Lab Course / Praktikum

Project Management and Software Development for Medical Applications

UML: Unified Modeling Language – 26/04/2021

Summer Semester 2022

Conducted by:
Ardit Ramadani, Tianyu Song, Lennart Bastian
Slides by: Farid Azampour, Lennart Bastian

TUM
Technische Universität München

Johns Hopkins
Whiting School of Engineering

Agenda

Part 1

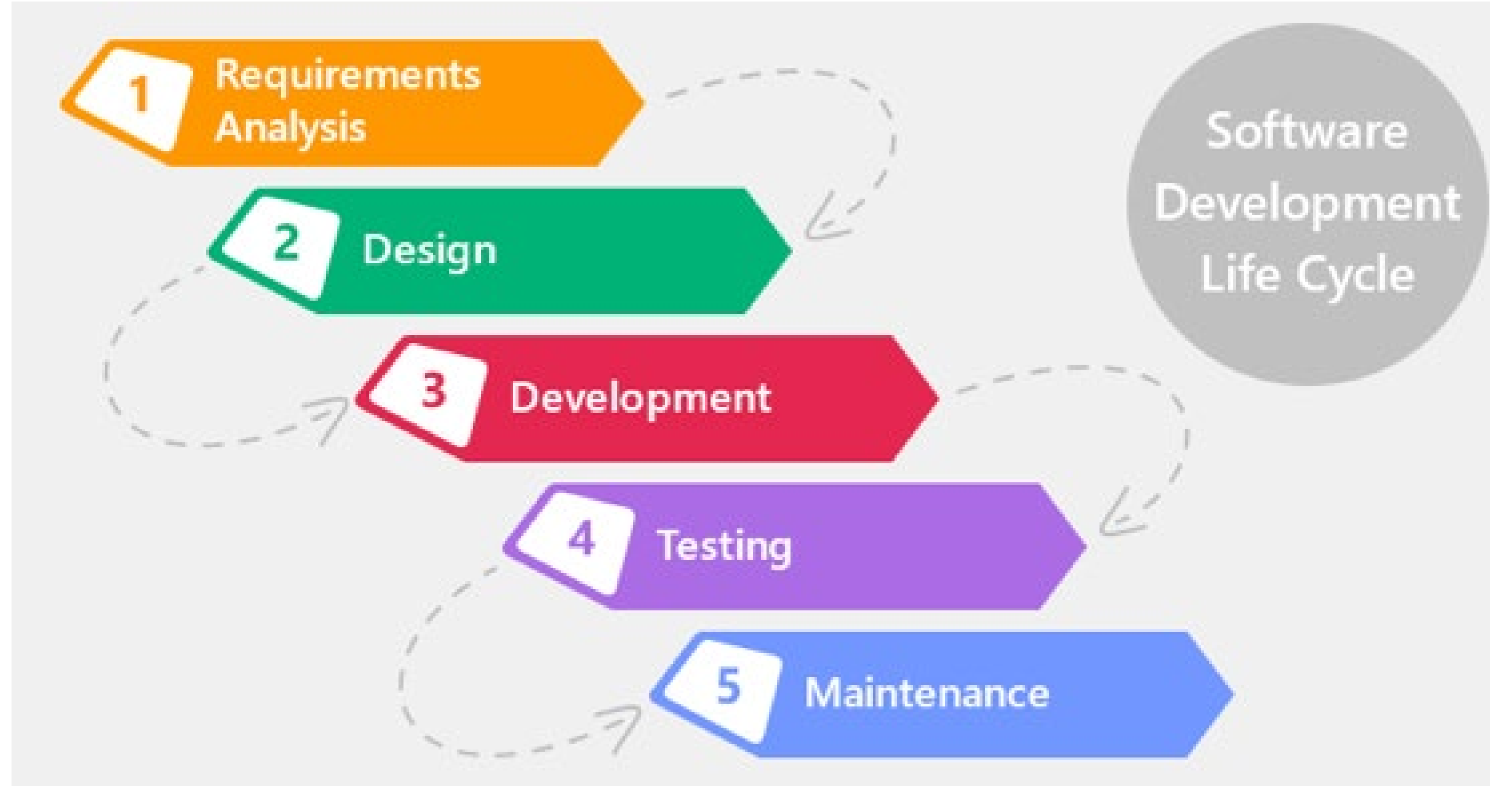
- Software Design
- What is UML?
- Class Diagrams
- Use Case Diagrams

Short break (if needed)

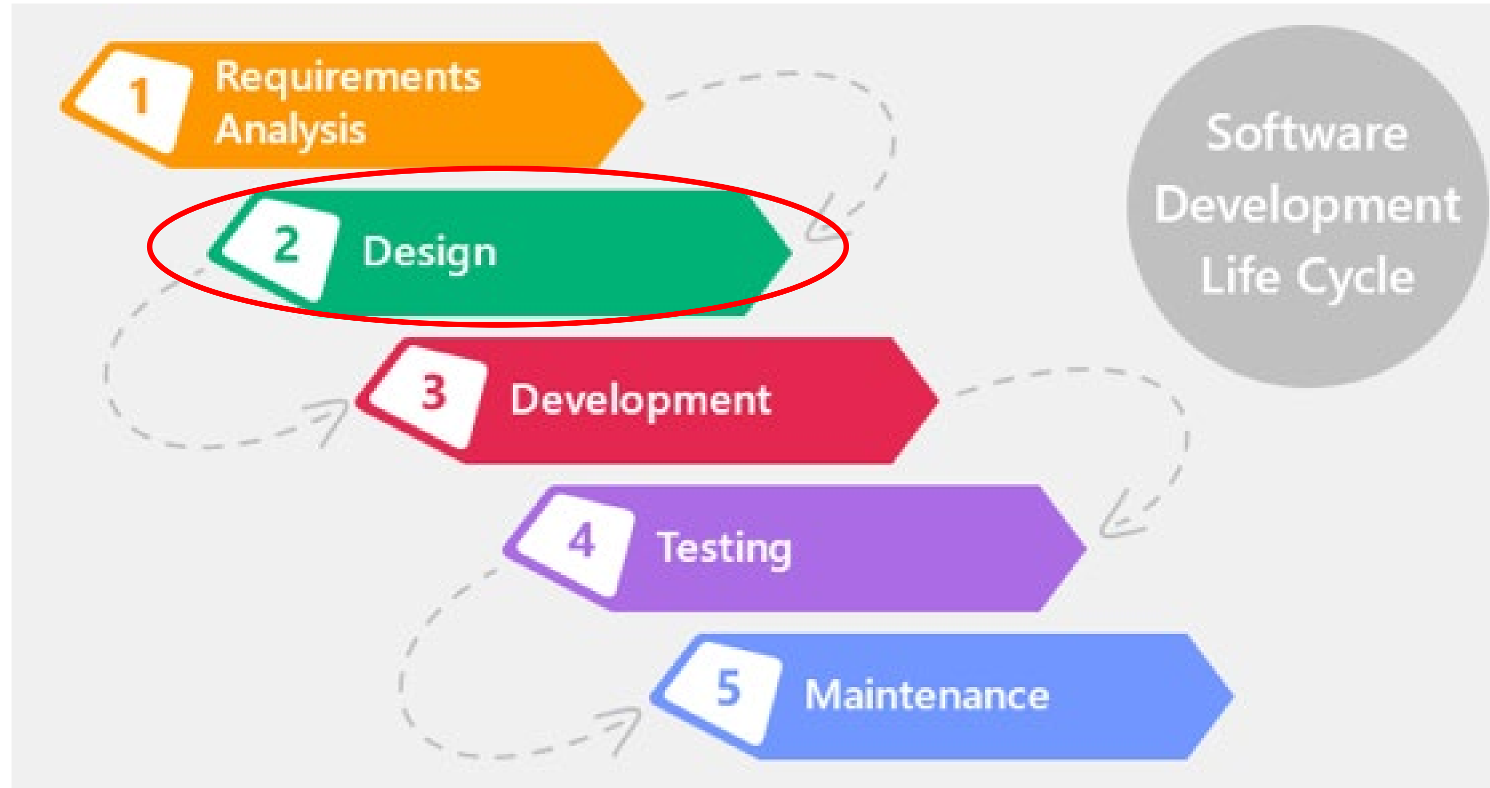
Part 2

- Component Diagrams
- Sequence Diagrams
- Activity Diagrams
- UML Tools

Software Development Life Cycle



Software Development Life Cycle



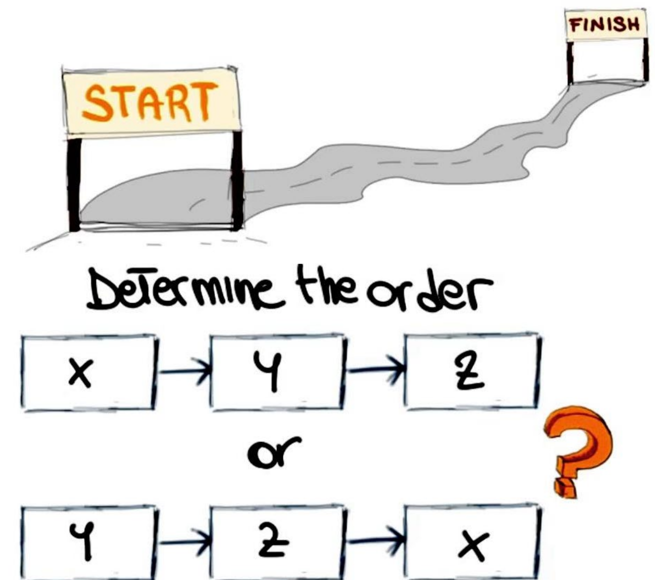
Software Design is about modeling software systems

The Importance of Modeling

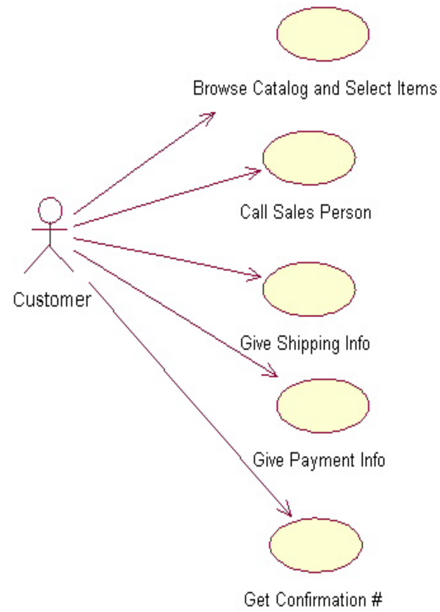
- A model is a simplification of reality.
- We build models so we can better understand the system we are developing

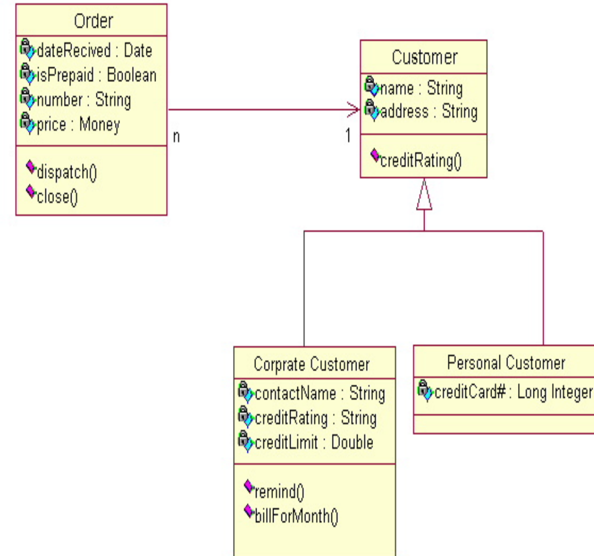
Benefits:

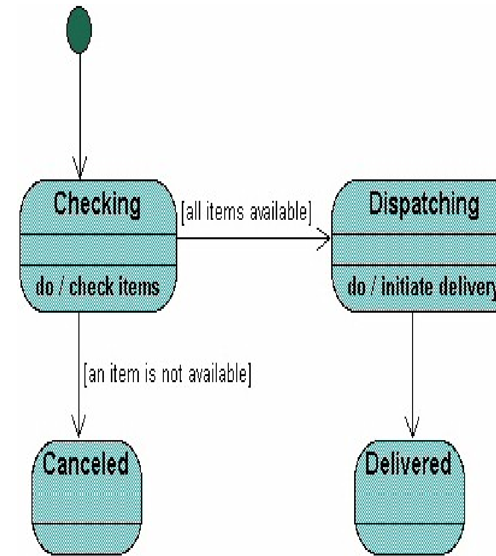
- Visualization:
  - Improves the productivity of the development team
  - Permits early exploration of alternatives
- Behavior and Structure:
  - Improves the understanding of the system
- System Template:
  - Reduces the number of defects in the final code
  - Increases the decomposition and modularization of the system
  - Facilitates the system's evolution and maintenance
- Documentation



Visuals Help in Software Design







Clients/stakeholders

Developers

Quality Assurance

Unified Modeling Language (UML)…

… is an industry standard for visualizing, specifying, constructing, and documenting the artifacts of a software system.

… is not for the sake of creating diagrams, however…

... it is about **creating a model** which you visualize in a diagram according to the de facto standard modeling language.

… is extended and updated over time:

1996 UML 1.0
2005 UML 2.0
2009 UML 2.2
2015 UML 2.5
2017 UML 2.5.1

https://www.omg.org/spec/UML/

UML Is a Formal Visual Language

- UML offers a framework for deriving simplified models of a complex reality

- UML is a graphics-based language with syntax and semantics
  - Every valid structure has a meaning (semantics)
  - There are rules defining how structures may be put together (syntax)
  - Programming platform independent

- UML is a great communication tool
  - Natural language is often imprecise and source code too detailed
  - It standardizes a very natural, visual communication approach: drawings
  - UML moves us from fragmentation to standardization while offering flexibility

Structure and Behavior Are The Main Two UML Types

Meta: UML diagram of available UML diagrams

Diagram

What things are needed and how are they connected?

Structure Diagram

- Class Diagram
- Component Diagram
- Object Diagram
- Profile Diagram
- Composite Structure Diagram
- Deployment Diagram
- Package Diagram

What's the functionality and how is it achieved?

Behaviour Diagram

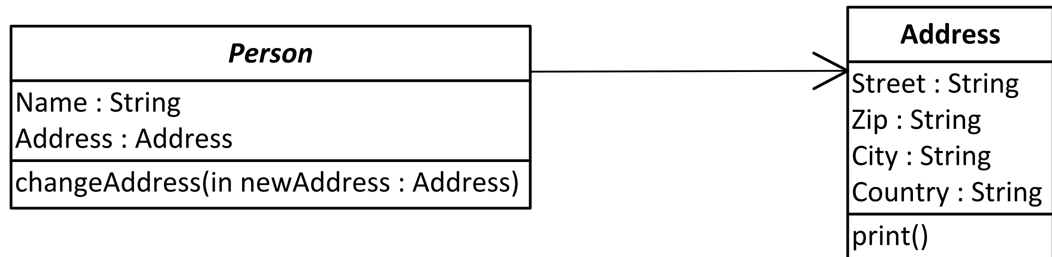
- Activity Diagram
- Use Case Diagram
- Interaction Diagram
  - Sequence Diagram
  - Communication Diagram
  - Interaction Overview Diagram
  - Timing Diagram
- State Machine Diagram

Notation: UML

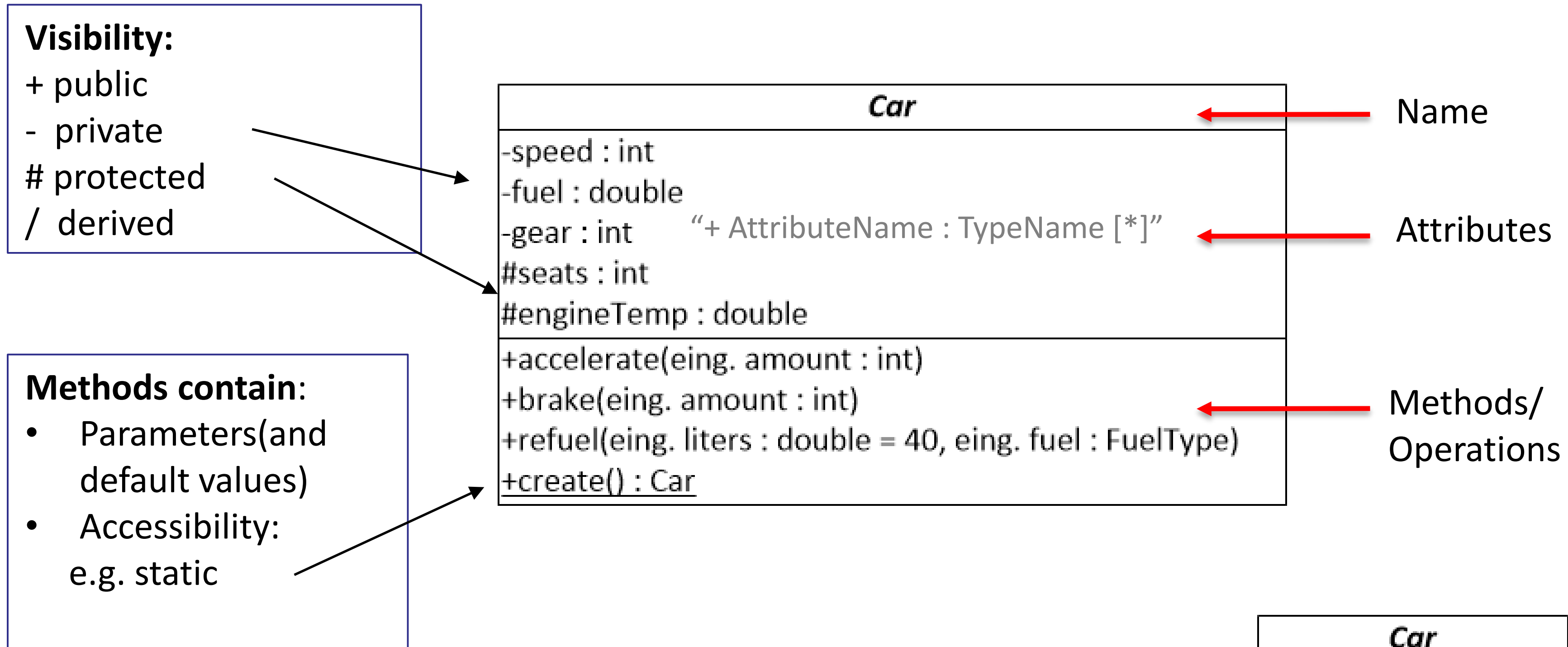
Class Diagram - Introduction

- Is a static structure diagram based on classes
- No logic or functionality included
- Probably the most popular and most used diagram type
  - Still UML diagram != UML class diagram
  - INSTEAD: Planning should start with other diagrams!

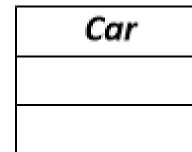
- Consists of
  1. Classes: attributes, operations or methods
  2. Relationships among the classes.

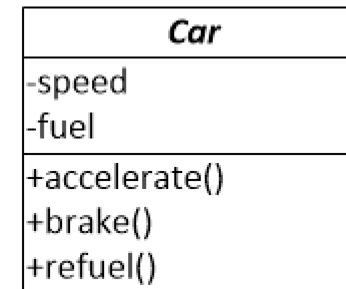


Class Diagram – Syntax



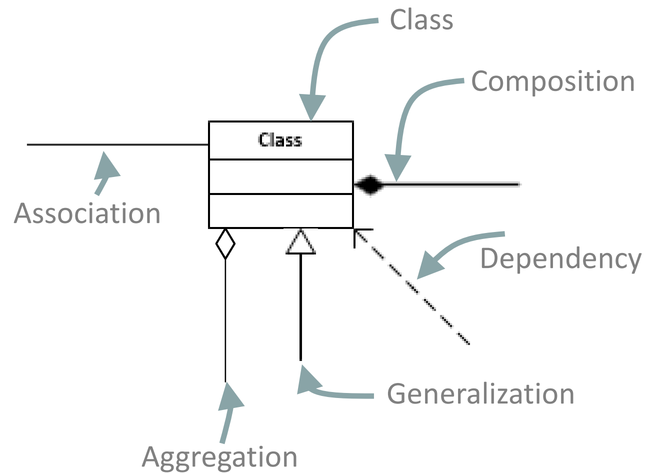
Can have different levels of detail!!

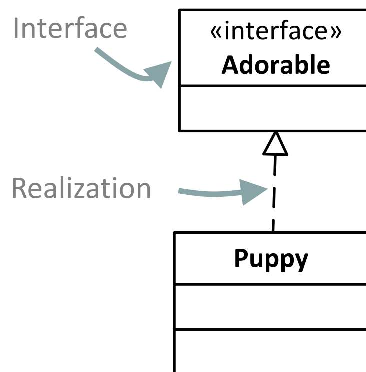




Name *Italic* if abstract

Class Diagram - Relationships





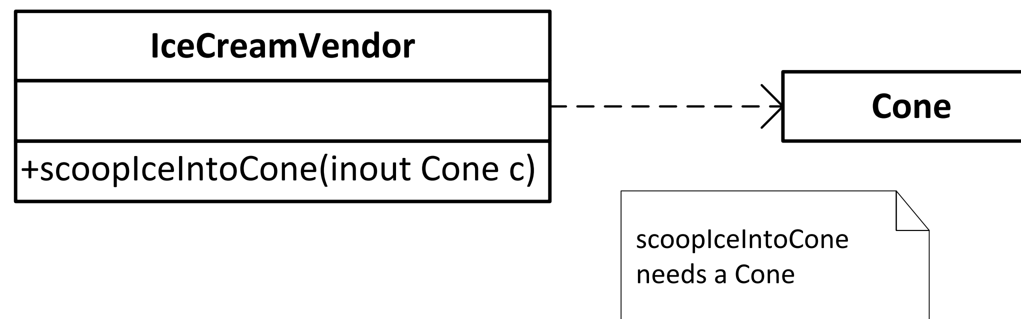
this class is associated with	this class
this class is dependent upon	this class
this class inherits from	this class
this class has	this interface
this class is a realisation of	this class
you can navigate from this class to	this class
these classes compose without belonging to	this class
these classes compose and are contained by	this class
this object sends a synchronous message to	this object
this object sends an asynchronous message to	this object

Multiplicity: How many objects can participate in the relationship. Indicates the number of instances of one class linked to instances of the other class.

0 or more: *****
1 or more **1..***
From 2 to 4: **2..4**
Only 7: **7**

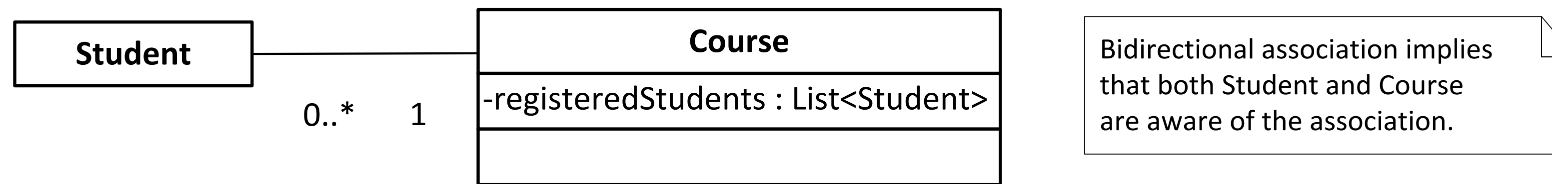
Class Diagram - Dependencies

- Means that one class makes use of another class at some point in time
- Can be read as "… needs a …" or "… uses a …"
- Weakest form of relation after "no relation"
- Typically used if the independent class is a parameter of an operation of the dependent class
- Stereotypes are commonly used to highlight the precise nature of a dependency, e.g. <<displays>>, <<calls>>, <<creates>>

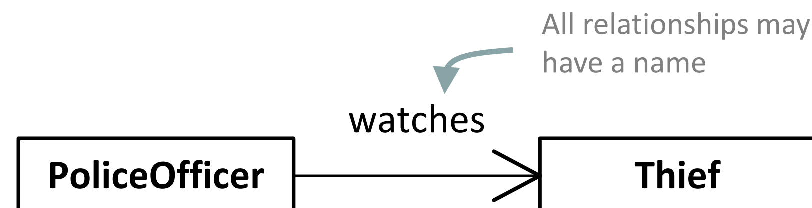


Class Diagram - Associations

- Stronger relationship than dependencies
- Implies that two classes are coupled over a long period of time
- Can be read as "… has a …" or "… knows of …"
- Typically used if one class has an attribute with a type of another class

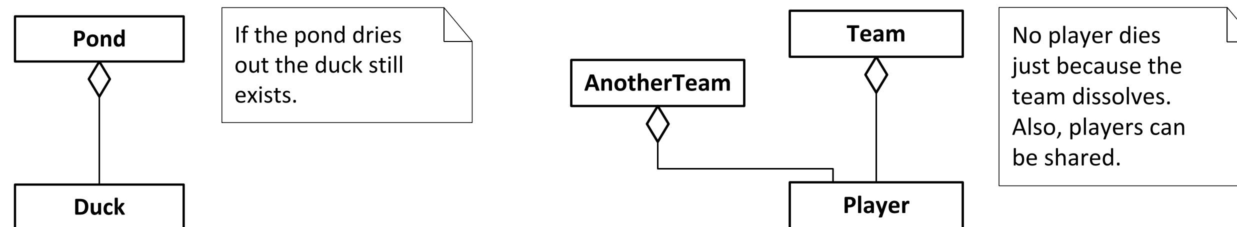


- Associations can be unidirectional to indicate navigability

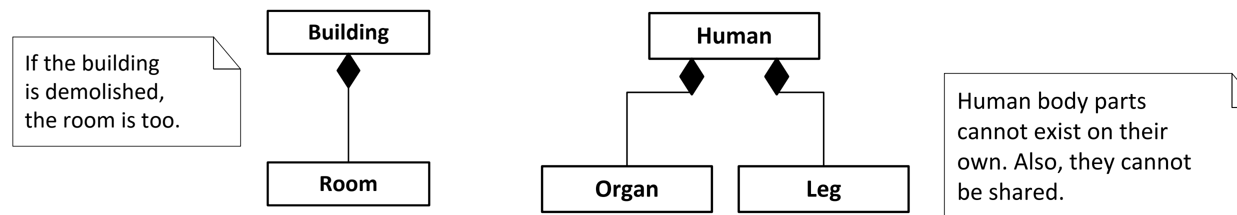


Class Diagram - Composition and Aggregation

- Both aggregation and composition represent a whole-part relationship
- The main difference is how dependent the "part" is on the "whole"
- **Aggregation**: parts do not depend on lifecycle of the whole

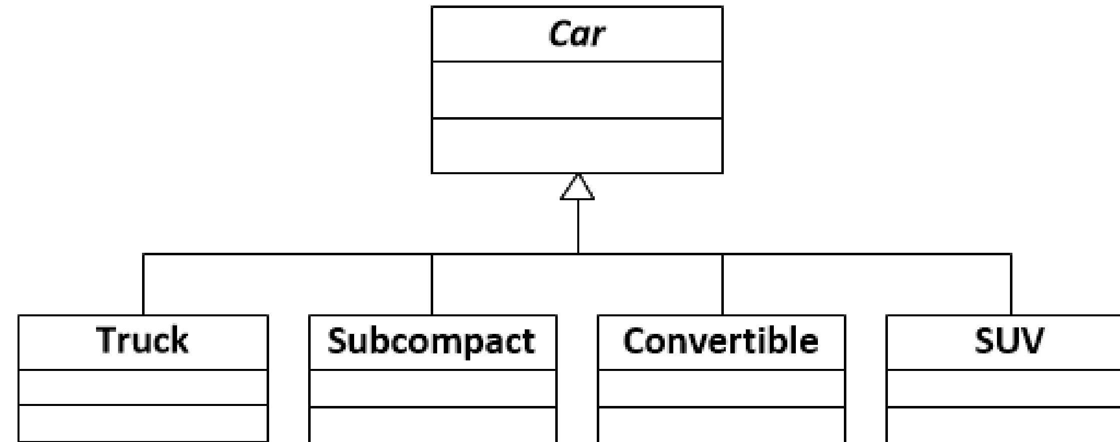


- **Composition**: parts depend on the lifecycle of the whole, they are "owned"

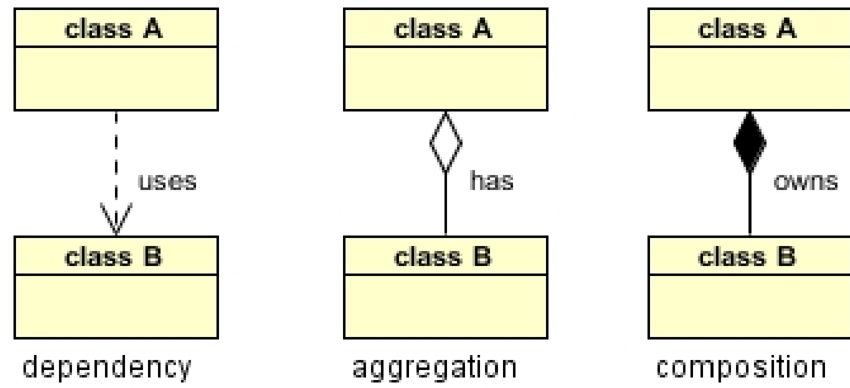


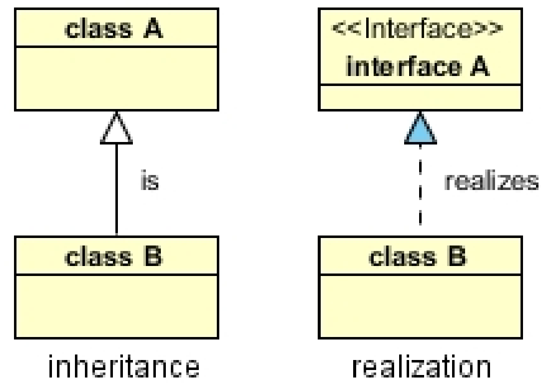
Class Diagram – Generalization/Specialization

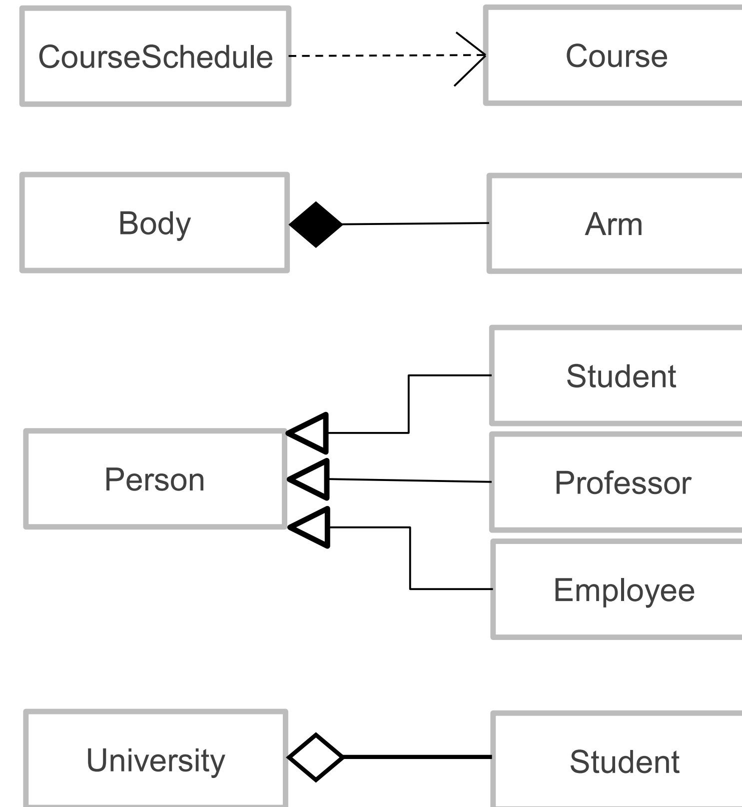
- Used to show inheritance
- Reads as "… is a …"



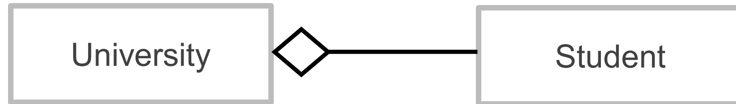
Class Diagram - Summary

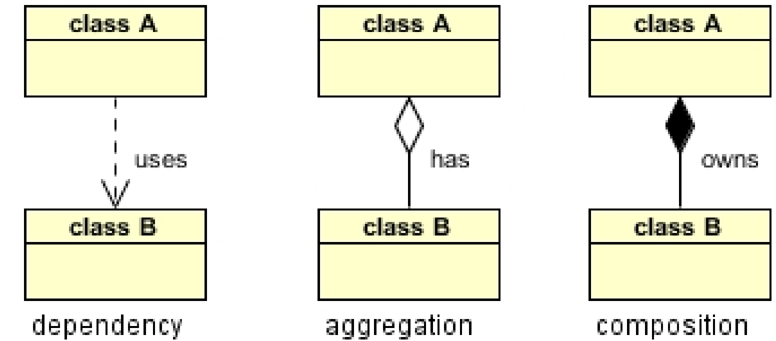






Quiz





Which statements are true?

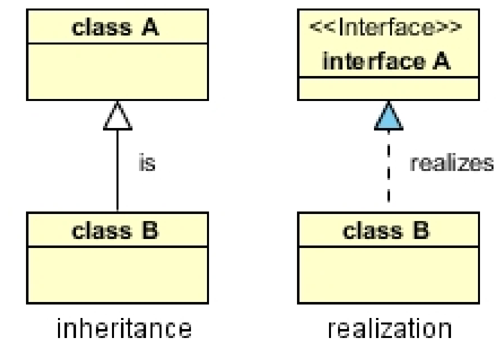
1. "Student" is part of "University".

2. If an instance of "University" is deleted, all contained instances of "Student" are also deleted.

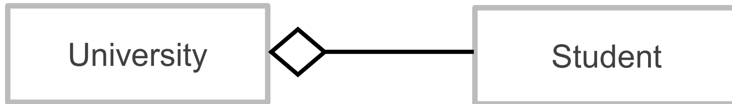
3. If an instance of "University" is deleted, the contained instances of "Student" are not affected.

4. "University" is part of "Student".

5. If an instance of "Student" is deleted, all contained instances of "University" are also deleted.



Quiz Time cont.



Which statements are true?

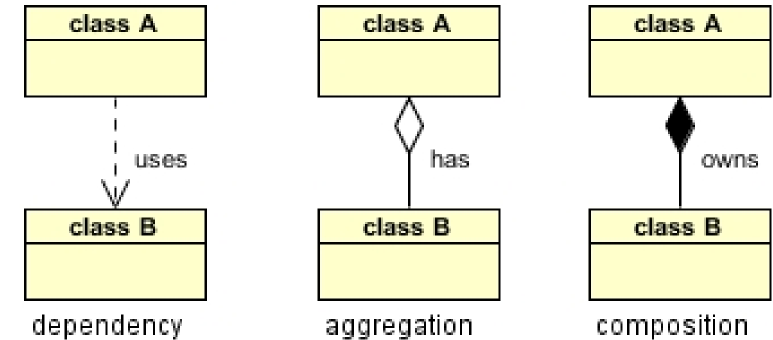
1. "Student" is part of "University". ✔

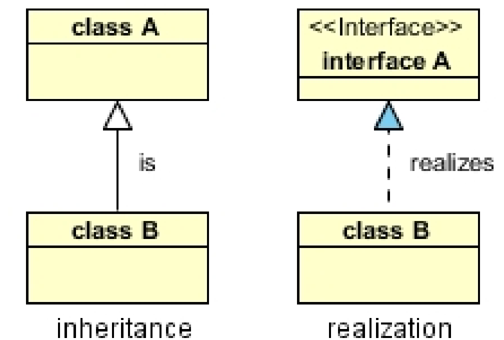
2. If an instance of "University" is deleted, all contained instances of "Student" are also deleted. ✘

3. If an instance of "University" is deleted, the contained instances of "Student" are not affected. ✔

4. "University" is part of "Student". ✘

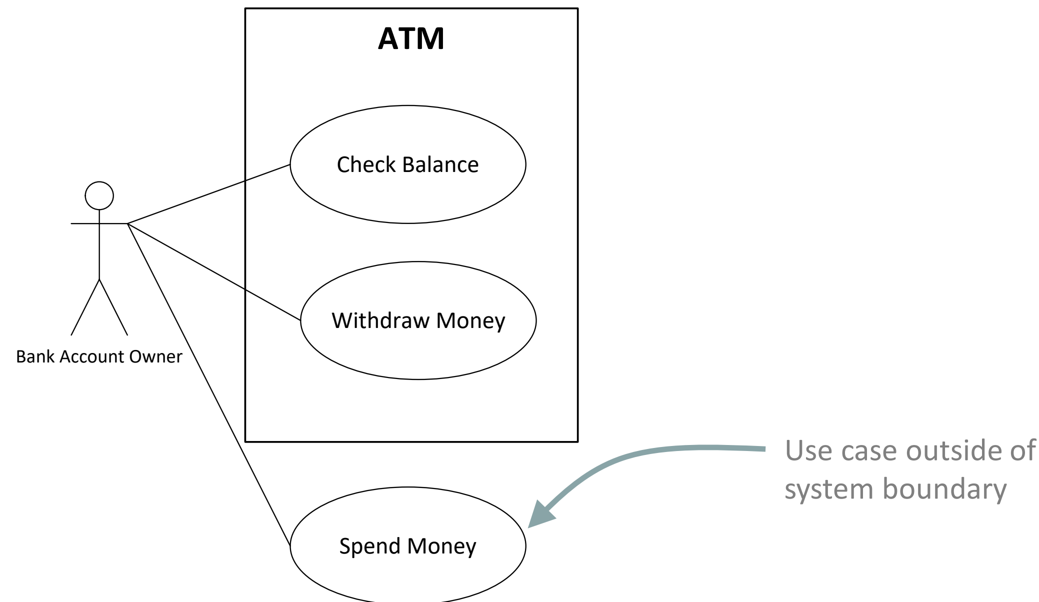
5. If an instance of "Student" is deleted, all contained instances of "University" are also deleted. ✘





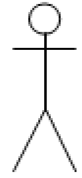
Use Case Diagram

- A use case diagram shows the functionality of a system
  - "What can a system do?", not "How does it do it?"
- It consists of named entities of functionality (use cases) and entities triggering the use cases or receiving their output (actors)
- Very important for requirements analysis: Identify all users and functionalities



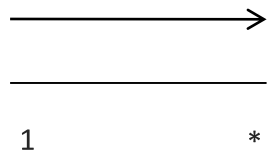
Use case outside of system boundary

Elements of Use Case Diagrams (I/II)



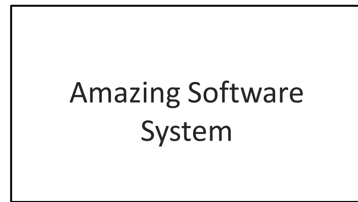
- Actors
  - Humans or external systems, interacting with the system
  - Often custom representations are used

- Use cases
  - A piece of functionality, provided by the system
  - Best practice for naming: [action verb] + [object]
    - E.g. “place order”, “invite attendees”, “pay invoice”
  - Use cases do **not** describe the process (the single steps) of how this interaction takes place, this might be documented within the use case description.



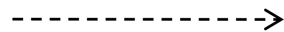
- Actor/Use Case Association
  - Indicates that the actor initiates a use case or receives results from the use case, or both → directionality optional
  - May show multiplicities
  - No Actor/Actor associations!

Elements of Use Case Diagrams (II/II)

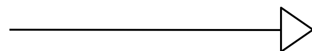


- System boundaries
  - Box with name, use cases inside box, actors outside box
  - Yet, showing use cases outside your system can be useful

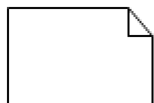
- Use Case Extension
  - Indicates additional functionality on top of a base use case
  - Base use case is complete use case on its own



- Use Case Inclusion
  - Indicates that base use case is not complete on its own but *includes* one or more other use cases
  - Used to group smaller use cases into a large one or to factor out common functionality that is included by several use cases



- Generalizations (Actor/Actor or Use Case/Use Case)



- Comments

Use Case Diagram Example: Online Shopping System

Online Shopping

System boundary

Association

Create Account

Generalization

Use case

New Customer

Browse Items

Use case inclusion

Web Visitor

«includes»

Make Purchase

Actor

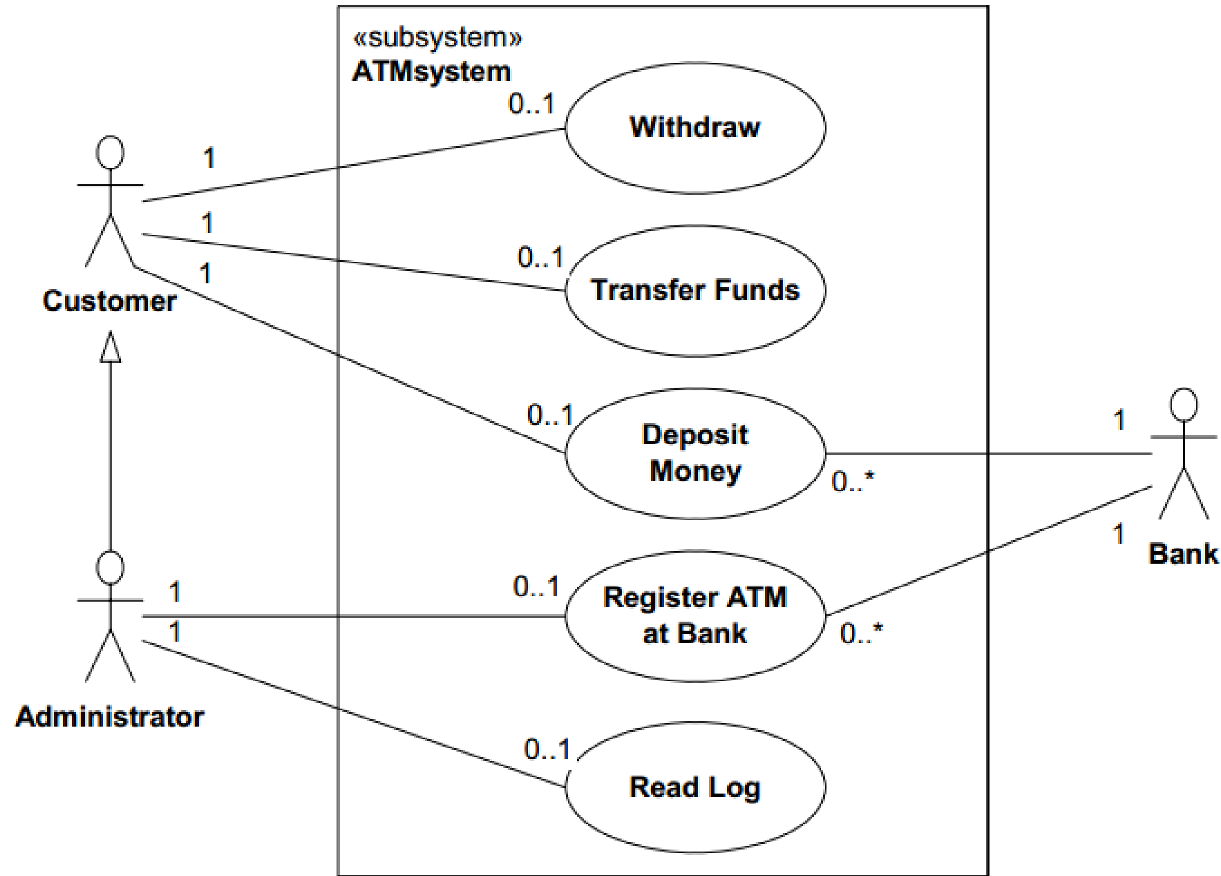
Registered Customer

«includes»

Checkout

Payment Processor

Use-Case Diagram Example: Multiplicities



From the UML Superstructure Specification

Quiz

Which of the following use cases are correct use cases when you want to design a use case diagram for an online book shop?

1. Look for a book

2. Cancel order

3. Do not order a book

4. Login

5. Enter book title

6. Order a book

7. Enter credit card details

Quiz Time

Which of the following use cases are correct use cases when you want to design a use case diagram for an online book shop?

1. Look for a book ✔
2. Cancel order ✔
3. Do not order a book ✖
4. Login ✖
5. Enter book title ✖
6. Order a book ✔
7. Enter credit card details ✖

N.B.: Use cases do not describe the single steps of how this interaction takes place. A user will not just enter his credit card details; this might be one single step within the use case "Order a book".

Structure and Behavior Are The Main Two UML Types

Meta: UML diagram of available UML diagrams

Diagram

What things are needed and how are they connected?

Structure Diagram

What's the functionality and how is it achieved?

Behaviour Diagram

Class Diagram

Component Diagram

Object Diagram

Profile Diagram

Composite Structure Diagram

Deployment Diagram

Package Diagram

Activity Diagram

Use Case Diagram

Interaction Diagram

State Machine Diagram

Sequence Diagram

Communication Diagram

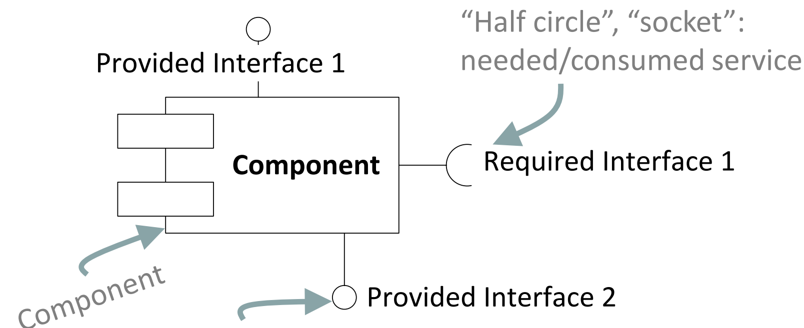
Interaction Overview Diagram

Timing Diagram

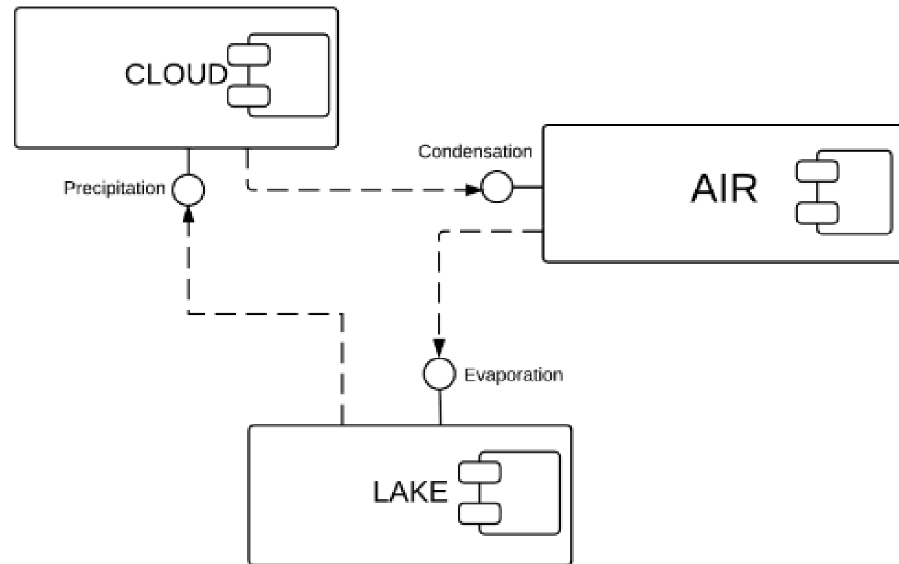
Notation: UML

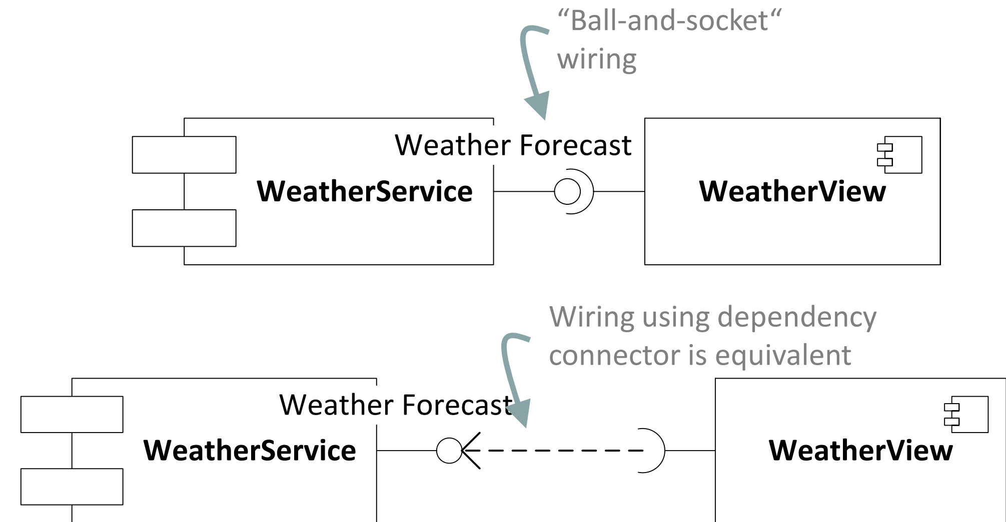
Component Diagram

- Show the structural relationships between the components of a system (higher level than class diagram).
- Very easy type of diagram, related to class diagram
- Components are highly cohesive and encapsulated high-level design units (“black boxes”).
  - Often components are classes but not necessarily
- The communication is through interfaces with the inputs required and services provided.
- Can be used independent of OO programming languages



Component Diagram - Examples





More complex example for Component Diagrams

internal structure compartment
structured classifier – subsystem component
«subsystem» WebStore
internal structure
ProductSearch
:SearchEngine
port
provided interface
delegation connector
required interface
role, part component
OnlineShopping
:Shopping Cart
provided interface
UserSession
assembly connector ball-and-socket
UserSession
:Authentication
delegation connector
delegation connector
Search Inventory
Manage Orders
dependency
Manage Customers
«subsystem» Warehouses
internal structure
:Inventory
Manage Inventory
provided interface
dependency
«subsystem» Accounting
internal structure
:Orders
Manage Inventory
required interface
Manage Customers
assembly connector ball-and-socket
:Customers
© uml-diagrams.org

http://www.uml-diagrams.org/component-diagrams.html

Quiz Time!

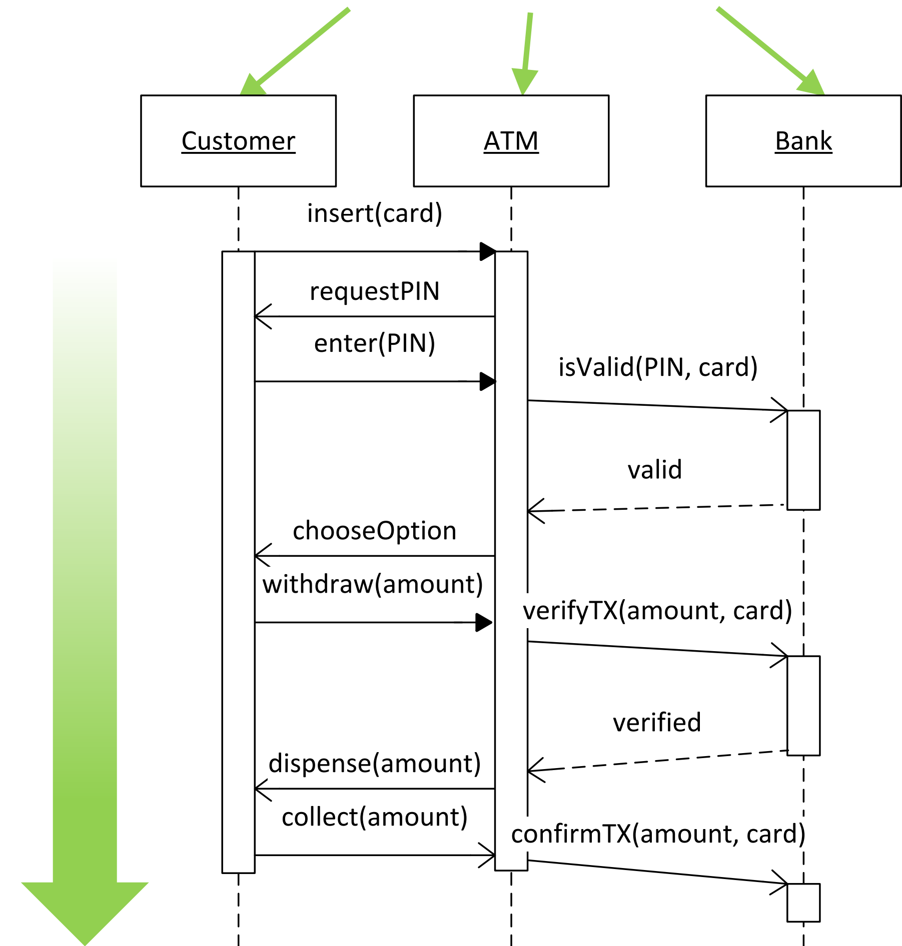
1. Are Component Diagrams Behaviour or Structure Diagrams?

2. What are Component Diagrams used for?

3. Name two examples for components!

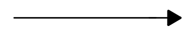
Sequence Diagram

- Is an interaction diagram
  - Models interaction of entities in a system with each other
- Typically used to show system behavior and model the logic in a specific use case
  - “How?” not “What?”
- The horizontal dimension shows the objects participating in the interaction
- The vertical dimension shows messages in their relative order

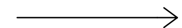


Sequence Diagram - Elements

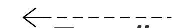
- Objects (interaction participants) with lifeline
  - Notation in software engineering: "Name:Type"
  - Show how long object is in existence
  - Can explicitly put an "x" at the end to highlight destruction
- Activation blocks/bars (formally "Execution specification")
  - Participant is sending, waiting, in general: busy
- Messages (synchronous, asynchronous, return)
  - Synchronous, caller blocks



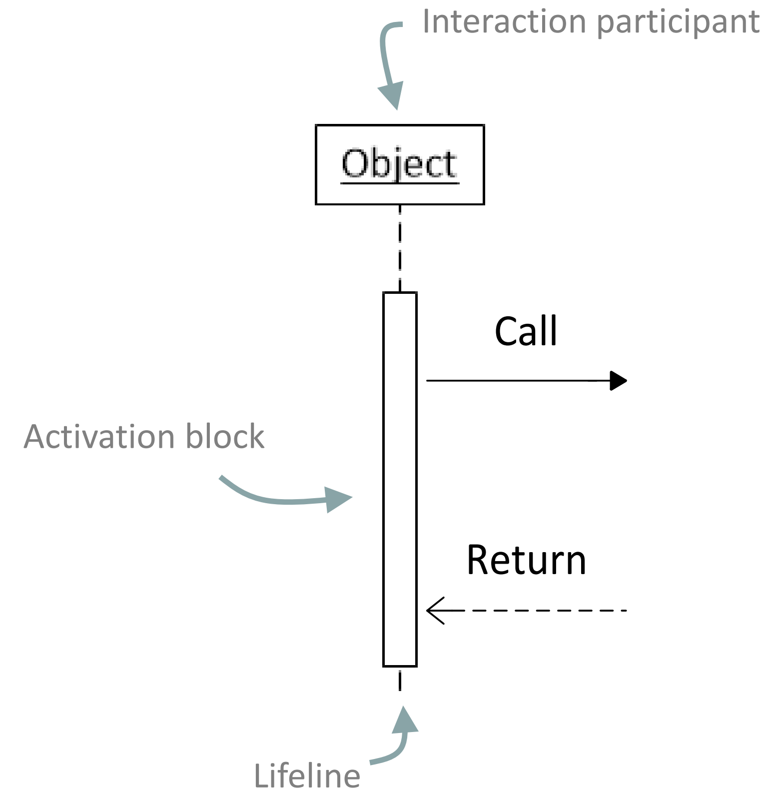
  - Asynchronous, caller does not block



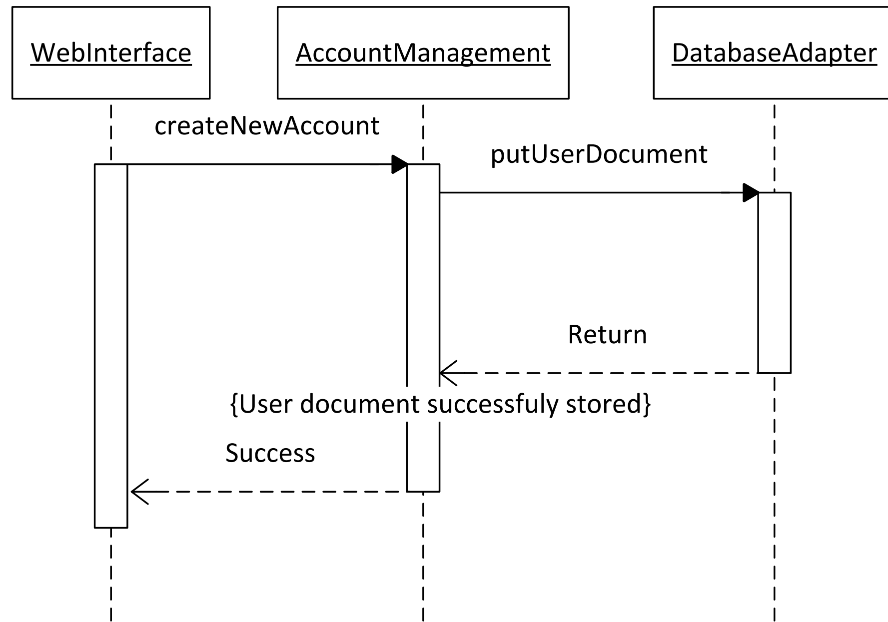
  - Return

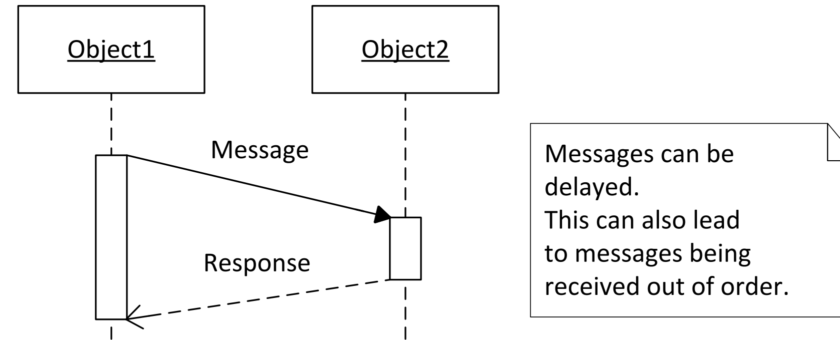


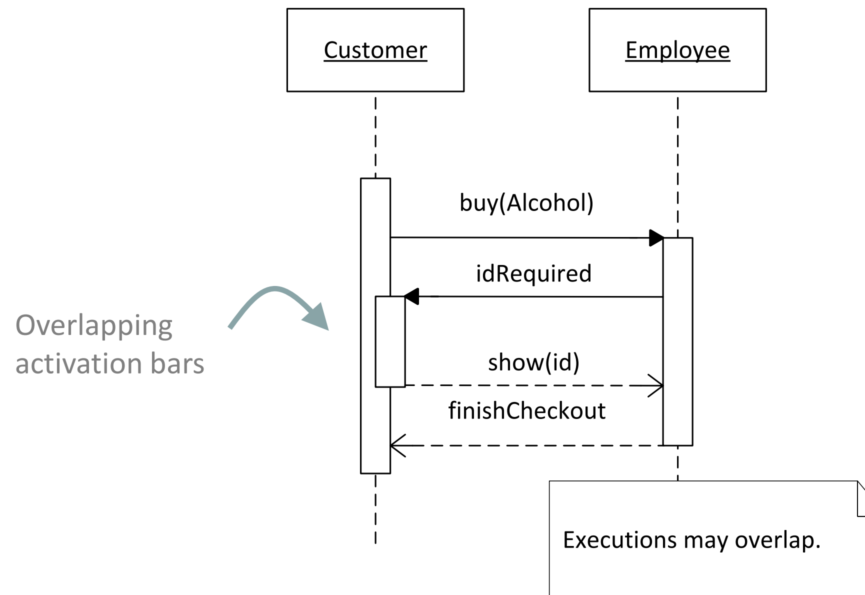
- General notation "*Name(Parameters):ReturnType*"



Sequence Diagram - Examples



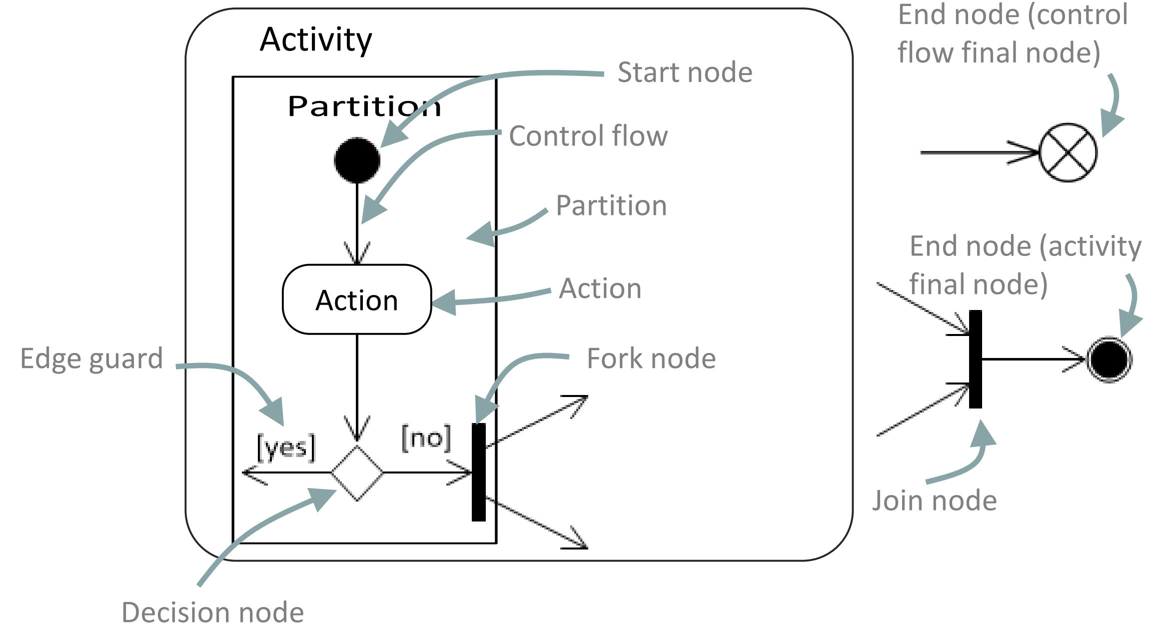




Activity Diagram

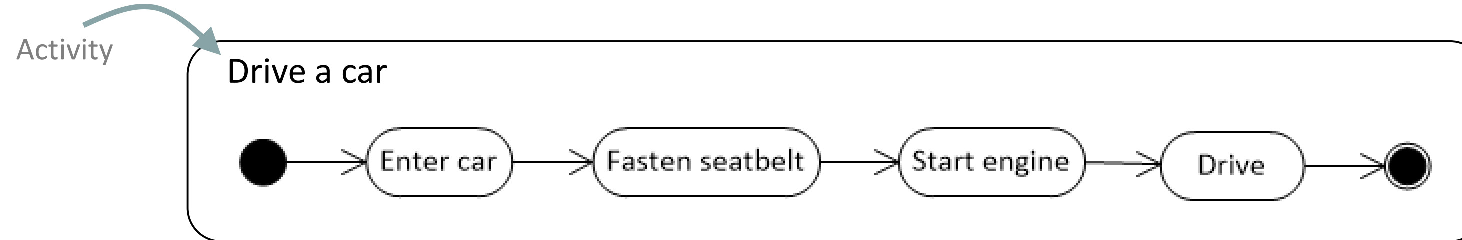
- Models execution and behavior / workflow of a system, not its structure
- Widely applicable to any kind of process modeling
- An activity is described by the actions and objects involved and the flow between them, not the components and their communications
- **Elements of activity diagrams**
  - Activity
  - Actions
  - Control or object flows
  - Decisions and Merges
  - Forks and joins
  - Start and end node
  - Partitions

  …

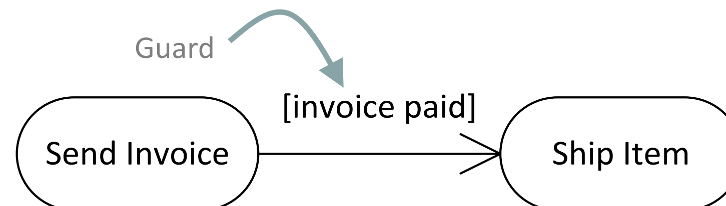


Activity Diagram: Activities, Actions and Transitions

- An activity is a sequence of actions and displayed using round-cornered rectangles that contain all elements of the activity
- An action is a single step within an activity and displayed using round-cornered rectangles

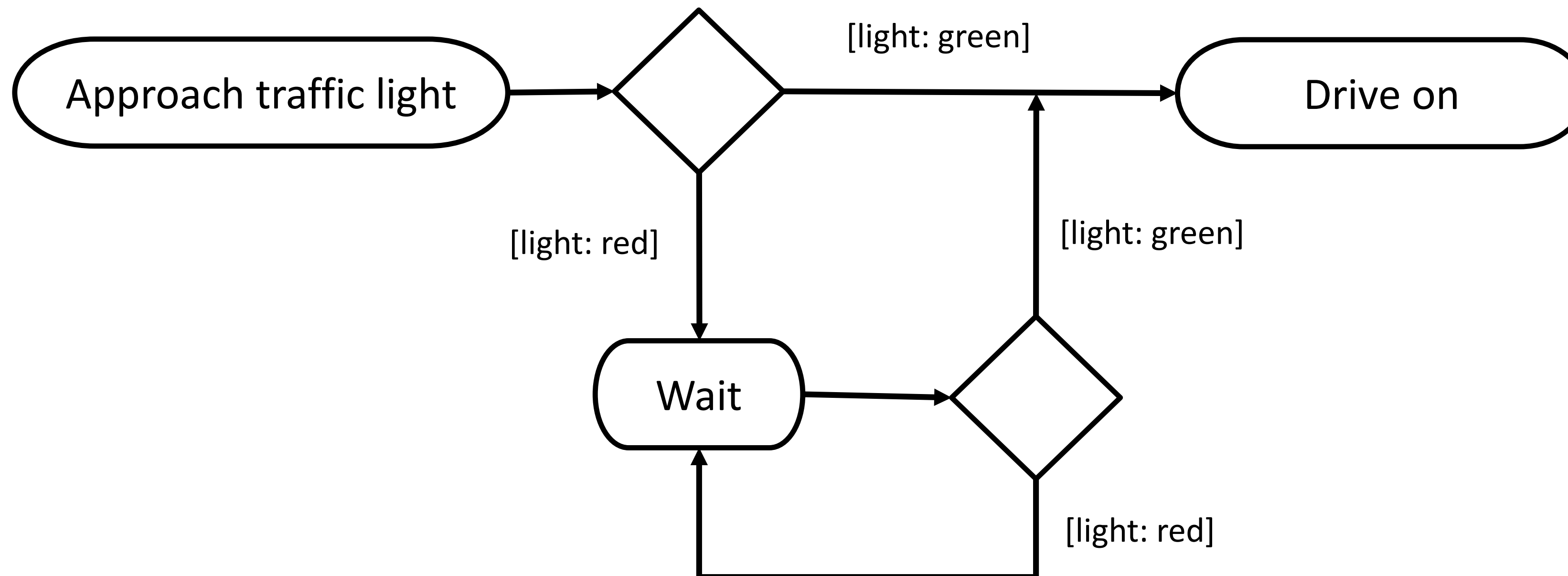


- Edges can have names and/or *guards*
  - A guard is a condition evaluated at runtime to determine if the edge can be traversed
  - Always put guards at decision nodes



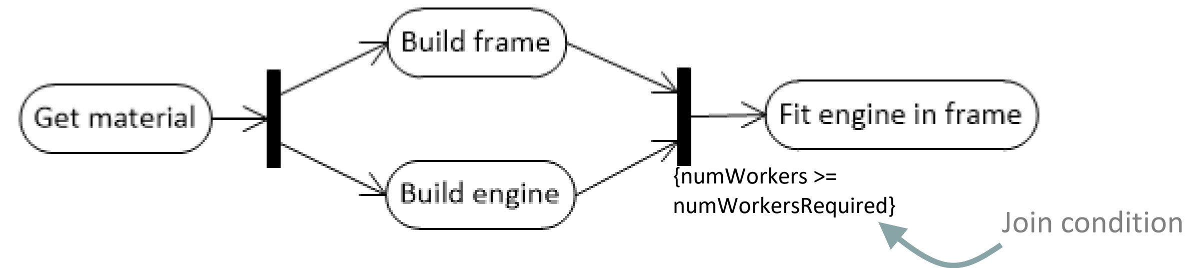
Activity Diagram: Object Decision Nodes

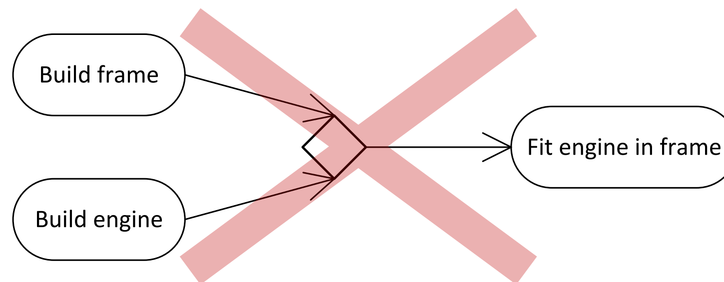
- Can divert and merge the flow
  - Only one outgoing edge can be traversed by each control token
- Guards at the branches define how the flow continues
  - Only one guard should evaluate to true to avoid race conditions
- Merging does not synchronize multiple concurrent flows



Activity Diagram: Forks and Joins

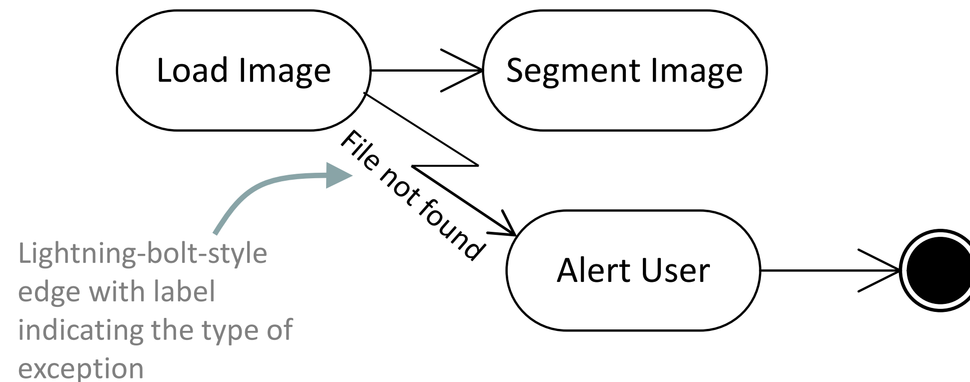
- Forks divide the flow unconditionally into multiple concurrent flows
- Joins synchronize multiple flows and continue once all flows are merged
- Joins can be conditional





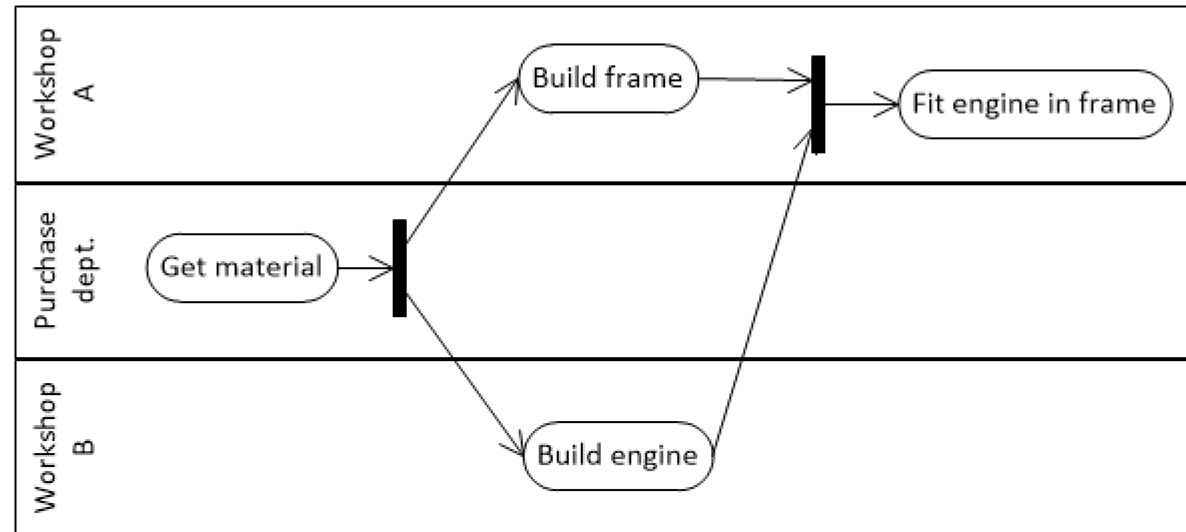
Activity Diagram: Exception Handler

- Especially important when describing the workflow of a software system
- An exception is an error that occurs during the execution of an action
- An exception handler protects an action from a specific type of exception by catching the exception
- If not caught, exceptions propagate



Partitions

- Partitions are a grouping mechanism
- They can be used to visualize responsibilities



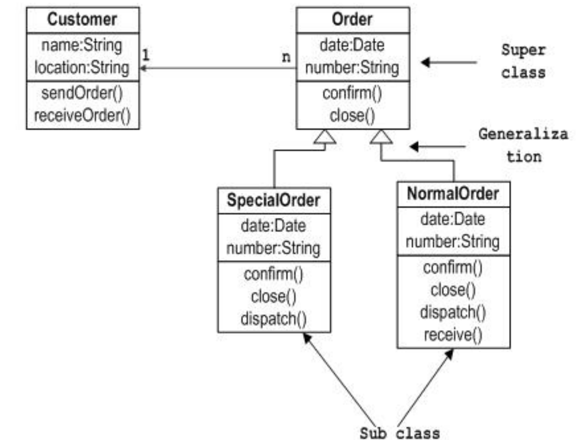
There are many UML tools

- First iteration during initial design: By hand

- ArgoUML (http://argouml.tigris.org)
- Dia (http://live.gnome.org/Dia)
- Modelio (http://www.modelio.org)
- Microsoft Visio (available for students through www.studisoft.de)
  - Be sure to check out supplementary UML stencils: http://www.softwarestencils.com/uml/index.html
- MySQL Workbench
- yED (http://www.yworks.com/en/products_yed_about.html)
- IDEs like Visual Studio, Eclipse and NetBeans often offer UML functionality (directly or through plugins)
- Actually there are hundreds of tools available – a good start is on wiki: http://en.wikipedia.org/wiki/List_of_UML_tools

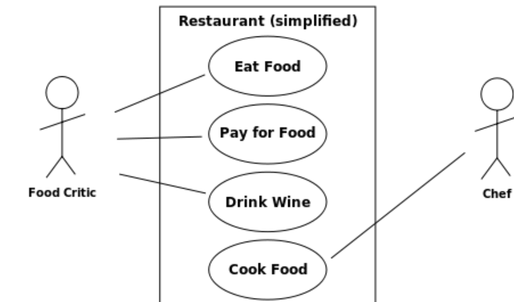
Summary

- UML is a visual modeling language that eases communication about legacy as well as new (software) systems
  - Use it for the requirements analysis and for designing your implementation



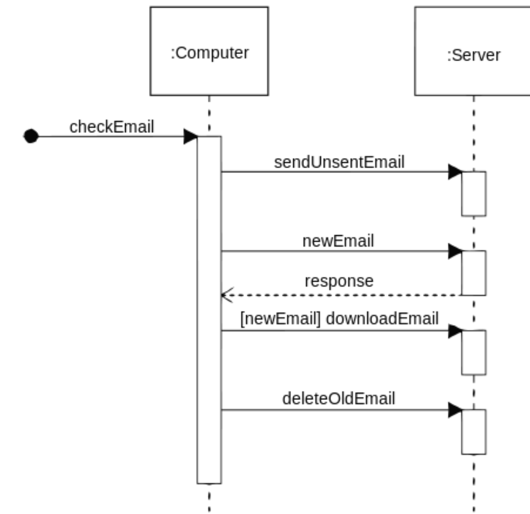
- **Class and component** diagrams are structural diagrams
  - Start designing your implementation by drawing a component diagram
  - Afterwards, and in case you are using OOP, draw a more detailed class diagram

- **Use case** diagrams are behavioral diagrams that show the functionality your system provides on a high level
  - Use them while defining the functionality of your software project
  - Use them to quickly communicate what your system is capable of (e.g. here in class)

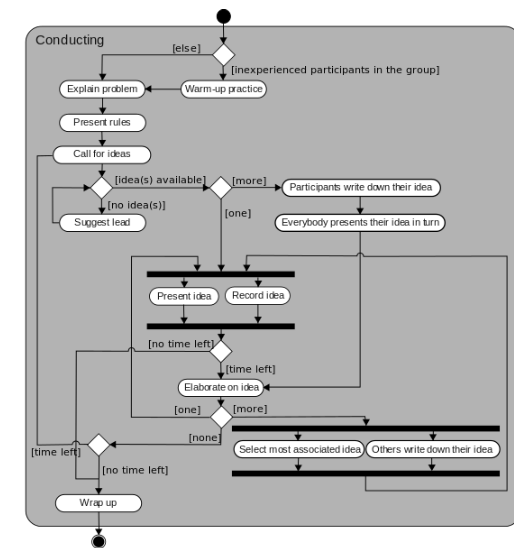


Summary

- **Sequence** diagrams are behavioral diagrams that show how functionality of your system is realized by visualizing detailed communication flows between the components involved
  - Use them to show the inner workings of your system that make a use case (functionality) possible
  - Good for showing detailed logic



- **Activity** diagrams are behavioral diagrams that capture how a functionality is executed in terms of subsequent actions
  - Use them to show how a use case is realized by a series of simple steps
  - Good for showing logic of your system without exposing too many technical details



Nota Bene

We do expect you to show us (at least) one **UML diagram of your choice**
in the kick-of and intermediate presentation!

- Usually recommended* for **kick-off:** Use Case Diagram "What functionality should my application offer? Who are the actors?"

- Usually recommended* for **intermediate**: Class Diagram or Sequence Diagram for more detailed visualization for your design choices

- Do not mix the UML diagram types and stick to the rules

* Of course this is heavily depending on your individual project and application.

Nota Bene

Identify ideas and resources that are not your own!

TUM **Citation Guide** and Informatics **Code of Conduct for Students**

If you are unsure about copyright or whether you can use a software library, github repository etc.,

... read the copyright notice or license!
... ask your supervisor or the course tutors!