Professional Software Engineering

Andrea Carrara and Patrick Berggold

Hritik Singh and Mohab Hassaan – Tutors

Chair of Computational Modeling and Simulation

andrea.carrara@tum.de; patrick.berggold@tum.de

Schedule Lecture 6

» UML

- Introduction
- Structural Diagrams
- Behavioral Diagrams

andrea.carrara@tum.de; patrick.berggold@tum.de

INTRODUCTION TO UML

andrea.carrara@tum.de; patrick.berggold@tum.de

Introduction

- Unified Modeling Language (UML) is a standardized, general-purpose modeling language
- Visualize code structures and data bases with graphs
- Contains many diagram types dedicated to specific purposes

andrea.carrara@tum.de; patrick.berggold@tum.de

History

- Structural analysis methods not suited for OOP
- No standard for teams to communicate

- UML 1.0 came around 1997 (quite new)
- UML 2.5 current version
- Since 2005:
  - ISO/IEC 19501:2005

- Have a look at the UML specification: https://www.omg.org/spec/UML/2.5/PDF

Who uses UML?

- Product owner
- Business analyst
- Architect
- Quality management
- Developer
- Operations

- UML helps to communicate between each role
- Works for all languages

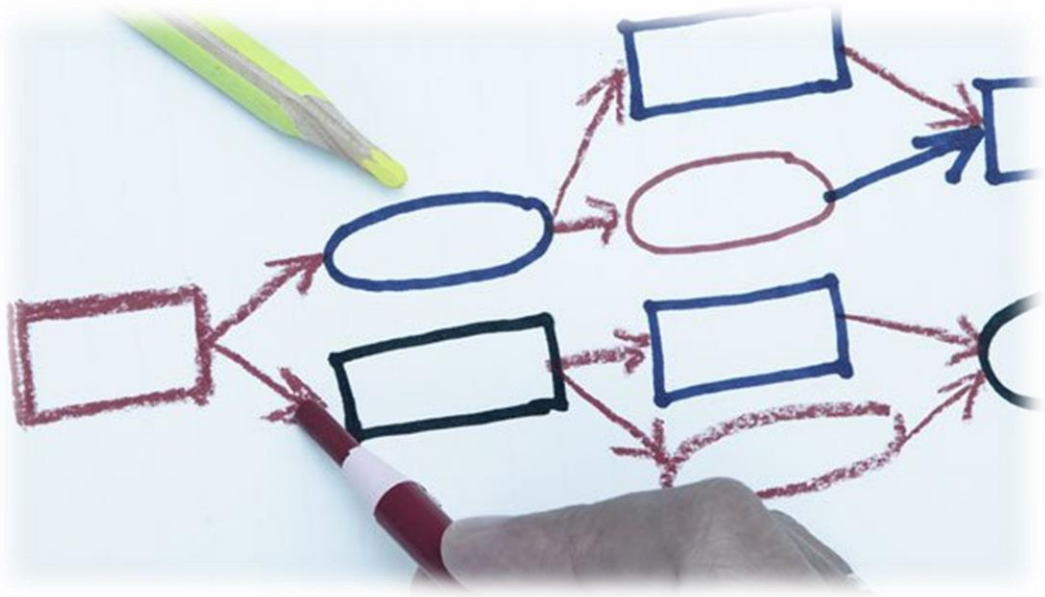
- Helps with documentation
- Precise expression

andrea.carrara@tum.de; patrick.berggold@tum.de

UML Diagrams

» Structural Modeling

- Properties
- Static parts
- Often Nouns
- Examples:
  class diagram, component diagram



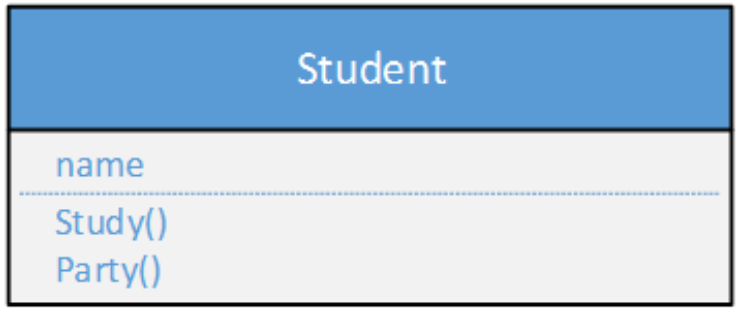
» Behavioral Modeling

- Dynamic processes
- Mostly verbs / actions
- Examples:
  activity diagram, use case diagram

andrea.carrara@tum.de; patrick.berggold@tum.de

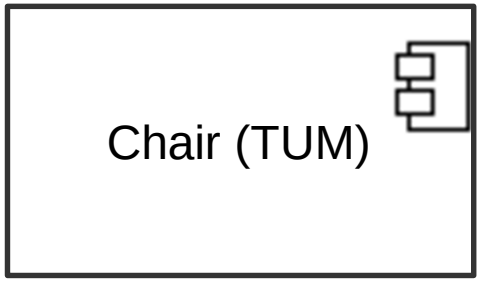
Basic UML Blocks

- Objects and structures



- Use case

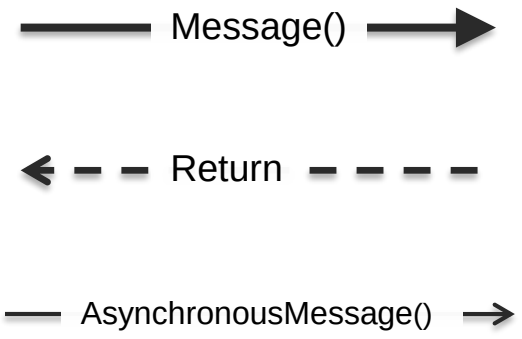
- Component



andrea.carrara@tum.de; patrick.berggold@tum.de

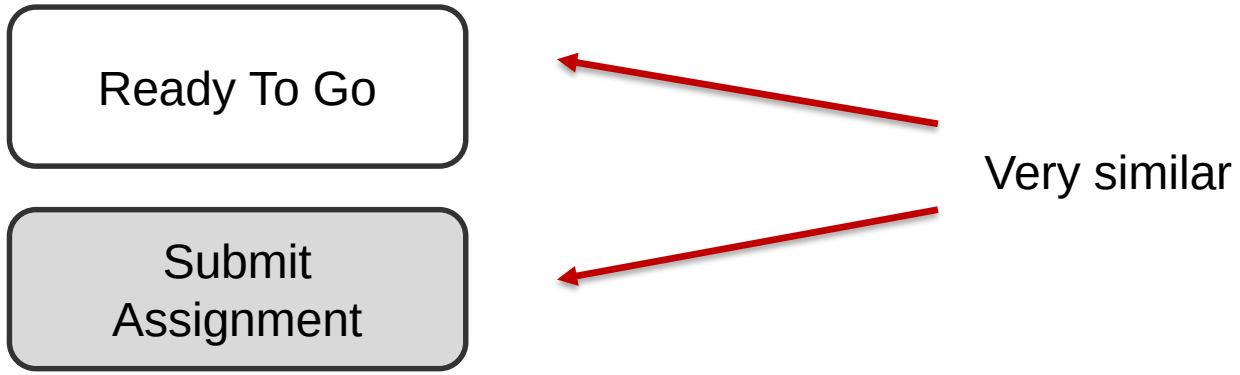
Messages and Actions

» Messages



» States

» Actions



andrea.carrara@tum.de; patrick.berggold@tum.de

Relationships

- Generalization 
  - Arrow points to parent (*inheritance*)
- Implementation 
  - Arrow points to definition (*implementation*)
    - **end points to implementation**
- Dependency 
  - Arrow points to relier

andrea.carrara@tum.de; patrick.berggold@tum.de

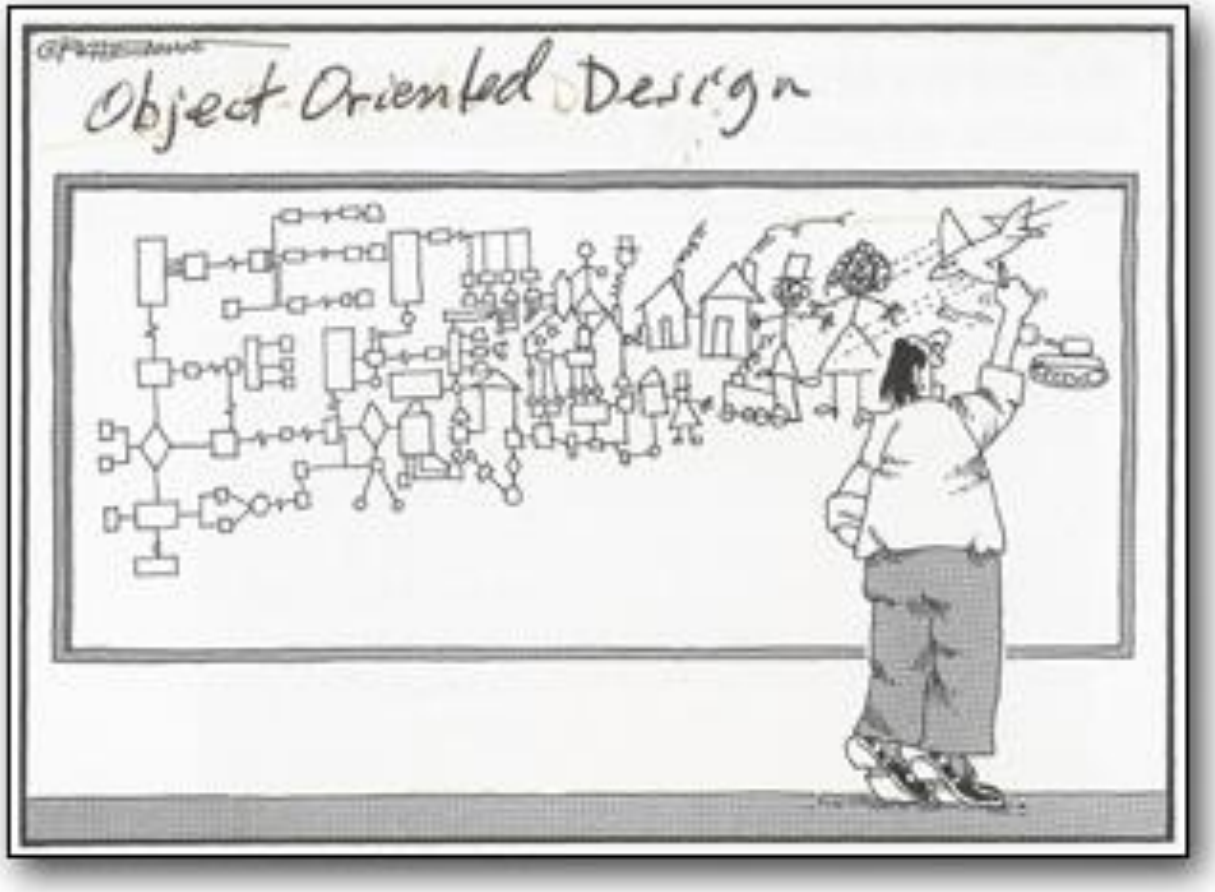
About Diagrams

- Readable
  - Try to avoid line crossing
- Scope
  - Use a small scope
  - Focus on the important parts
- Diagrams are a visualization
- They document the system

➡ **Who will be the reader of this diagram?**

andrea.carrara@tum.de; patrick.berggold@tum.de

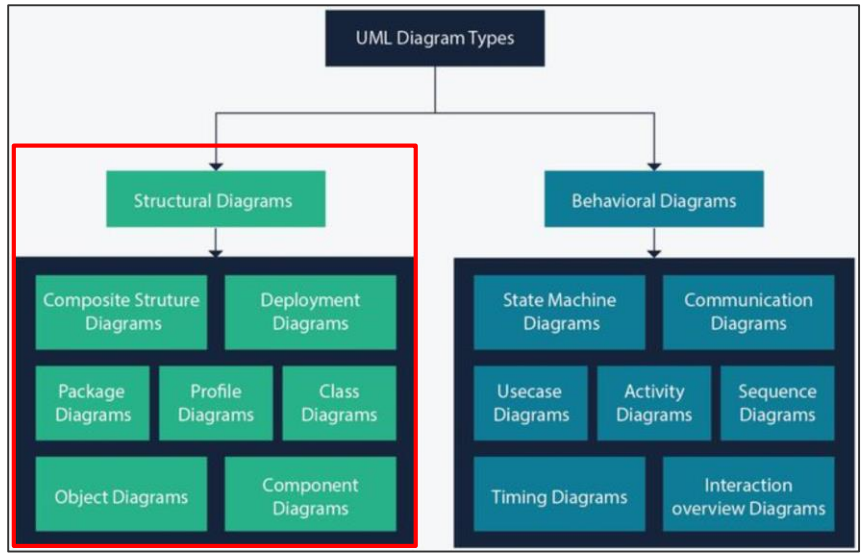
STRUCTURAL DIAGRAMS



andrea.carrara@tum.de; patrick.berggold@tum.de

Structural Diagrams

- Depict the inner structure and relationships of a system
- System can practically anything, e.g. software architecture, real-life objects, organizations, …
- Static view, independent of time
- In the UML specification there are 7 structural diagrams

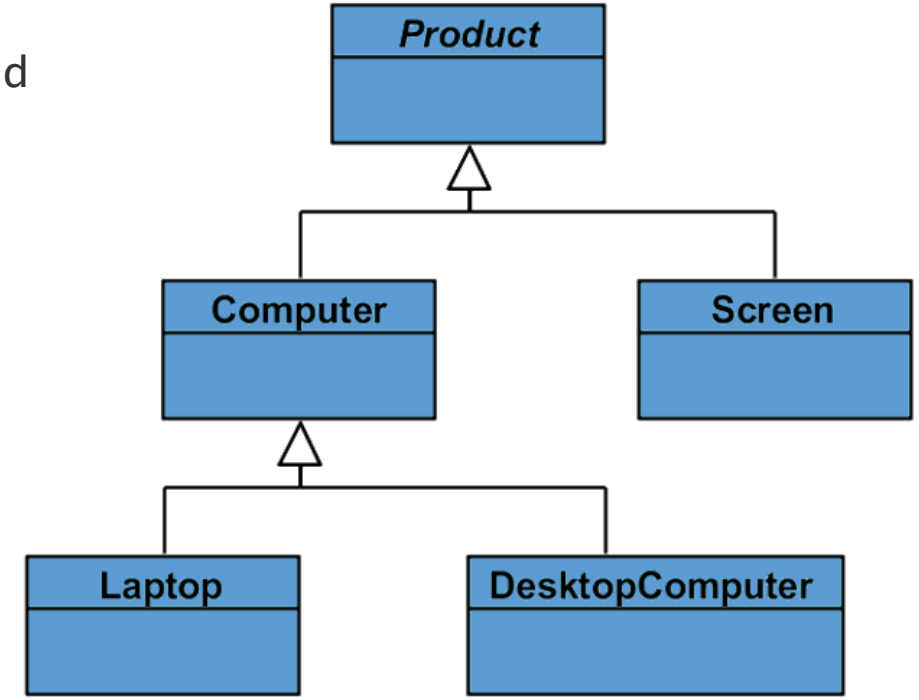


Most prominent type is the **class diagram**

andrea.carrara@tum.de; patrick.berggold@tum.de

Class Diagram

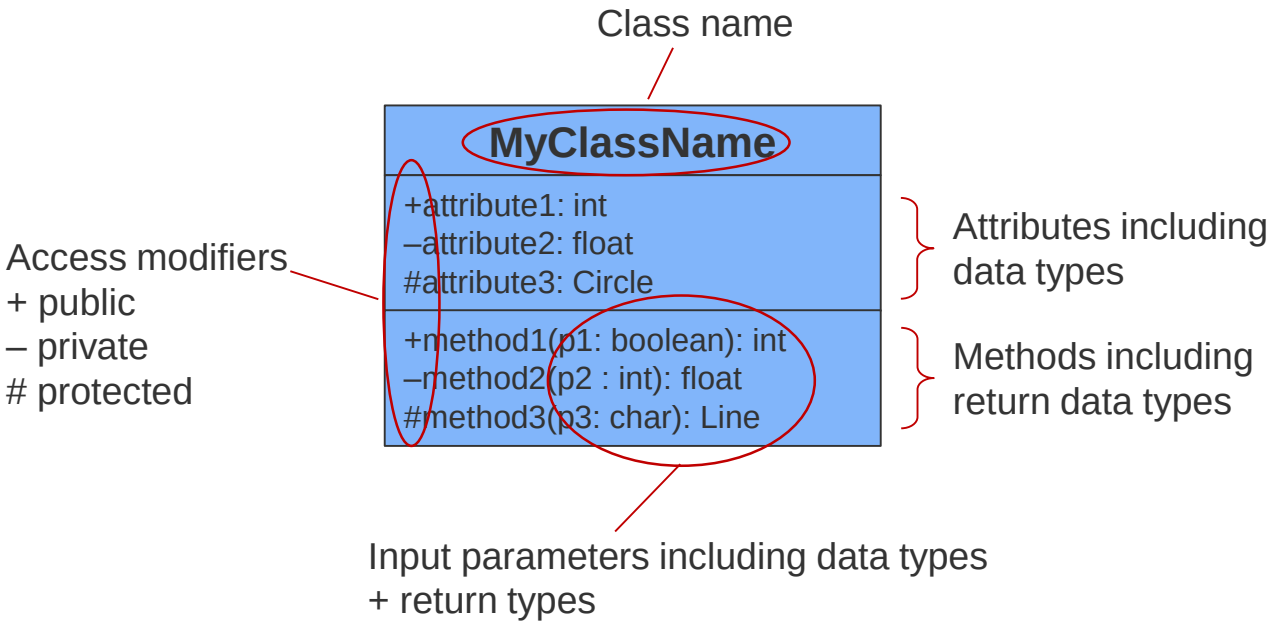
- Shows relationships between classes (OOP)
- Static model describing what exists and which behavior and attributes are implemented
- Mostly applicable to show software as it contains implementation details

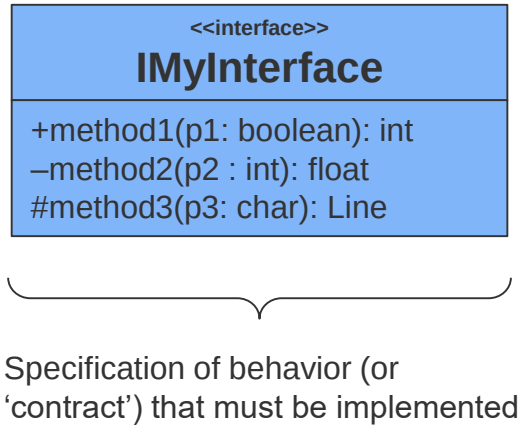


andrea.carrara@tum.de; patrick.berggold@tum.de

Class Diagram – basic blocks

» Basic building blocks: classes and interfaces

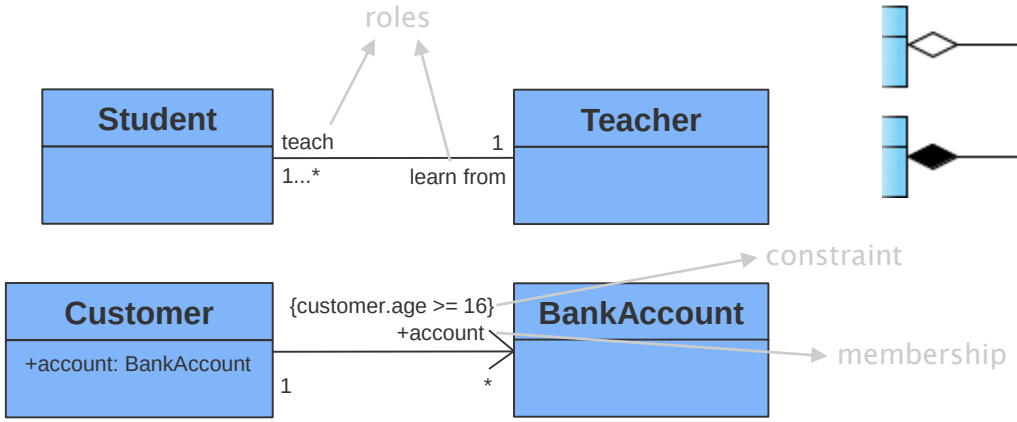


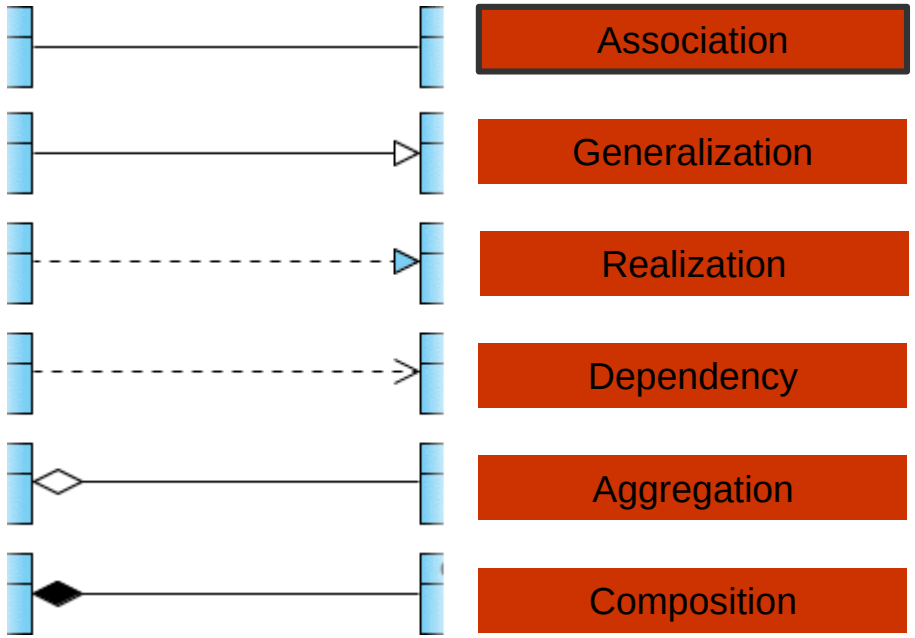


» Further elements: Enumeration, DataType and others

Class Diagram – relationships: association

- Classes could be either linked to each other or combined logically or physically
- Associations may include directions, roles and constraints ({...}).
- Number of instances indicated by cardinality.
- Directed associations can be used to imply membership.
- Cardinalities:



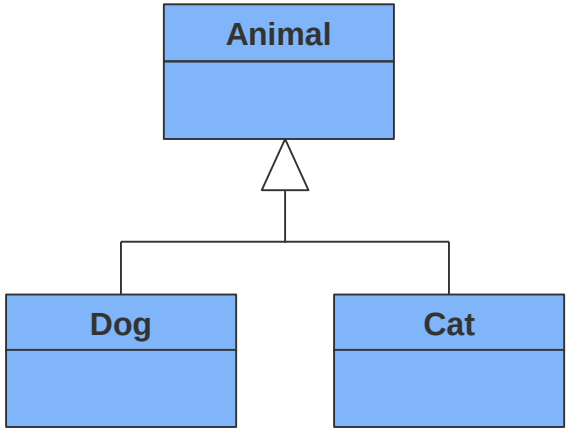


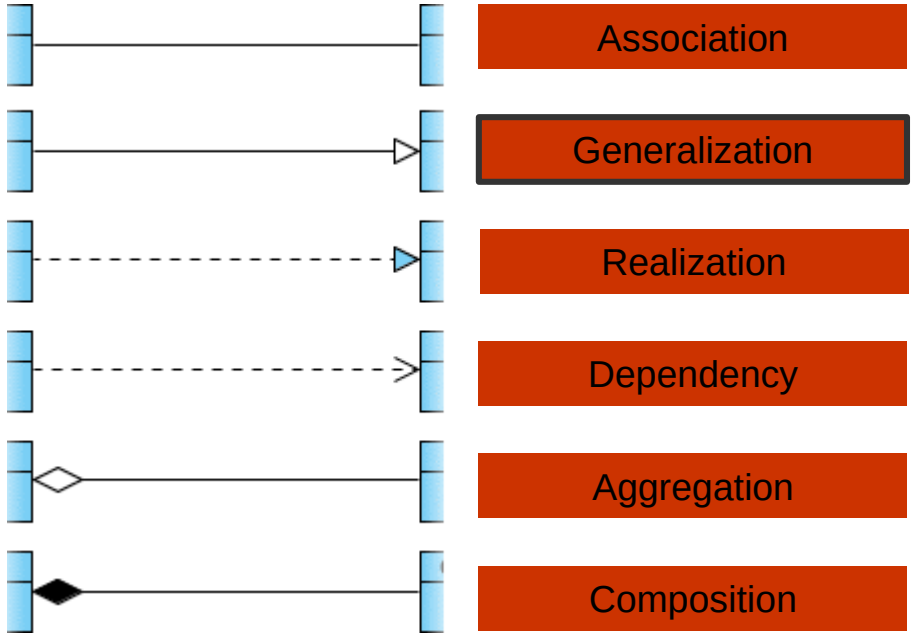
andrea.carrara@tum.de; patrick.berggold@tum.de

Class Diagram – relationships: generalization

» A child class (left) is derived from a parent class (right), thus the parent's features are inherited.

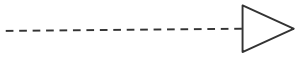
» Implements inheritance!

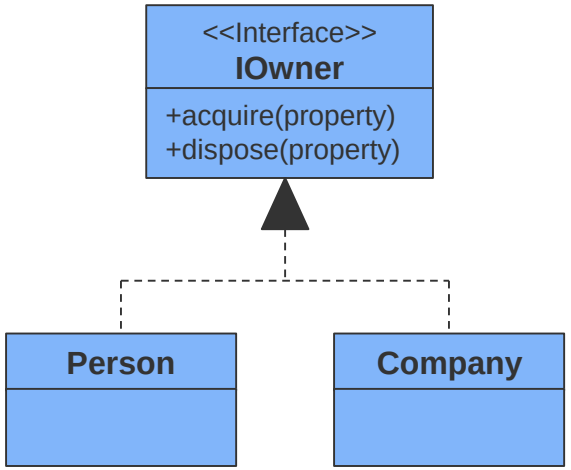


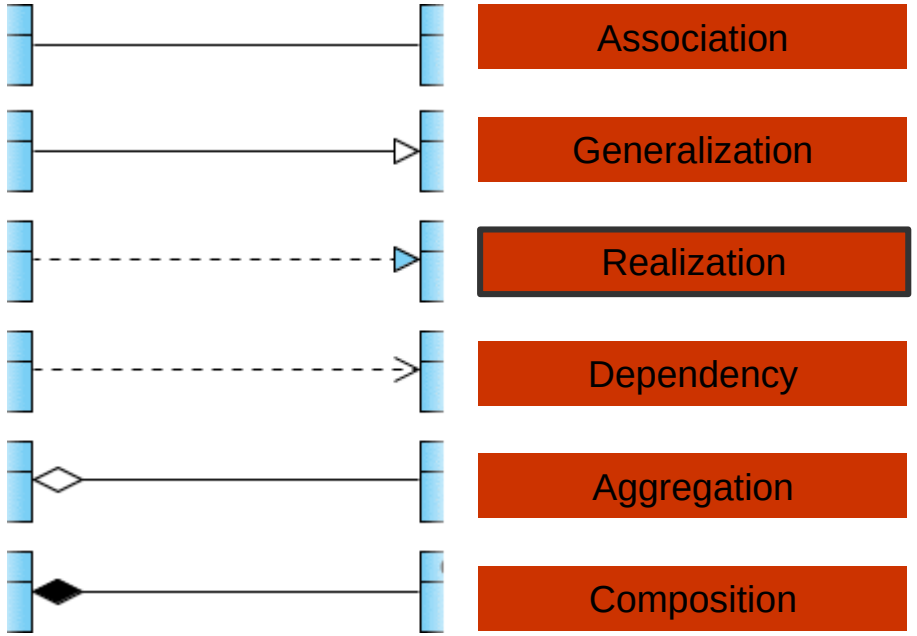


andrea.carrara@tum.de; patrick.berggold@tum.de

Class Diagram – relationships: realization

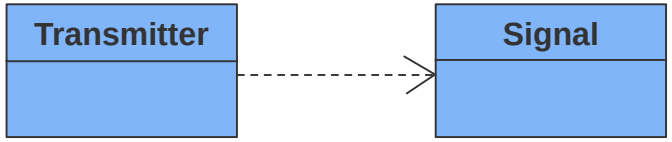
- Realization between two classes in which one class must implement (or realize) the behavior of the other (called blueprint class).
- Other common arrow: 
- Implements interfaces!

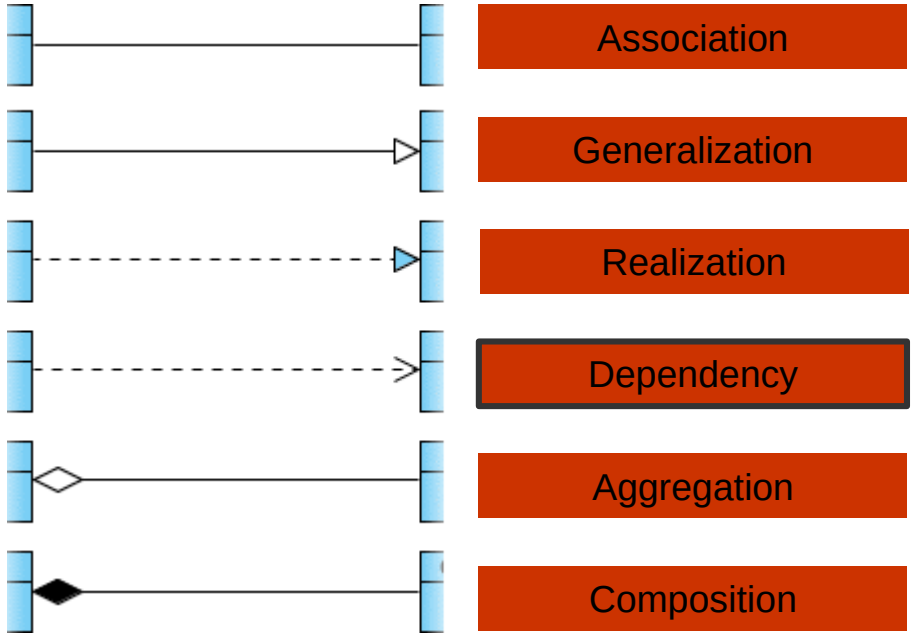




Class Diagram – relationships: dependency

- At least two elements are dependent on each other.
- This means that changes in one element automatically affect the other.
- Many different types of dependencies



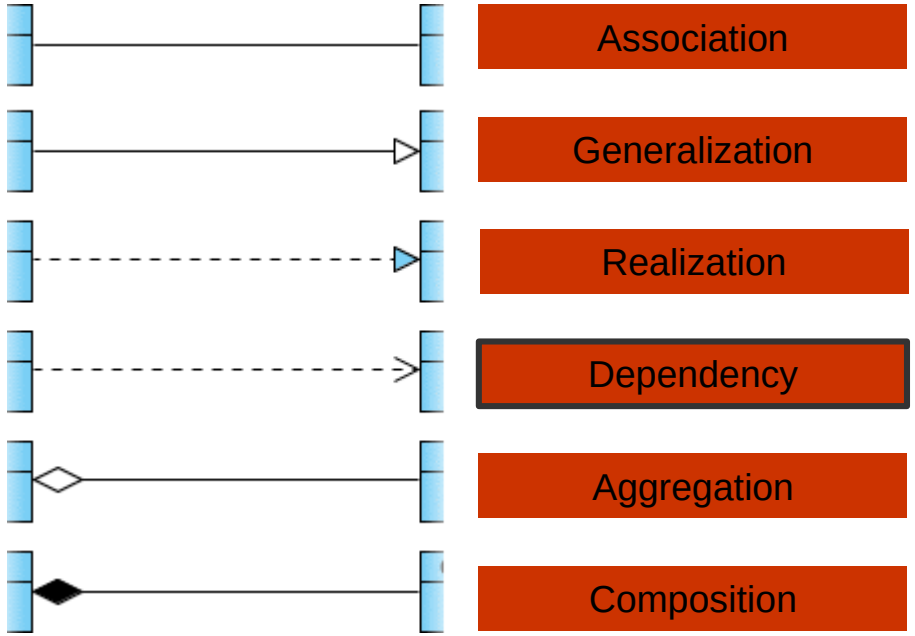


andrea.carrara@tum.de; patrick.berggold@tum.de

Class Diagram – relationships: dependency

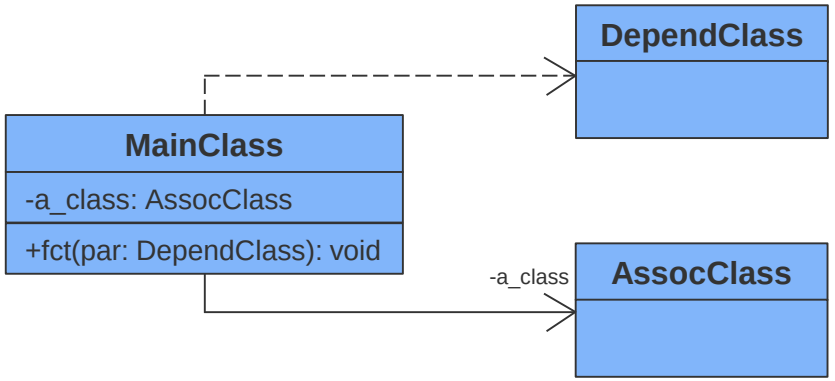
» A dependency is a very broad definition, here a common examples:

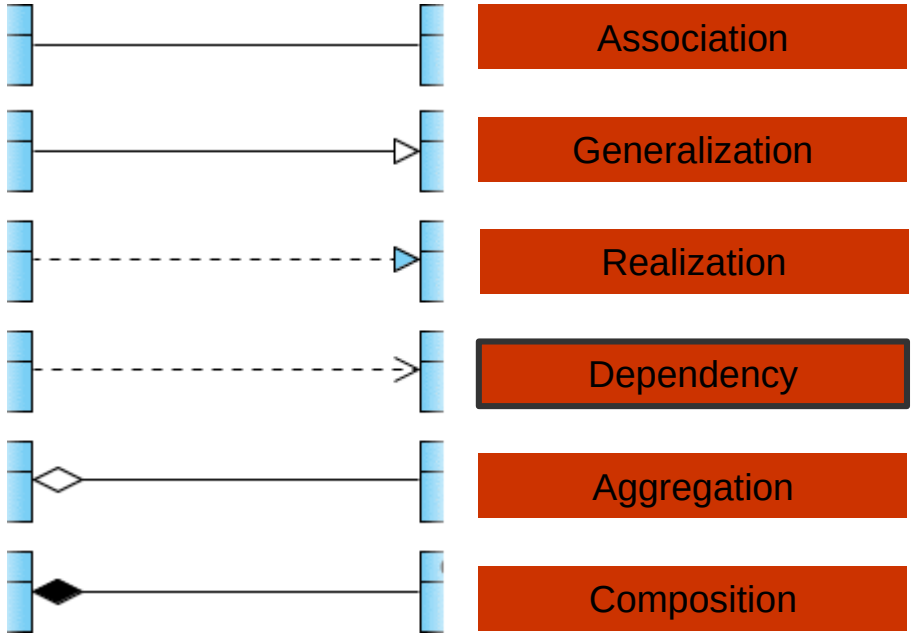
<<access>>
<<derive>>
<<friend>>
<<refine>>
<<use>>
<<substitute>>
<<import>>
<<permit>>
<<extend>>
<<include>>
<<become>>
<<call>>
<<copy>>
<<parameter>>
<<send>>



Class Diagram – relationships: dependency vs. association

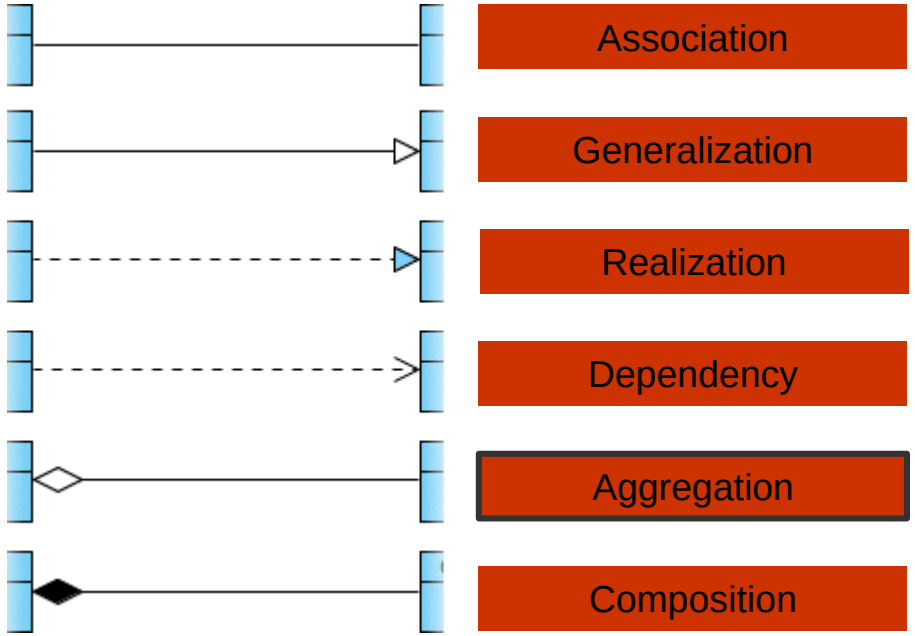
- Common question: What's the difference between association and dependency?
- Associations are commonly used to model membership or access
- Dependencies are used when changes in one class affect its dependent classes
- In terms of OOP, this is a typical setting:





Class Diagram – relationships: aggregation

- Implies optional, maybe shared ownership
- All objects <u>can</u> exist independently of each other
- Binary association

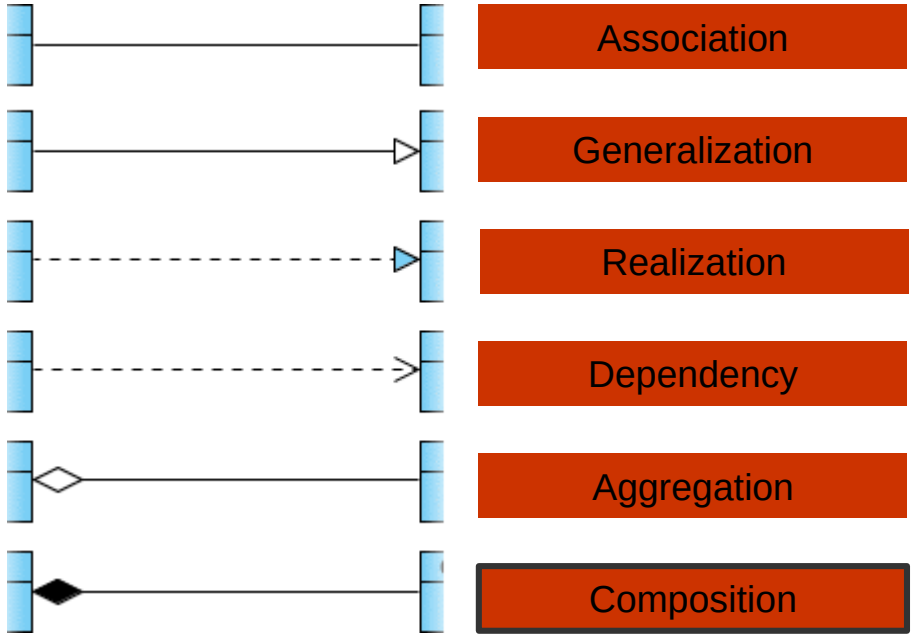


andrea.carrara@tum.de; patrick.berggold@tum.de

Class Diagram – relationships: composition

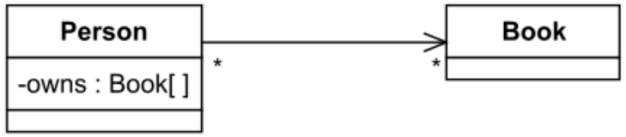
- Implies essential, strictly exclusive ownership
- Objects <u>cannot</u> exist independently from each other. If one is deleted, the other one is deleted, too.

- Binary association



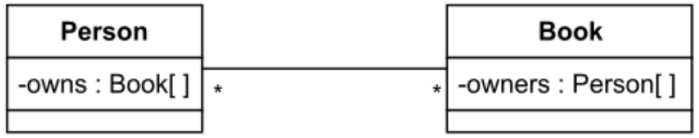
It all depends on what you want to model... Part 1

» Unidirectional association



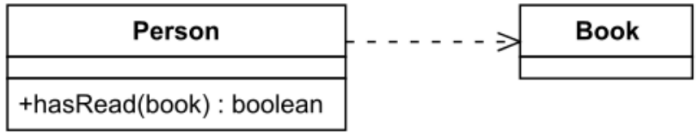
Person instances owns book instances in a private field as array of type Book.

» Bidirectional association



Person owns several books, and a Book instance lists all the people that own it.

» Dependency

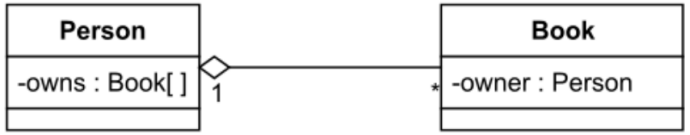


If a person has read a certain book already, it might influence an internal Book variable.

from http://www.cs.utsa.edu/~cs3443/uml/uml.html

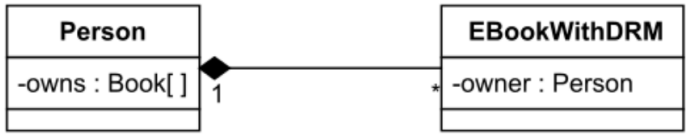
It all depends on what you want to model... Part 2

» Aggregation



A Person instance owns several books as fields, but can exist on its own without them.

» Composition



A Person instance owns several personalized e-books as fields. Hence, a person cannot sell the their e-books, meaning that there lifetimes are equal.

from http://www.cs.utsa.edu/~cs3443/uml/uml.html

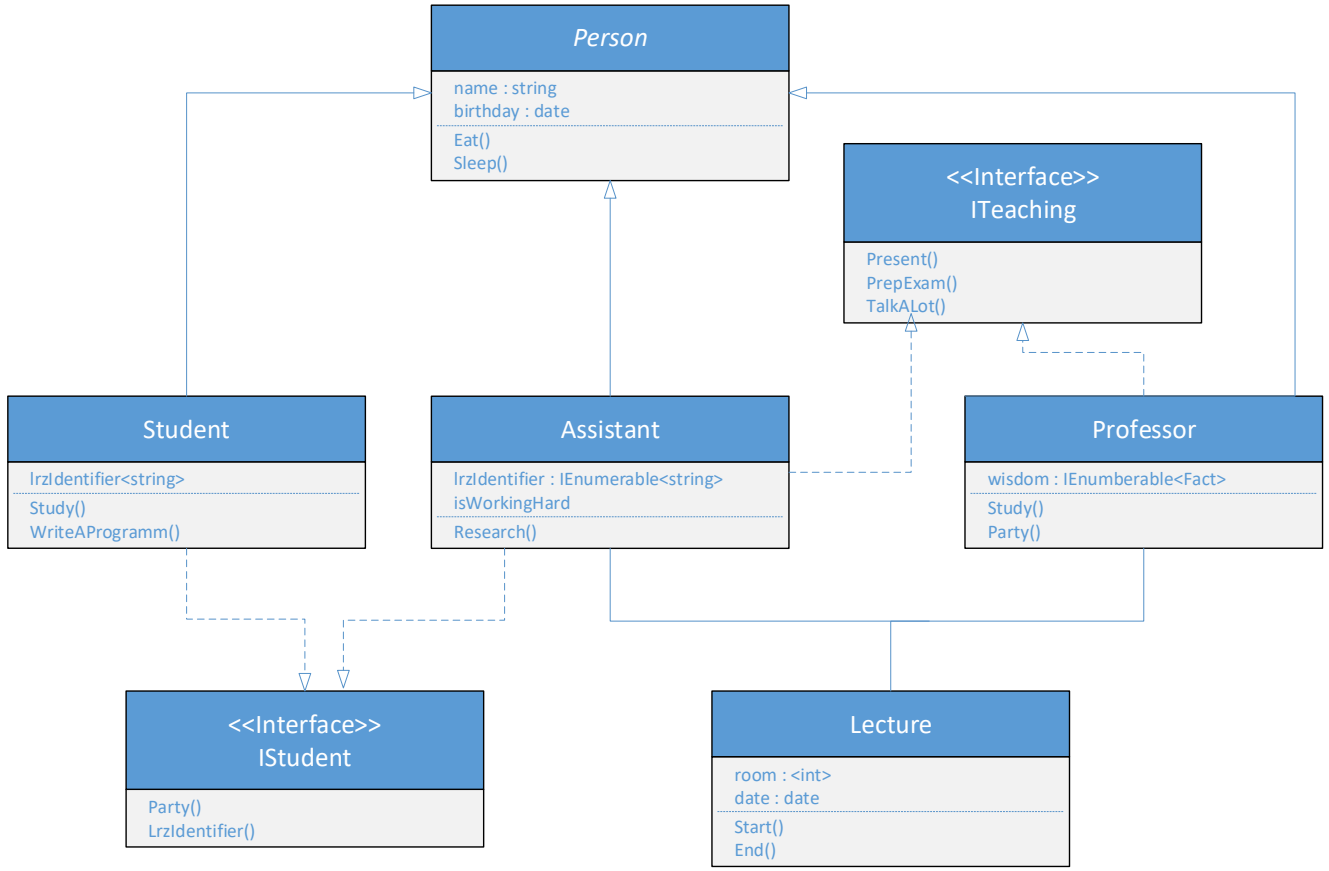
Why is it important?

- Based upon principles of OOP
- Can be used in all phases of the development
  - **During analysis => domain model**
  - **During design => blueprint of software**
  - **During Implementation => basic code structure**

- Review your design
  - **Are the interfaces designed properly?**
  - **Aggregation or Inheritance ?**
- Class diagram concepts (e.g. association, dependency, generalization) can be transferred to other UML diagrams

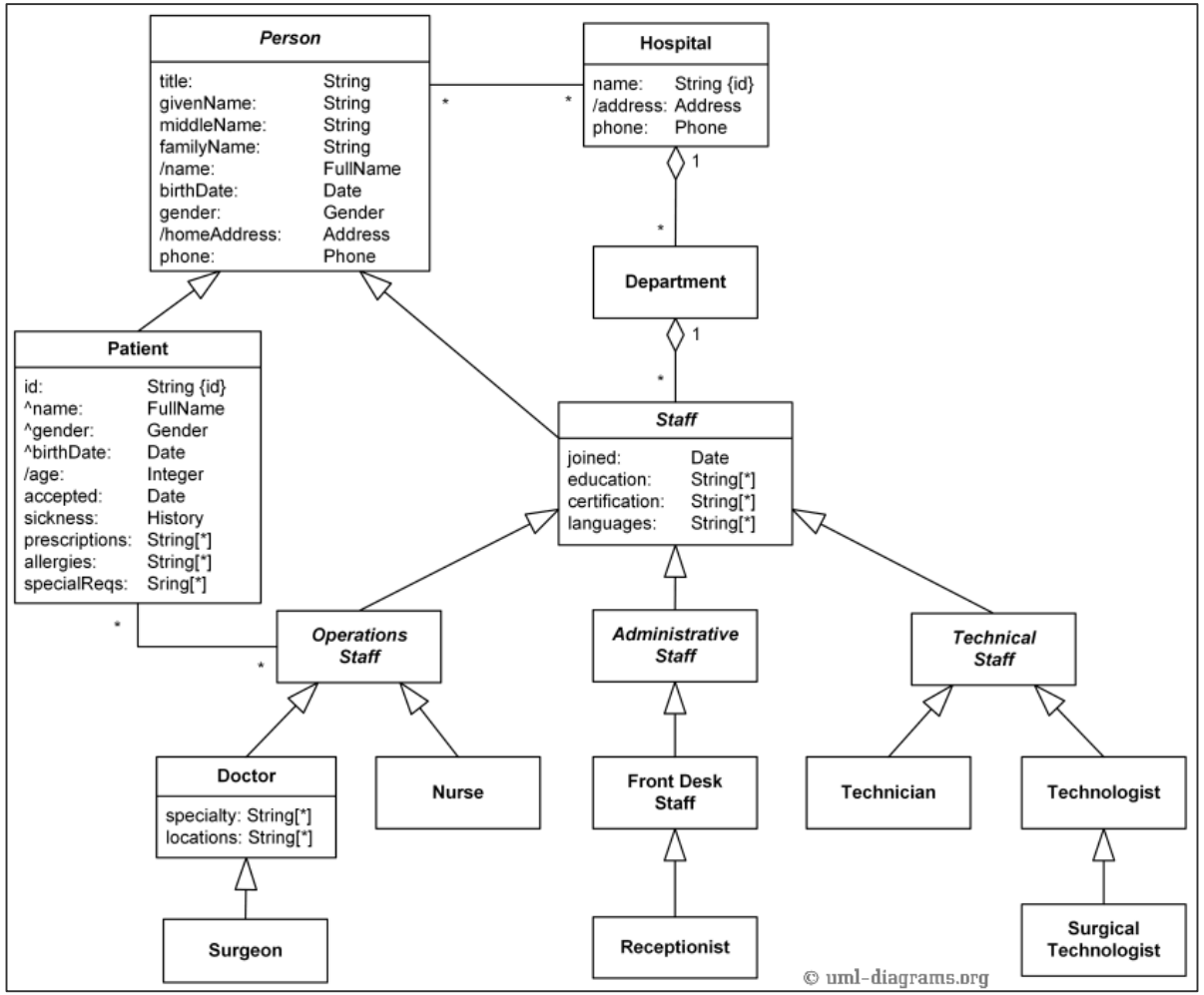
andrea.carrara@tum.de; patrick.berggold@tum.de

Basic class diagrams



Basic class diagrams

Hospital management diagram:



https://www.uml-diagrams.org/examples/hospital-domain-diagram.html?context=cls-examples

Few additional features:

Derived attribute (/):
computed from other information, for example other properties

Inherited attribute (^):
inherited from parent class

Component Diagrams

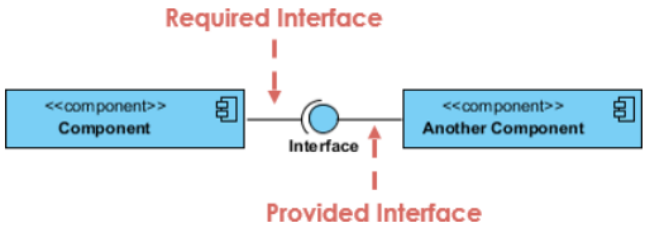
- Typically used for **components in software and hardware** (although general application)
- Visualize high-level structure of a system
- Explain service behavior provided by components and parts through interfaces and ports
- Compared to class diagrams, component diagrams …
  - don't show the actual implementation
  - visualize components which are usually composed of several or many classes (➡ higher level abstractions)
  - typically illustrate an overall use case

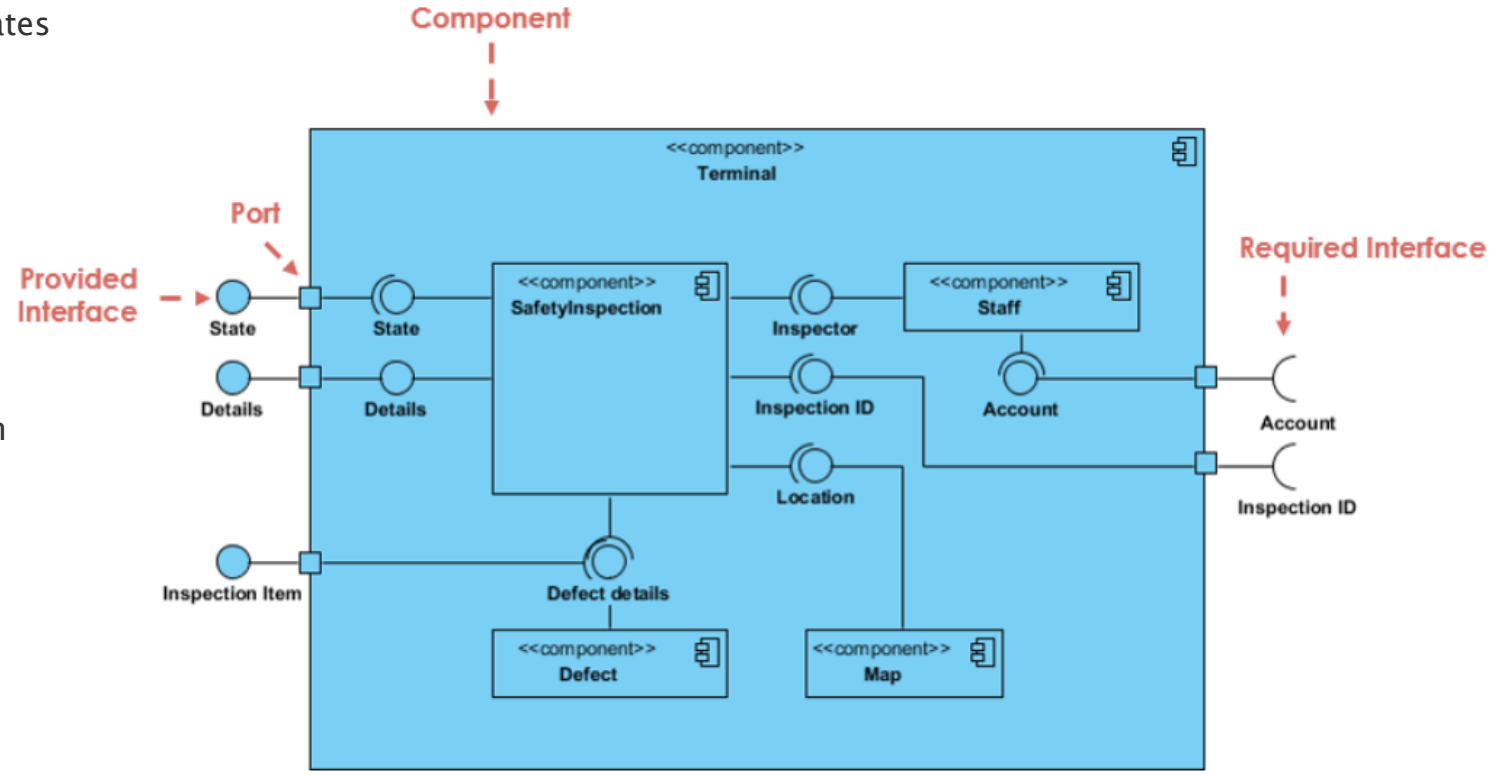
➡ Component diagrams (typically) operate on a higher level focusing on the main services

andrea.carrara@tum.de; patrick.berggold@tum.de

Component Diagrams

- **Component**
  Reusable piece with some functionality that communicates through interfaces
- **Provided Interface**
  Collection of several methods and attributes that a component provides
- **Required Interface**
  Collection of several methods and attributes that a component requires
- **Port**
  Feature of a component specifying a distinct interaction point between component and environment





Good overview: https://learn.microsoft.com/de-de/previous-versions/visualstudio/visual-studio-2015/modeling/uml-component-diagrams-reference?view=vs-2015&redirectedfrom=MSDN

andrea.carrara@tum.de; patrick.berggold@tum.de

Other Structural Diagrams

- Composite structure diagram

  Specializes on the **internal collaboration** of classes, interfaces and components (class diagram in more detail)

- Deployment Diagram

  Specializes on the **how and where** a system is deployed w.r.t. its hardware and the corresponding software

- Object diagram

  Specializes on the relationships and collaboration of **class instances**

- Package diagram

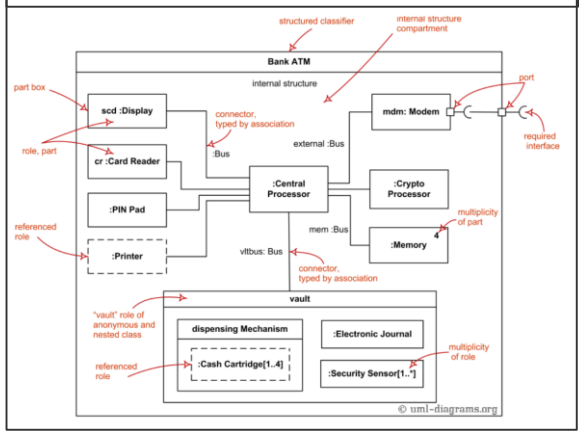
  Specializes on **organizing functional units** (e.g. namespaces) into packages and show the dependencies between packages

- Profile diagram

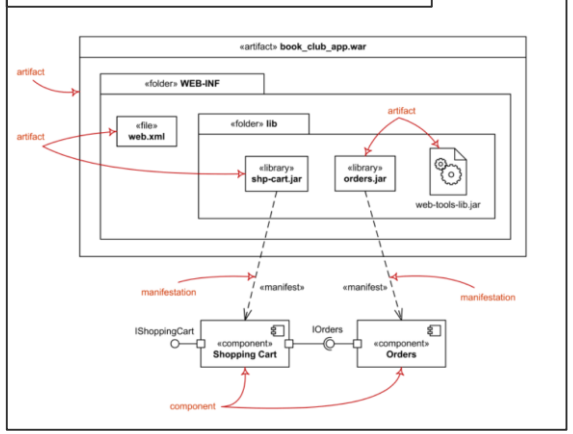
  Provides an extension mechanism for **customizing UML** models for particular domains, allowing the refinement of standard semantics (e.g. customize UML for aerospace, healthcare, financial, etc.)

Other Structural Diagrams

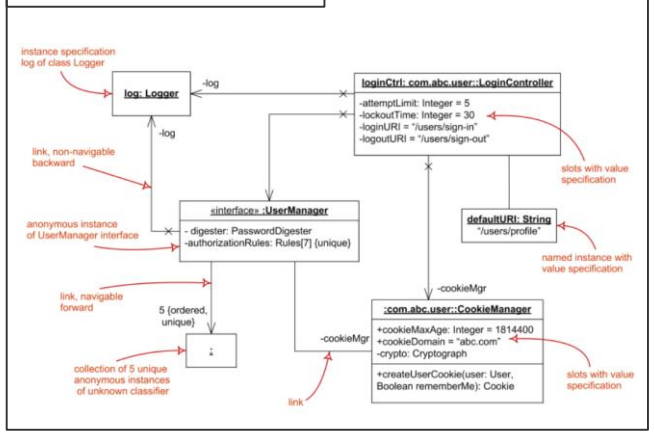
Composite structure diagram



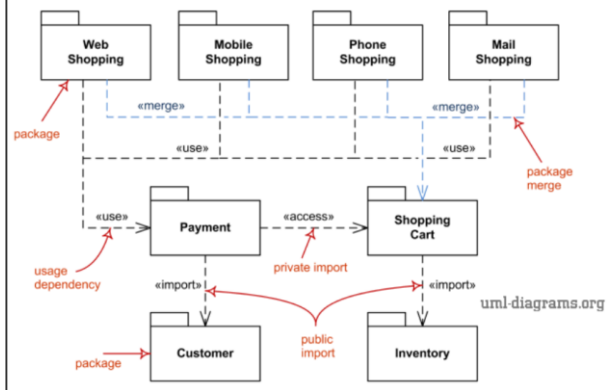
Deployment diagram



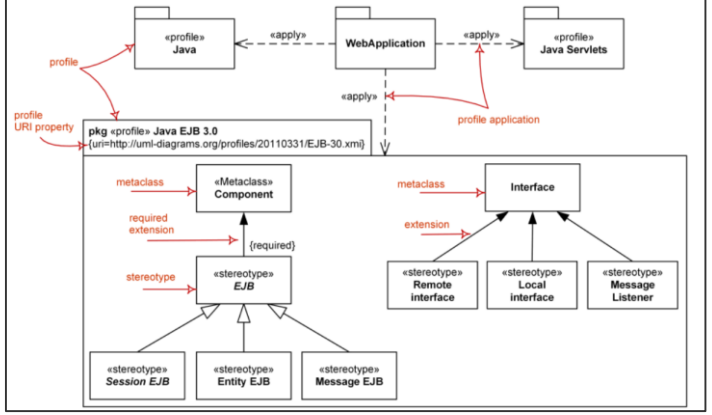
Object diagram



Package diagram



Profile diagram



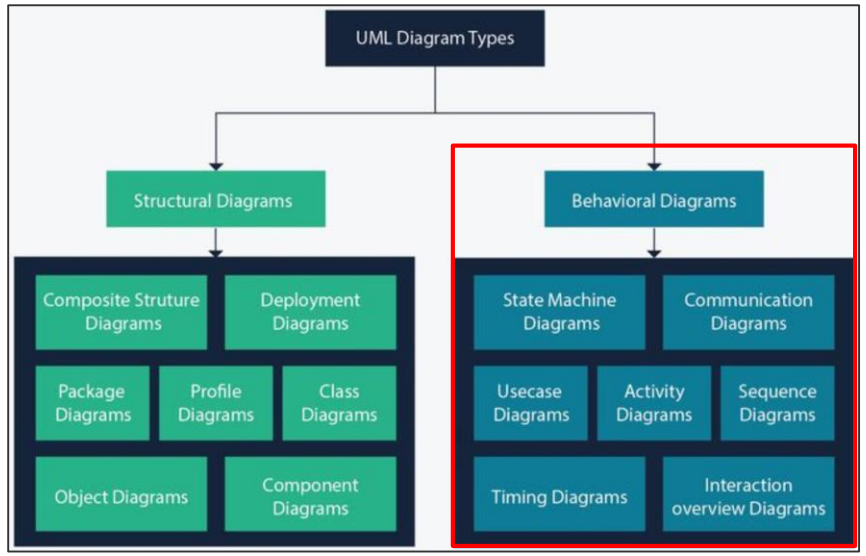
https://www.uml-diagrams.org/uml-24-diagrams.html

BEHAVIORAL DIAGRAMS

andrea.carrara@tum.de; patrick.berggold@tum.de

Behavioral Diagrams

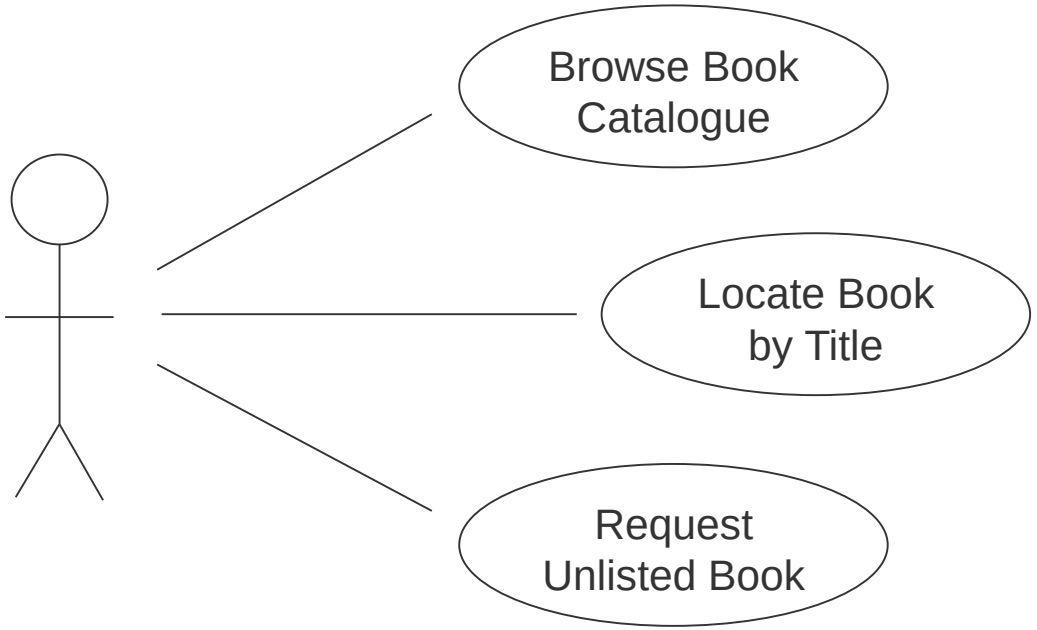
- Depict the dynamic behavior and interactions of system elements along time
- Dynamic view, dependent on time
- In the UML specification there are 7 behavioral diagrams



andrea.carrara@tum.de; patrick.berggold@tum.de

Use Case Diagram

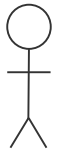
- Describe functional requirements (use cases) of a system
- Describe the dynamic interaction between external actors and system boundary
- Describe the response of a system to user input
- It does not (!) show the order in which steps are performed



andrea.carrara@tum.de; patrick.berggold@tum.de

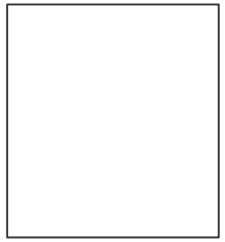
Use Case Diagram Elements

» Actor – user of the system (e.g. human, machine, other system)



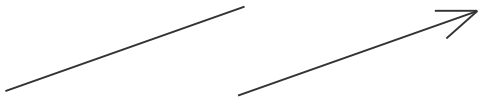
» Use case – describes how the actor interacts with the system

» System boundary – element defines the system's conceptual boundary



Use Case Diagram Connectors

- Association – implies a relationship between two elements



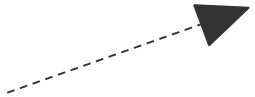
- Dependency – changes in one element automatically affect dependent elements

  <<include>> inclusion of functionalities

  <<extend>> extension of functionalities

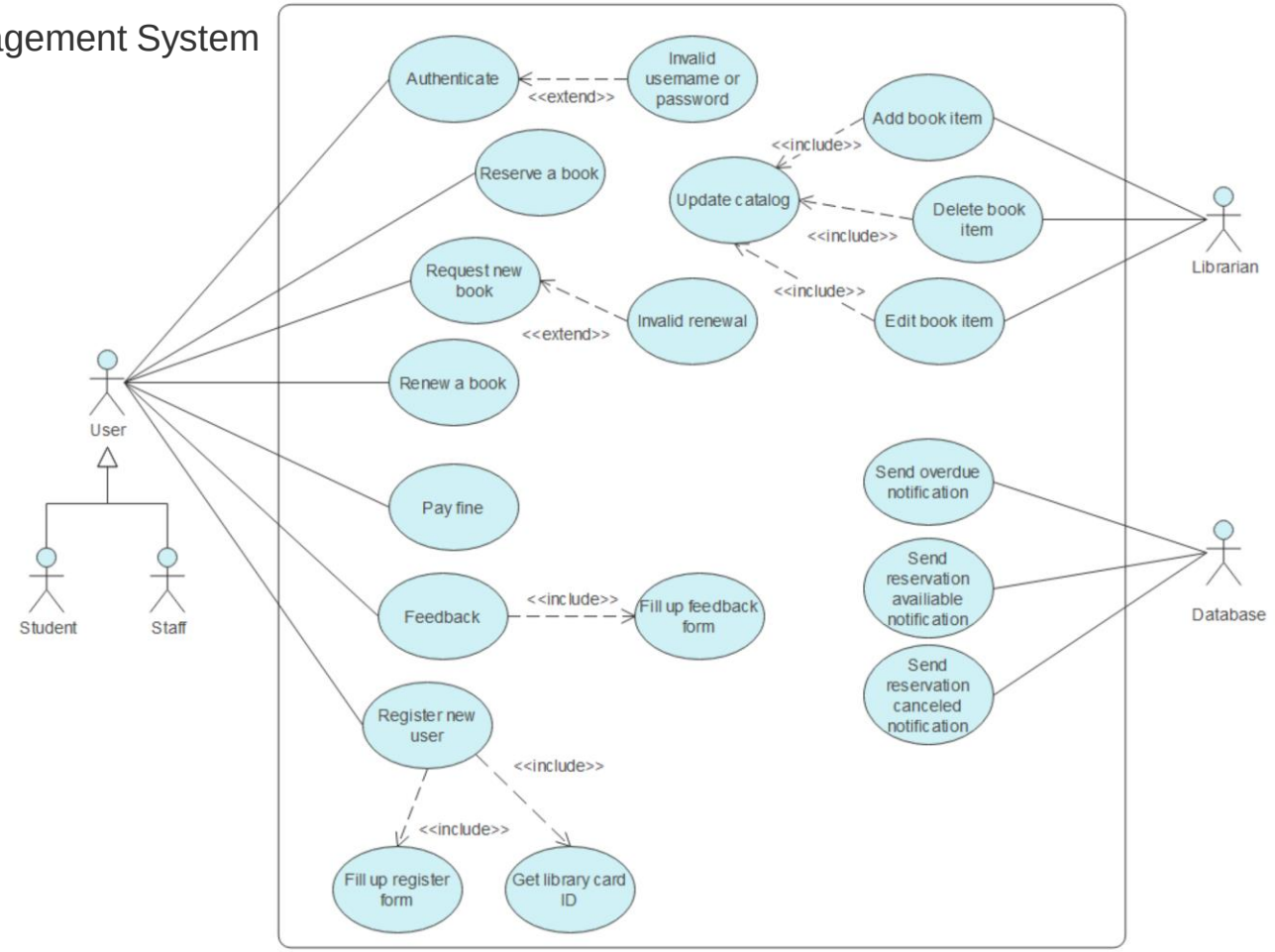
- Generalization – implies that one element is a specialization of the another

- Realization – one element implements the functionalities of another



Use Case Diagram Example

Library Management System

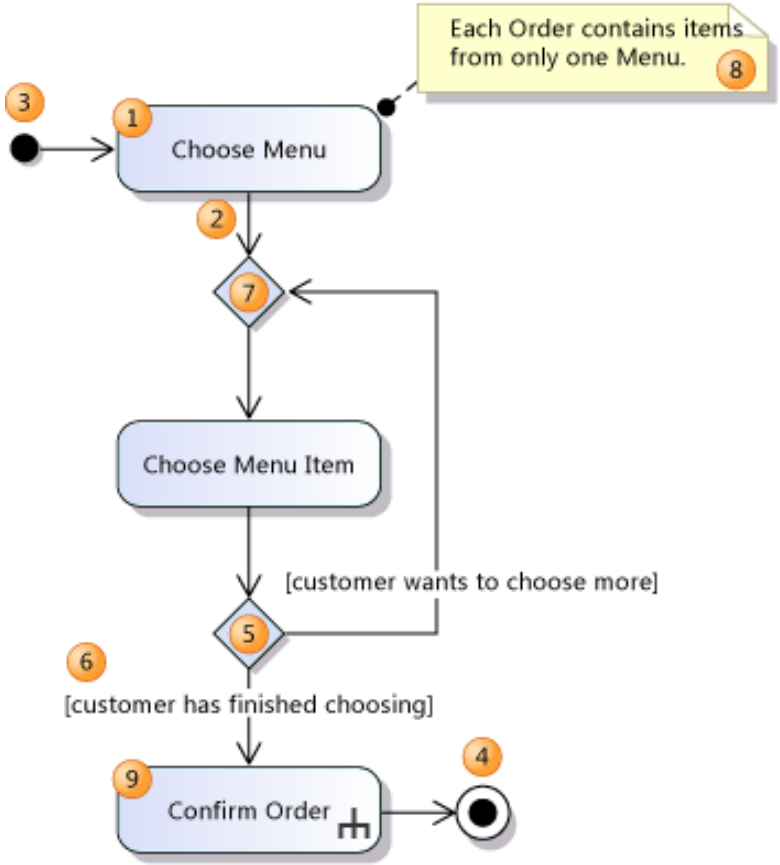


https://www.edrawsoft.com/article/use-case-diagram-examples.html

Activity Diagram

- Describe the sequential order in which steps are performed
- The entire workflow is shown, from start to end point
- Typically uses high-level abstractions to describe activities

- Thus, the logic and the sequential behavior of an algorithm or a process is described

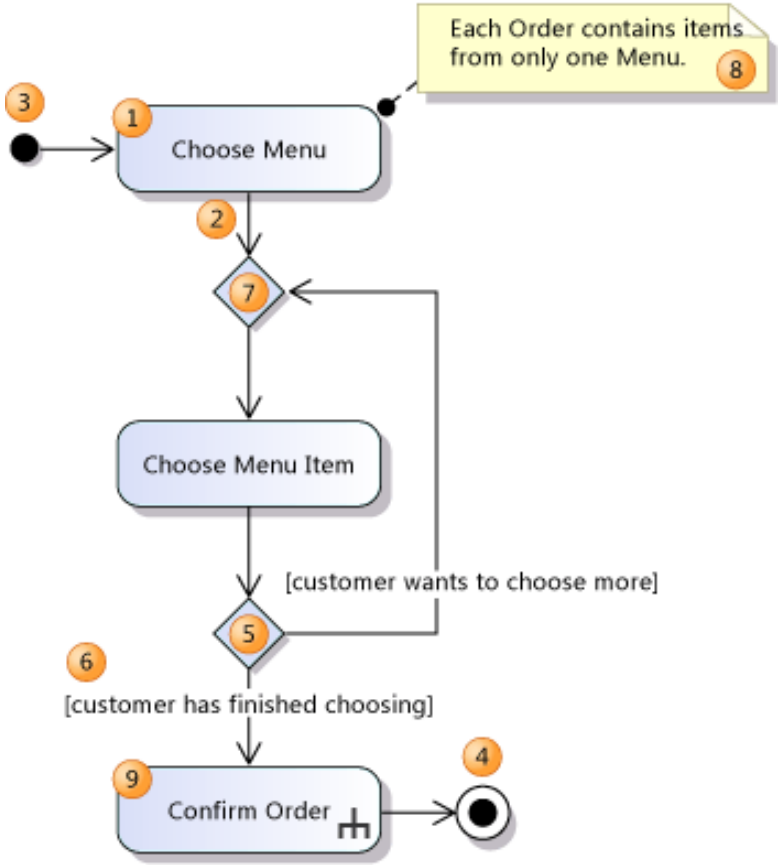


andrea.carrara@tum.de; patrick.berggold@tum.de

Activity Diagram

Imagine that a token (or thread of control) passes along the connectors from one action to the next…

1. **Action**
   Performance of a task of some kind (typically by the user or some software)
2. **Control Flow**
   Connector that shows the flow control between actions
3. **Initial Node**
   Indicates the first action (e.g. the first token flows from initial node to actions)
4. **Activity Final Node**
   Indicates the end of the activity
5. **Decision Node**
   Conditional branch with one input and several outputs
6. **Guard**
   Condition specifying whether a token can follow along a collector (usually used after a decision node)
7. **Merge Node**
   Merge flows with at least two inputs and one output
8. **Comment**
   Always helpful!
9. **Call Behaviour Action**
   Action that is defined in more detail in another diagram



andrea.carrara@tum.de; patrick.berggold@tum.de

Other Behavioral Diagrams

- State Diagrams

  Used to describe the **evolution of states** in which a system can be

- Timing Diagrams

  Describe **changes of internal states** of elements over time based on their exchanged messages

- Sequence Diagrams

  Show interactions between elements **emphasizing the time and order** of events

- Communication Diagrams

  Show interactions between elements **emphasizing the messages** sent between objects

- Interaction Overview Diagrams

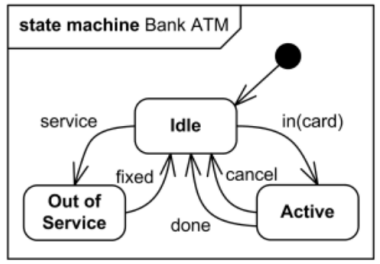
  Illustrate a control flow by visualizing the **cooperation between interaction diagrams**

also called **interaction diagrams**

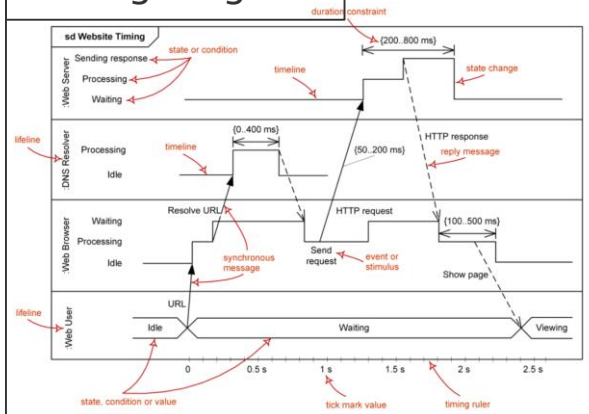
andrea.carrara@tum.de; patrick.berggold@tum.de

Other Behavioral Diagrams

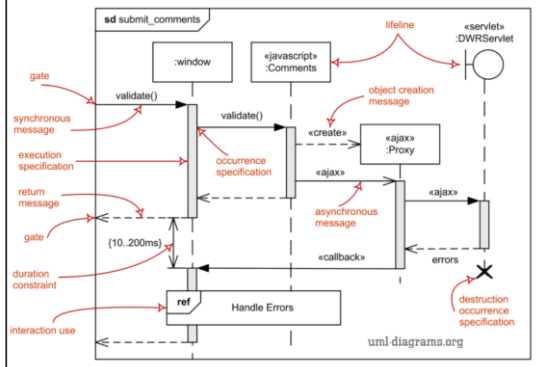
State Machine Diagram



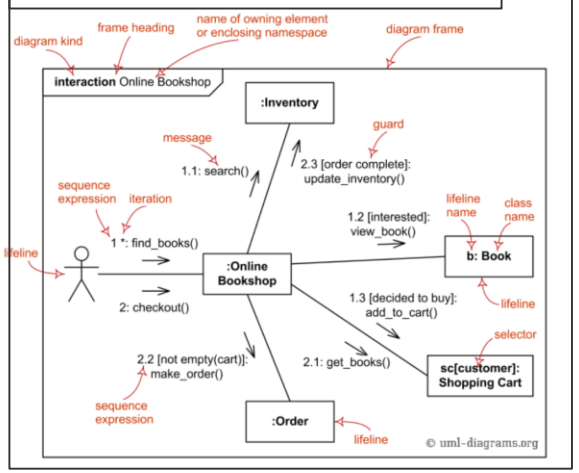
Timing Diagram



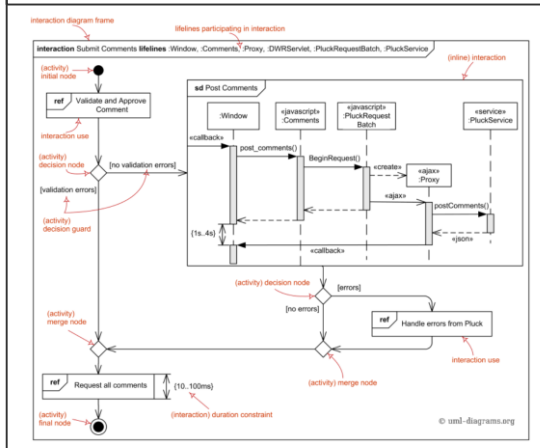
Sequence Diagram



Communication Diagram



Interaction Overview Diagram



https://www.uml-diagrams.org/uml-24-diagrams.html

Common UML Practices

- Avoid large, complicated diagrams
- Focus on the critical parts
- Make the diagram easy to look at
  - Avoid line crossings
  - Only use horizontal and vertical lines (thus only 90° angles)
  - Align elements
  - Use same size icons whenever possible
- For generalizations and realization, try to place parents above children

andrea.carrara@tum.de; patrick.berggold@tum.de

Grady Booch (UML Creator) Quote

"The UML metamodel became grossly bloated, because of the drive to model driven development. I think that complexity was an unnecessary mistake."

"I rather still like the UML. Seriously, you need about 20% of the UML to do 80% of the kind of design you might want to do in a project - agile or not […] *use the notation to reason about a system, to communicate your intent to others...and then throw away most of your diagrams."*

Grady Booch, designer of the UML https://www.infoq.com/articles/booch-cope-interview/

Do it yourself

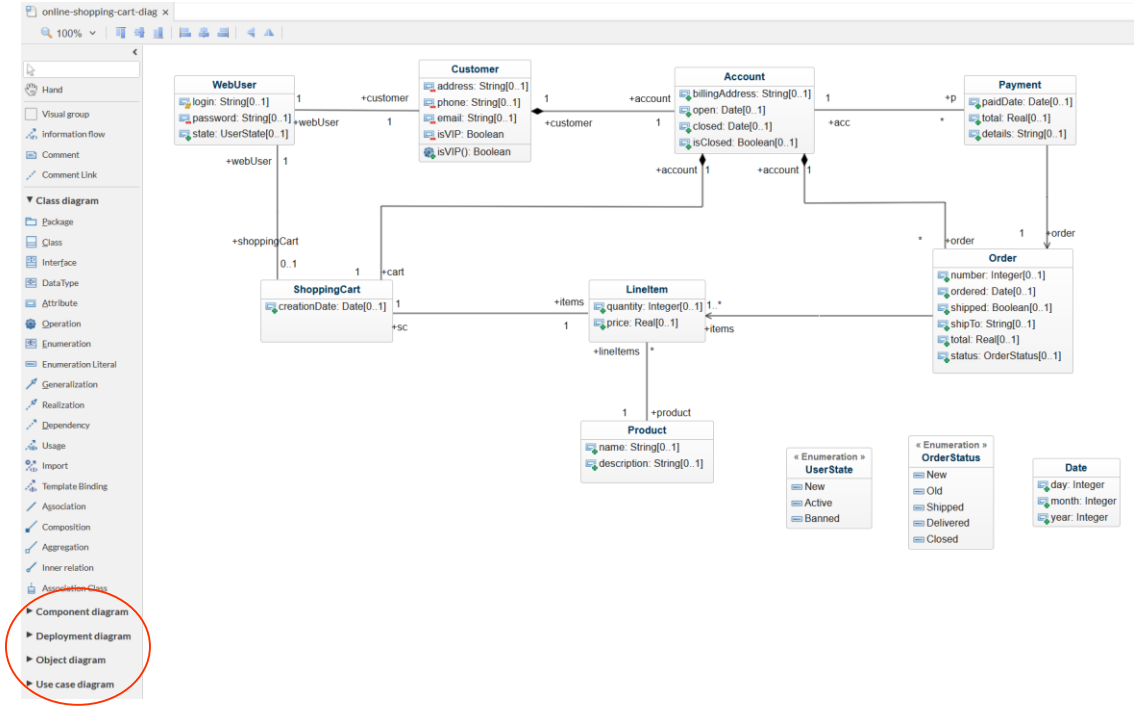
- LucidChart: https://www.lucidchart.com/

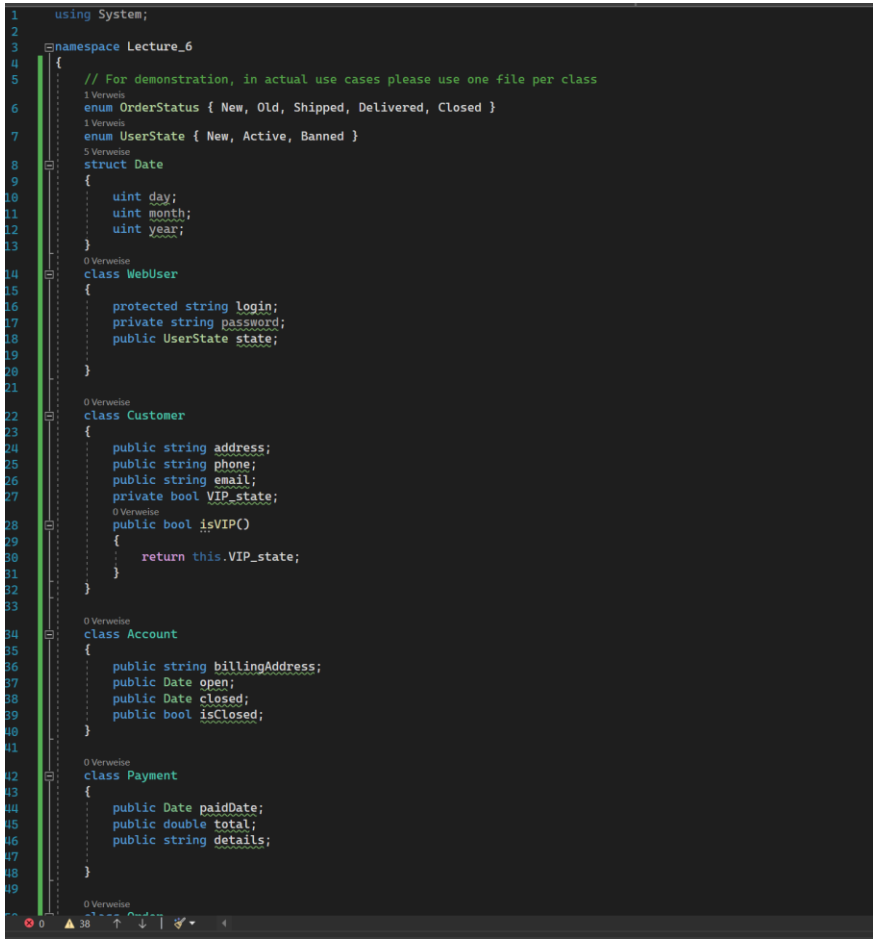
- GenMyProject: https://app.genmymodel.com/

andrea.carrara@tum.de; patrick.berggold@tum.de

Do it yourself

» Automatic code generation in different languages





```
using System;

namespace Lecture_6
{
    // For demonstration, in actual use cases please use one file per class
    1 Verweis
    enum OrderStatus { New, Old, Shipped, Delivered, Closed }
    1 Verweis
    enum UserState { New, Active, Banned }
    5 Verweise
    struct Date
    {
        uint day;
        uint month;
        uint year;
    }
    0 Verweise
    class WebUser
    {
        protected string login;
        private string password;
        public UserState state;

    }

    0 Verweise
    class Customer
    {
        public string address;
        public string phone;
        public string email;
        private bool VIP_state;
        0 Verweise
        public bool isVIP()
        {
            return this.VIP_state;
        }
    }

    0 Verweise
    class Account
    {
        public string billingAddress;
        public Date open;
        public Date closed;
        public bool isClosed;
    }

    0 Verweise
    class Payment
    {
        public Date paidDate;
        public double total;
        public string details;

    }

    0 Verweise
```

Summary UML

- UML is a standardized way to design and communicate your program structure
- You can display contents, use cases, implementation details and dynamic behavior
- Make sure that the diagram is comprehensible and clear, otherwise it loses its purpose

andrea.carrara@tum.de; patrick.berggold@tum.de

THANK YOU!

andrea.carrara@tum.de; patrick.berggold@tum.de