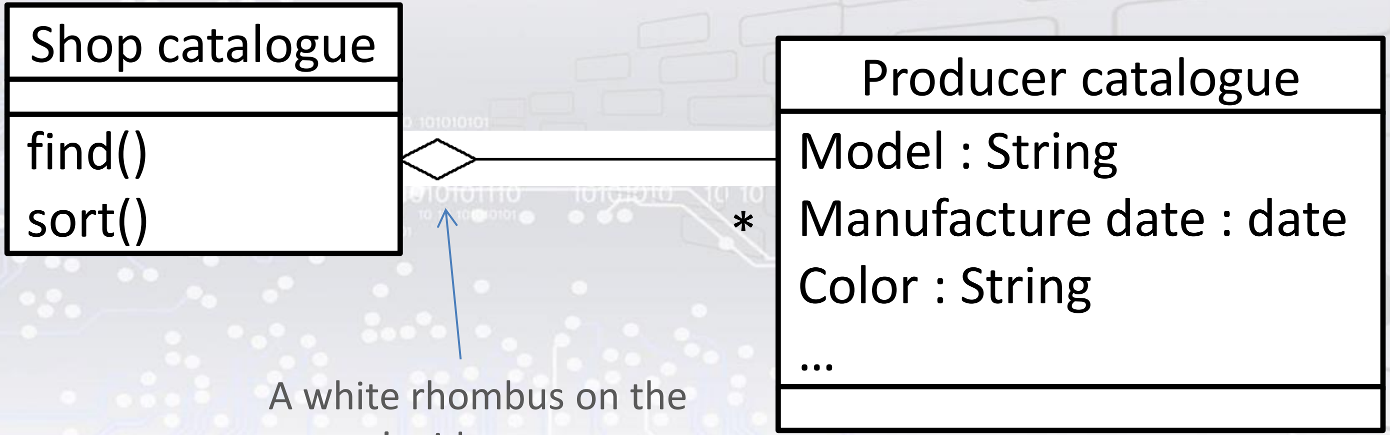
Example:



Relations

Aggregation is a relation *whole-part*

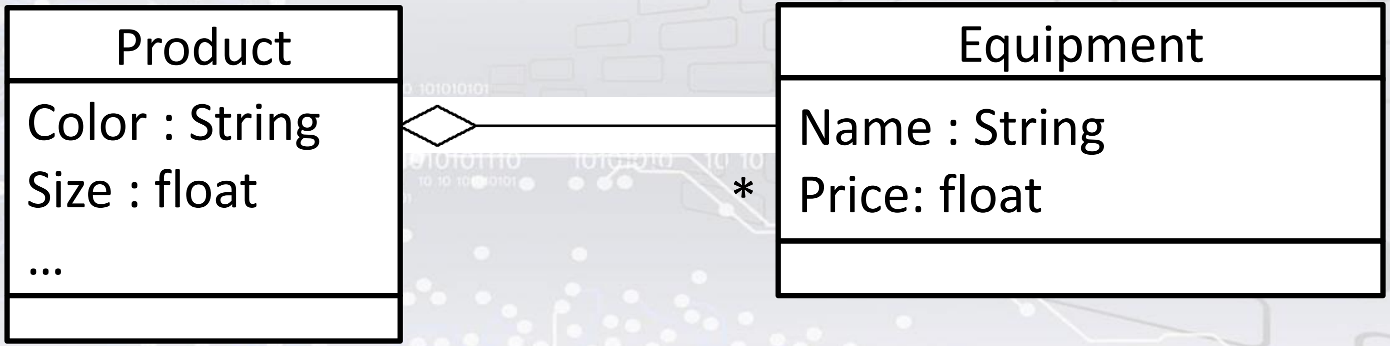
Partial aggregation – part may belong to many wholes



A white rhombus on the owner's side

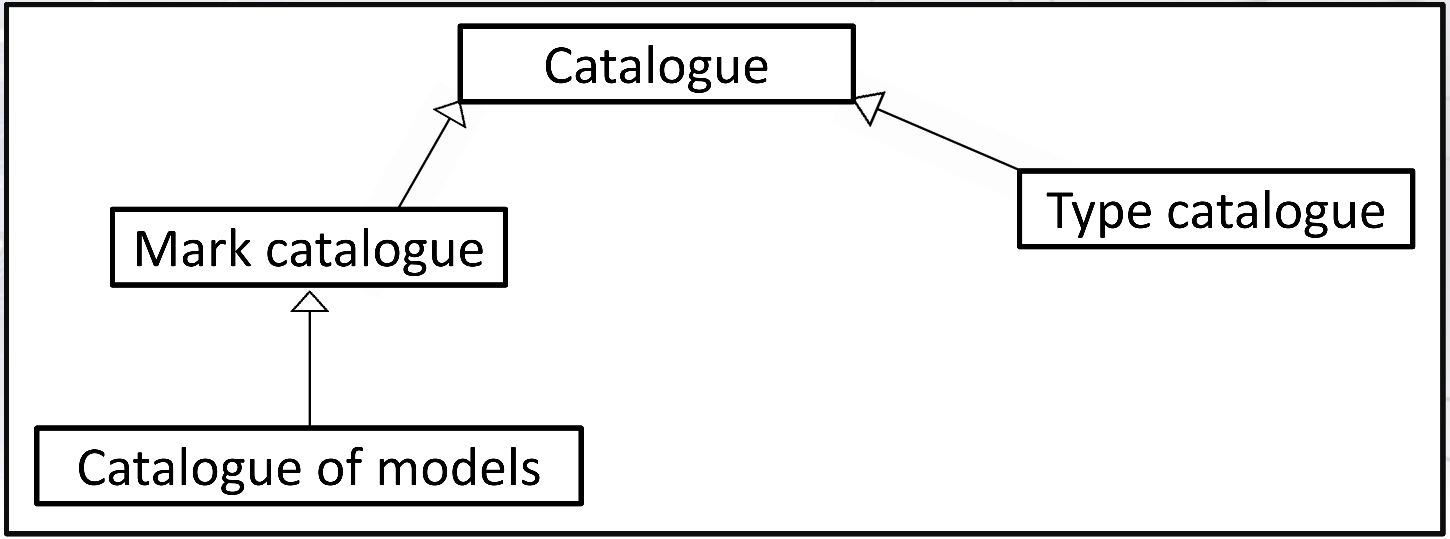
Relations

Complete aggregation – a part may belong to a whole which creates parts.



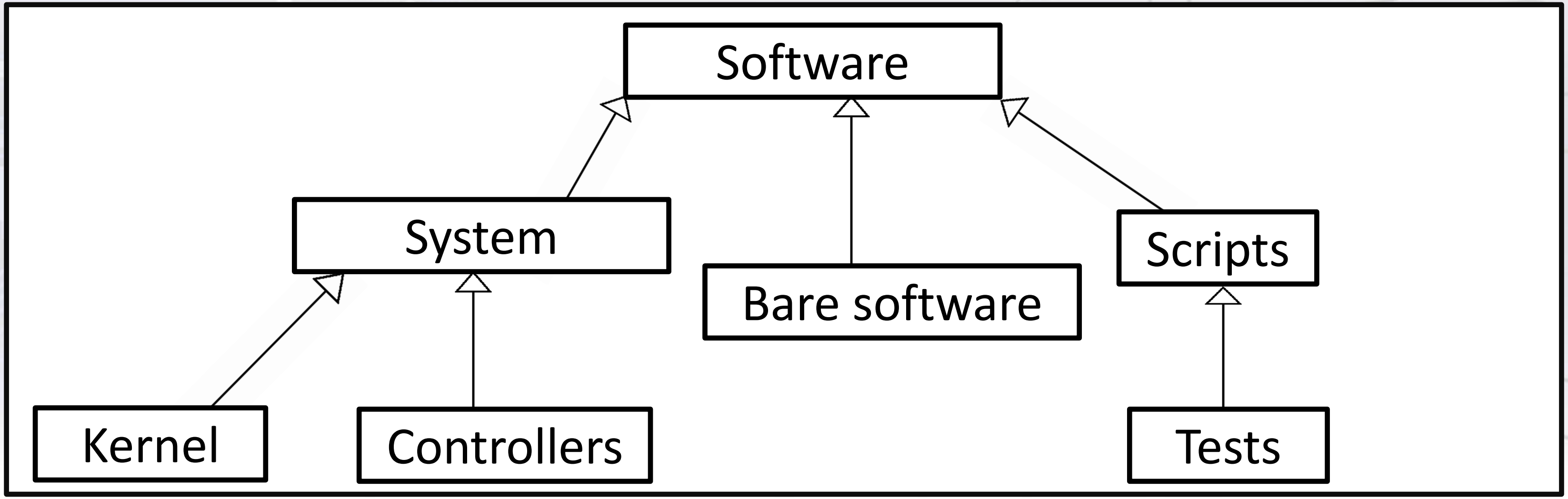
Relations

Inheritance – creates class hierarchy, general to specific



Relations

Multiple inheritance – a class inherits from more than one class



Relations

Abstract class:

- describes the common functionality of a group of classes – identifies common behavior of different classes
- notation

  class_name or {abstract} class_name
- cannot have objects
- must define subclasses because they cannot create their own instances
- methods are associated with an abstract class only through inheritance

Bibliography

[1] Grady Booch, James Rumbaugh, Ivar Jacobson, *Unified Modeling User Guide* (book PDF)

[1] Tom Pender: *UML Bible*. John Wiley & Sons, 2003.

[2] http://brasil.cel.agh.edu.pl/~09sbfraczek/uml-definicja-historia,1,54.html

[3] Jarosław Kuchta*, Unified Modeling Language (Foundation)*, 2015

[4] http://brasil.cel.agh.edu.pl/~09sbfraczek/diagram-klas,1,11.html#