Lecture 09

Business Informatics 2 (PWIN)
WS 2017/2018

ICS Development II
Object Orientation & UML

Prof. Dr. Kai Rannenberg

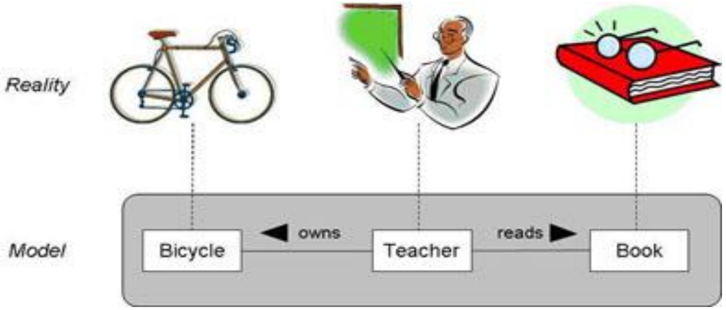
Deutsche Telekom Chair of Mobile Business & Multilateral Security
Johann Wolfgang Goethe University Frankfurt a. M.

Agenda

- Object-Oriented Approach
- Unified Modelling Language (UML)
- Model-Driven Development and Architectures

The Idea of Object Orientation (OO)

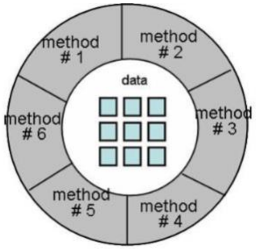
- OO sees things that are part of the real world.



- OO-Models represent only the relevant aspects of real world things.

- Name
- Phone No.
- E-Mail
- Teaching Subjects

- Objects store their data by themselves and encapsulate them for protection from other objects.



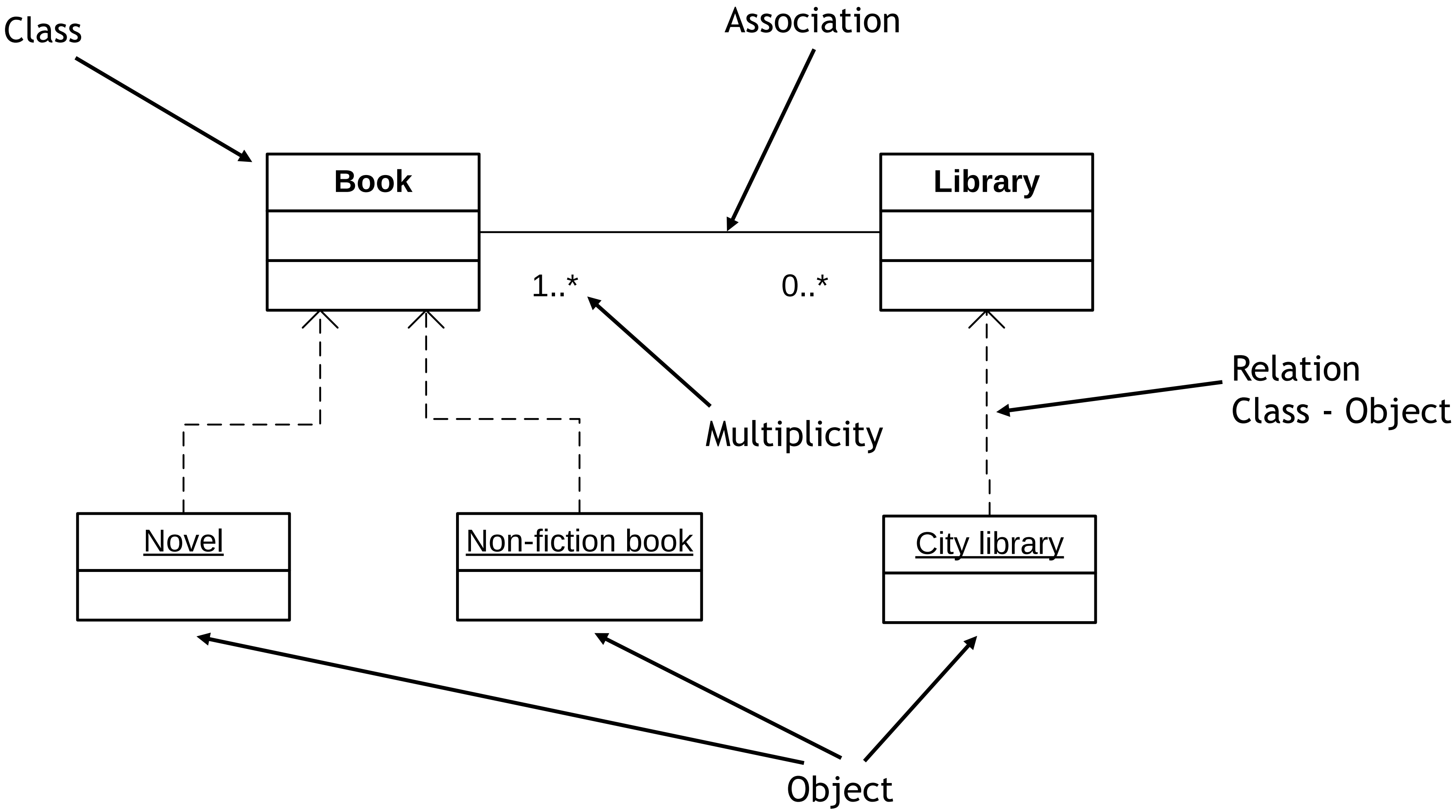
Object-Oriented Software Development

- Consideration of software as collection of interacting objects that work together in order to accomplish tasks.
  - Objects – things in a computer system that can respond to messages.
  - Conceptually, no processes, programs, data entities, or files are defined – just objects.

Basic OO Elements

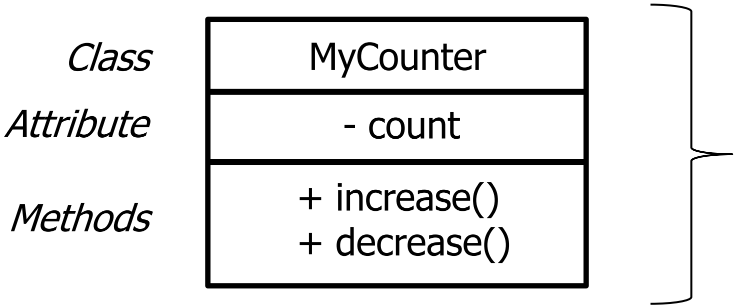
- Class
  - A class is a template for an object. It contains variables, constants and methods.
- Object
  - Objects are instances of classes, which exist during runtime. Multiple objects can be instantiated from a single class.
- Association
  - Relation between classes or objects
- Instantiation
  - Creation of objects according to the template of a class during runtime

Basic OO Elements



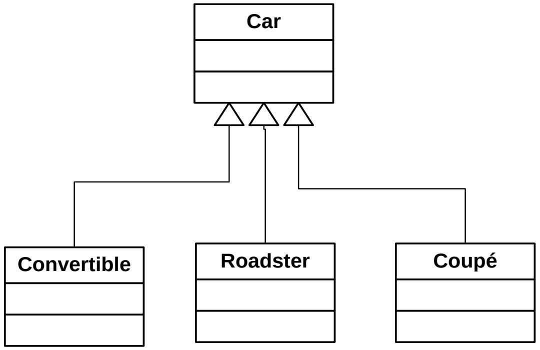
Basic OO Concepts

- Encapsulation
  - Data is stored in an object and can only be accessed via the offered methods.



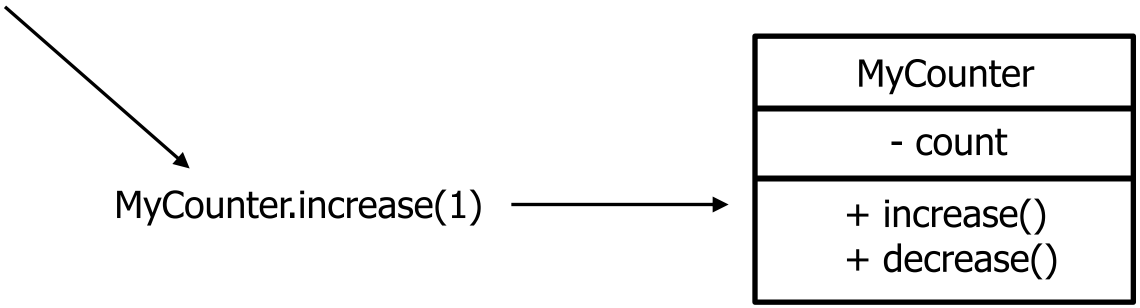
Increasing/decreasing the “count” property only works by sending a message to the “increase” or “decreasing” operation.

- Inheritance
  - Classes can inherit attributes or methods from other classes. The bequeathing class is called “super class” or “parent class”. The inheriting class is called a “subclass”.



Basic OO Concepts

- Messages
  - A message is sent to an object in order to instruct it to call a method.



MyCounter.increase(1)

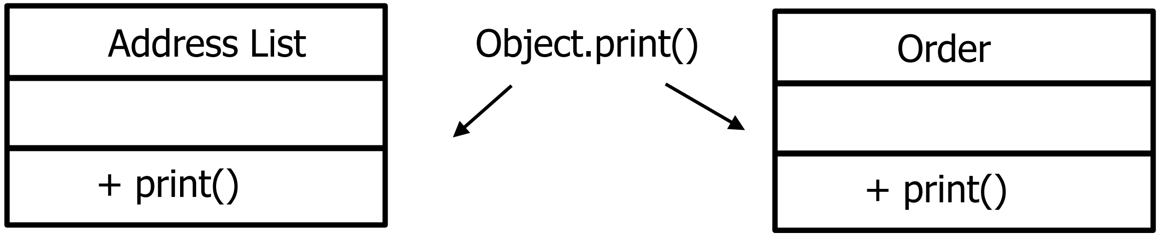
MyCounter

- count

+ increase()
+ decrease()

Polymorphism

- If a message is sent to objects of different classes, these objects return different results, as the called method can be implemented differently for each object.
- For instance, the message “Print” sent to the objects “Address List” and “Order”



Address List

+ print()

Object.print()

Order

+ print()

OO Terminology and Concepts

- Object-oriented Analysis (OOA)
- Object-oriented Design (OOD)
- Object-oriented Programming (OOP)

Object-Oriented Analysis (OOA)

- OOA describes a system as a group of interacting objects, generating a conceptual model within a problem domain.

- This results in a description of how the software is required to behave.

- The conceptual model does not describe any implementation details. Those are developed in the design phase.

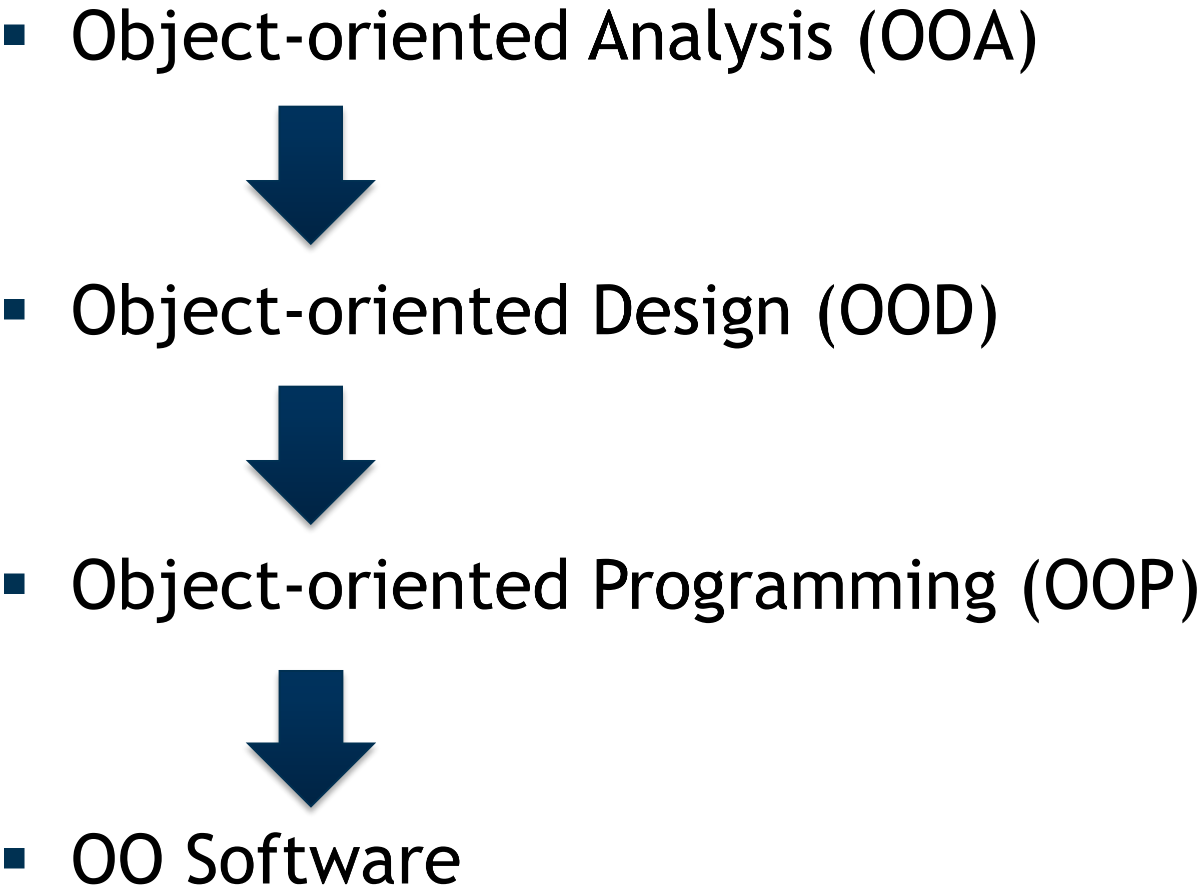
Object-Oriented Design (OOD)

- Takes the conceptual model generated by object oriented analysis as input.
- Refines each object type to be implemented with a specific language according to its environmental context
- Takes into account the chosen architecture, technological and environmental constraints
- Typical Output: Class-Diagram

Object-Oriented Programming (OOP)

- OOP is a programming paradigm for software
- It centres around the concept of “Objects”, which consist of data structures and methods
- It takes the results of the OOD as input
- OO languages: Java, C++, C#.NET, VB.NET

OO Development Process



- Object-oriented Analysis (OOA)
- Object-oriented Design (OOD)
- Object-oriented Programming (OOP)
- OO Software

Agenda

- Object-Oriented Approach
- Unified Modelling Language (UML)
- Model-Driven Development and Architectures

Unified Modelling Language (UML)

- Modelling language developed by Booch, Jacobson und Rumbaugh in 1996

- Standard of the OMG (Object Management Group)
- Current Version: 2.5 (March 2015)

- Standardisation …
  - of different object-oriented notations and
  - of methods through all phases of the software development

by using different types of models (data-oriented, object-oriented, process-oriented, etc.).

UML Concept

- Supports analysis and design of object-oriented software systems

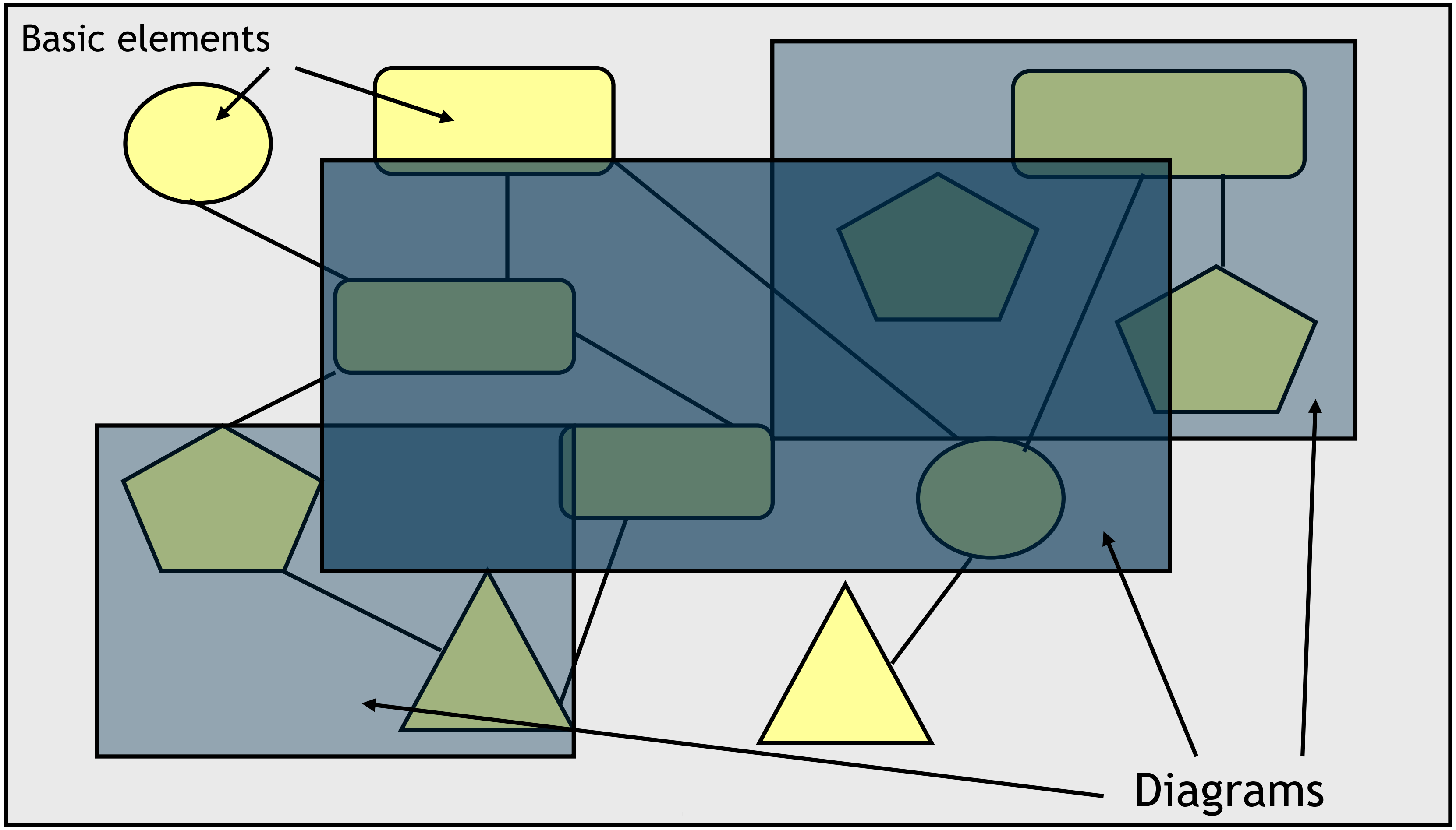
- UML includes multiple Views on a system
  - Each View specifies and documents a system from a different perspective.
  - Each View is supported by one or more diagrams.

- UML is not a process model → UML does not define a process for creating UML models.

UML Structure

- Basic elements
  - Object-oriented notation elements
  - Additional elements to describe the modelled system (e.g. activities, actor, etc.)
- Diagrams
  - Composition of notation elements
  - Represents a certain View on a system
- Complete model
  - The complete model is based on the basic elements.
  - Different Views on the complete model by different diagram types

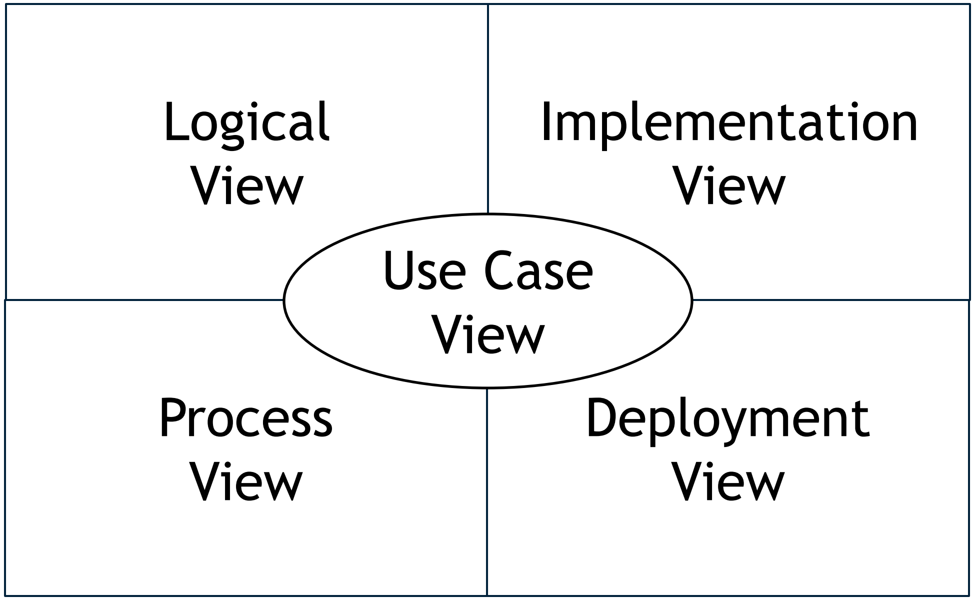
UML Structure



Complete model

UML Views

- Use case view
- Logical view
- Implementation view
- Process view
- Deployment view



Source: Hitz et al., 2015

Use Case View

- Describes high level functionalities of a system
- Used by stakeholders, designers, developers and testers
- Represented by use case diagrams
- Serves as the basis for other views

Logical View

- Describes functionalities to be designed and implemented
- Describes static and dynamic aspects of a system
- Mostly used by designers and developers
- Represented by class diagrams, object diagrams (static view), state diagrams, interaction and activity diagrams (dynamic view)

Implementation View

- Describes the organisation of software components
- It divides the logical entities into actual software components
- Represented by component diagrams
- Mostly used by developers

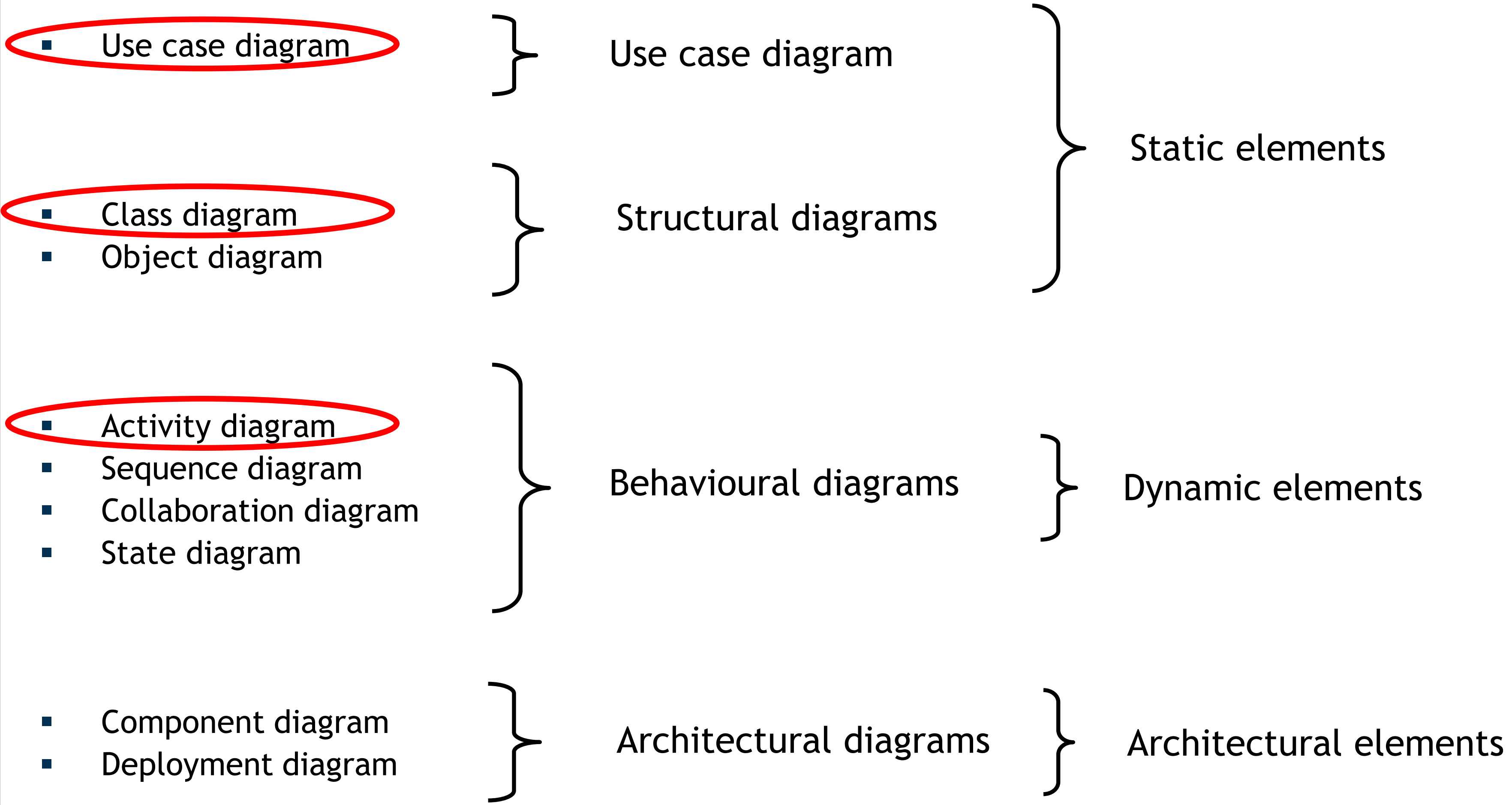
Process View

- Describes processes in a system
- Mostly used by developers and testers
- Represented by state, interaction and activity diagrams
- Supports concurrency and handling of asynchronous events

Deployment View

- Describes physical architecture and assignment of components to architectural elements
- Mostly used by designers, developers and managers
- Represented by package, component and deployment diagrams

UML Diagrams Examples



- Use case diagram
- Class diagram
- Object diagram
- Activity diagram
- Sequence diagram
- Collaboration diagram
- State diagram
- Component diagram
- Deployment diagram

Use case diagram

Structural diagrams

Behavioural diagrams

Architectural diagrams

Static elements

Dynamic elements

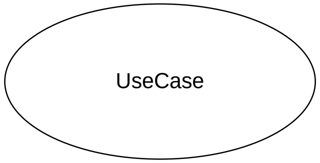
Architectural elements

Use Case Diagram

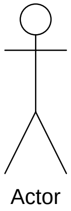
- Use cases describe the functionality, which a system has to provide
- The sum of all “Use cases” comprises the technical requirements of a system.
- Use cases define the interfaces between a user and the system
- Specification is developed together with the client/customer

Use Case Diagram Notation Elements

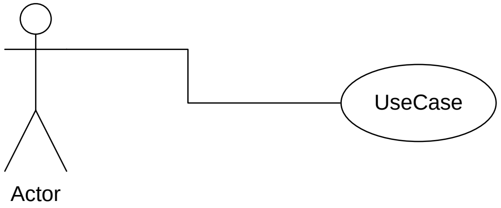
- Use Case
  - Representation of a sequence of actions that provides value to an actor.



- User of the system

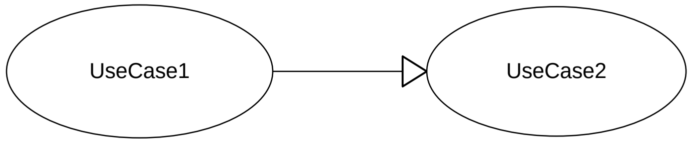


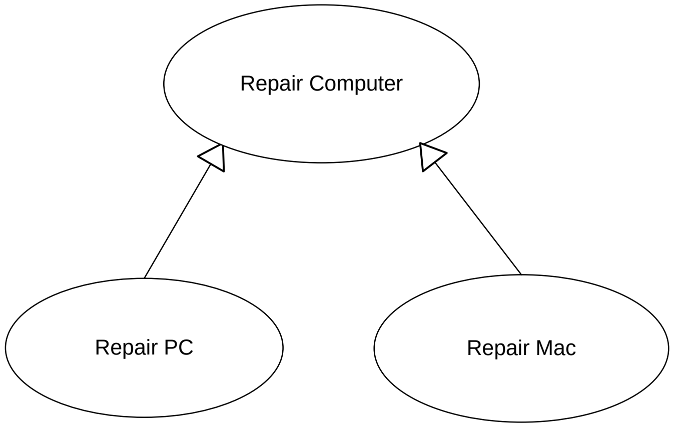
- Association
  - Interaction of an actor with a use case



Use Case Diagram Notation Elements

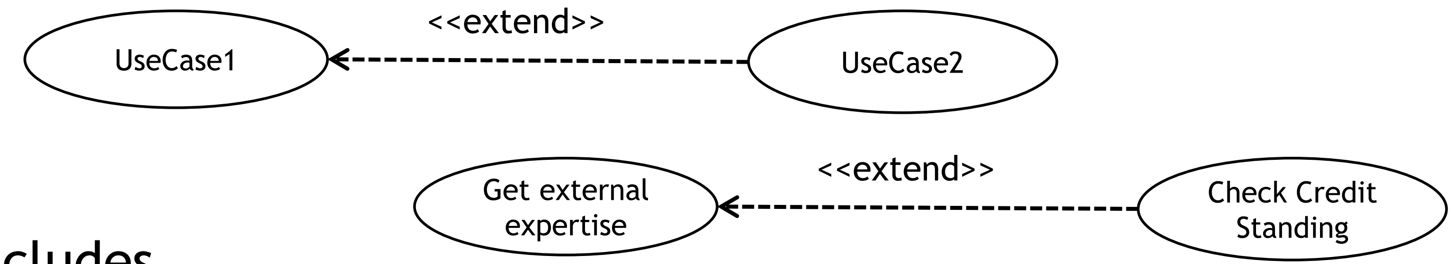
- Generalisation
  - Generalisation of Use Cases
  - UseCase2 generalises the behaviour of UseCase1



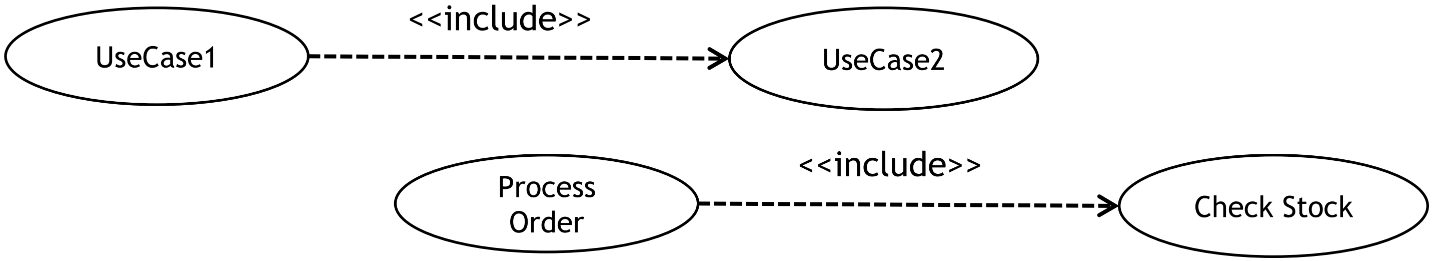


Use Case Diagram Notation Elements

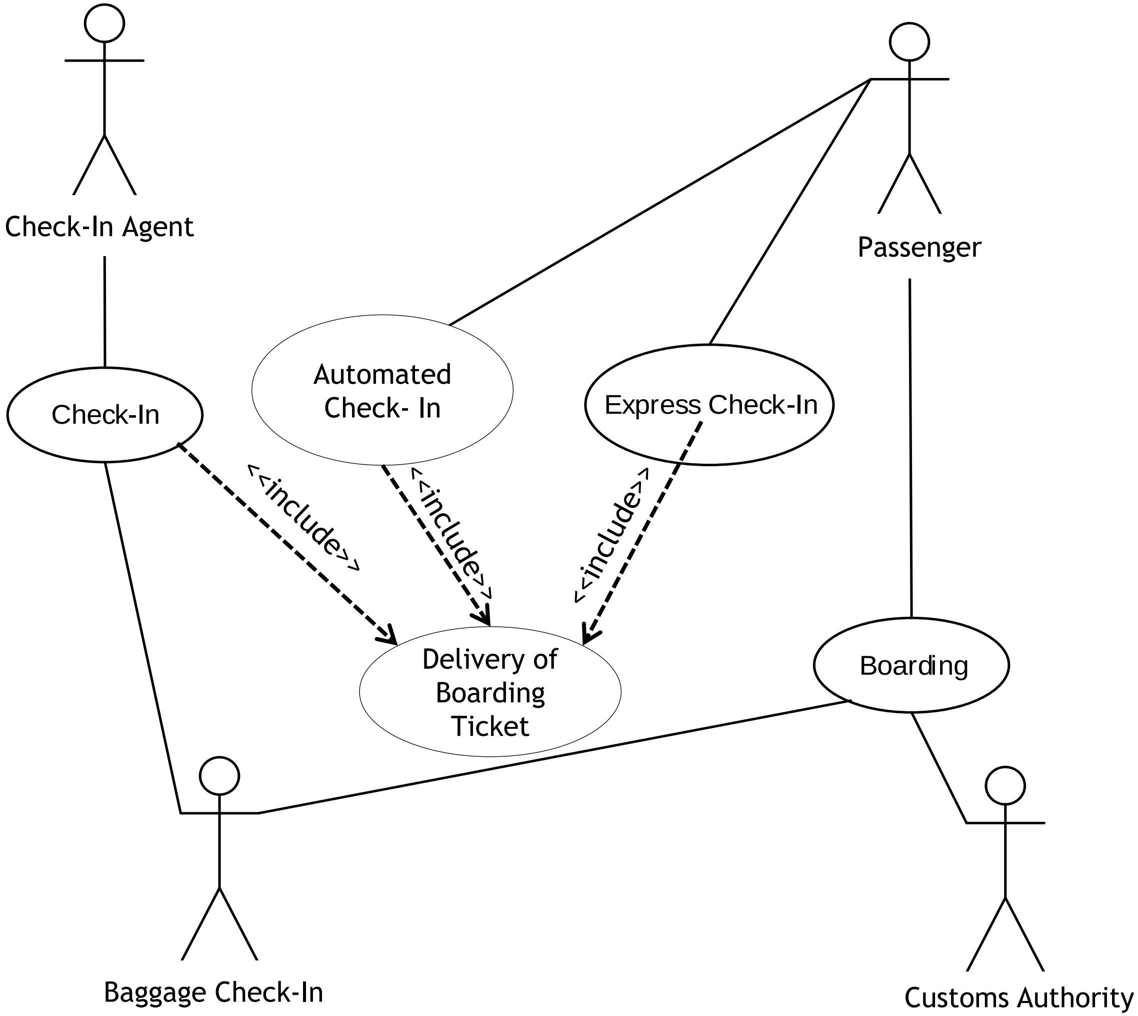
- Extends
  - Extends a Use Case
  - UseCase2 extends UseCase1



- Includes
  - Inclusion of a Use Case
  - UseCase1 includes the behaviour of UseCase2



Use Case Diagram (Example)



Structural Diagrams

- Class diagrams
  - Representation of the static structure of a software system
  - Description of logical relations between structural elements
  - No activity or control logic
- Object diagrams
  - Instances of a class diagram
  - „Snapshot“ of a system during runtime

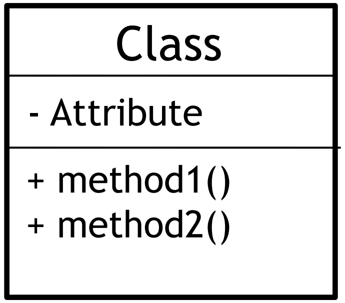
UML Class

- Classes are represented by rectangles, which include the name of the class, its attributes and methods.

- The class name is in singular and starts with an upper case letter.

- Attributes and methods are separated by horizontal lines.

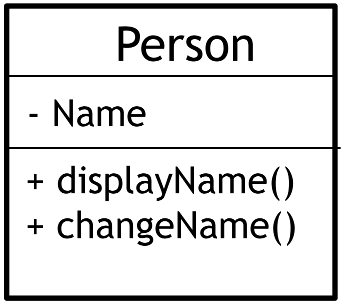
- „+/-“: Attribute/Method is public/private



Class

- Attribute

+ method1()
+ method2()



Person

- Name

+ displayName()
+ changeName()

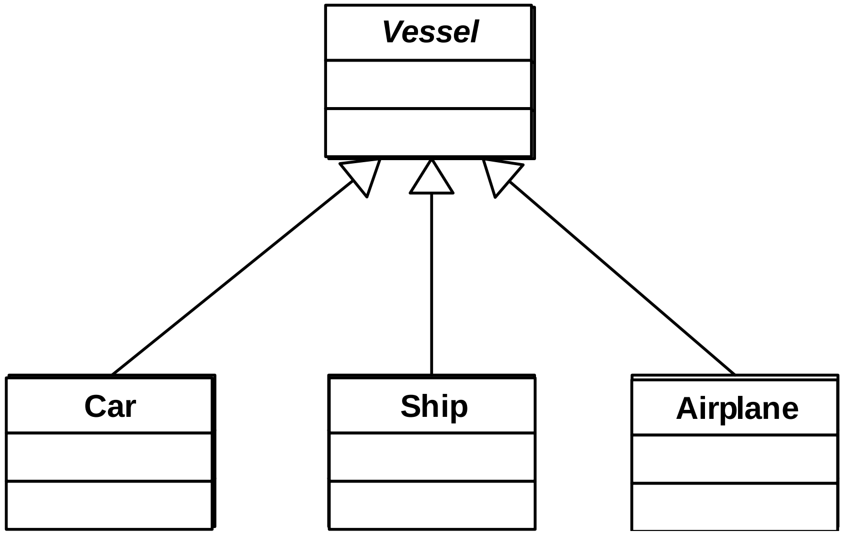
UML Class

- Class attributes
  - Class attributes belong to the class, not to the object.
  - Class attributes have the same value for all instances (objects). For instance, attribute „Number“ to count the number of created objects for a class.
  - Class attributes are underlined in the class diagram.

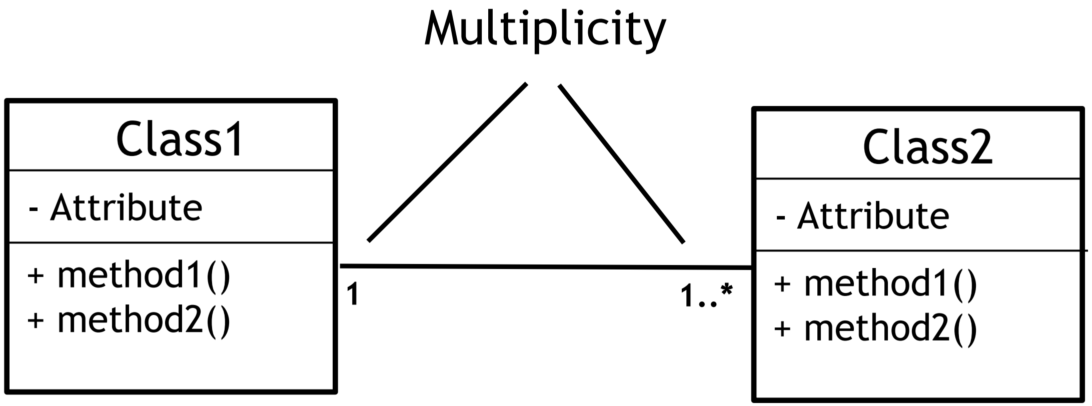
- Class methods
  - Class methods are executed within the class not on the object.
  - E.g. „count number of created objects of the class“
  - The class method is underlined in the class diagram.

Abstract Classes

- Definition / aggregation of common properties
- An abstract class does not allows objects to be instantiated.
- Template to create subclasses
- Abstract methods get “overwritten” by default
- The name of abstract classes is written in italic.



Associations

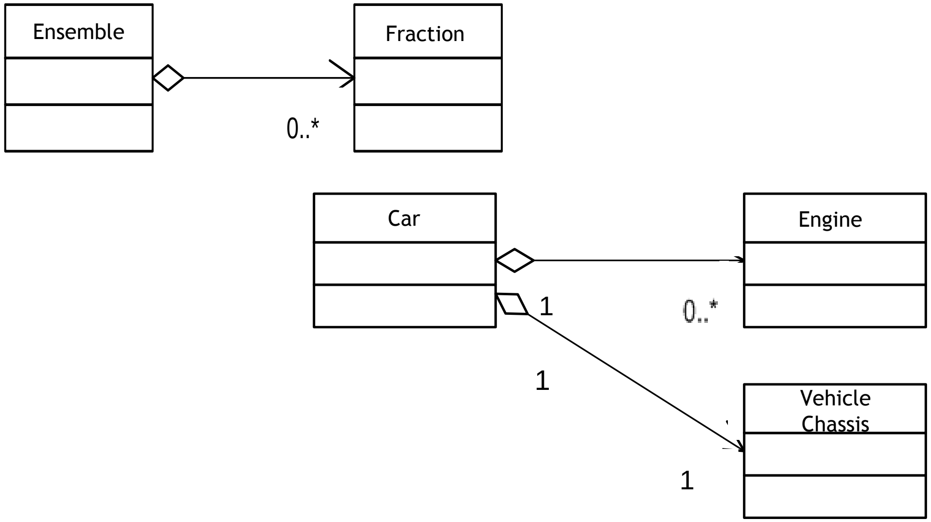


- Describes the relationship between two classes
- It is represented by a line connecting the two classes.
- The multiplicity min..max attached to the association defines the minimal or maximal number of associations between the objects of the two classes.

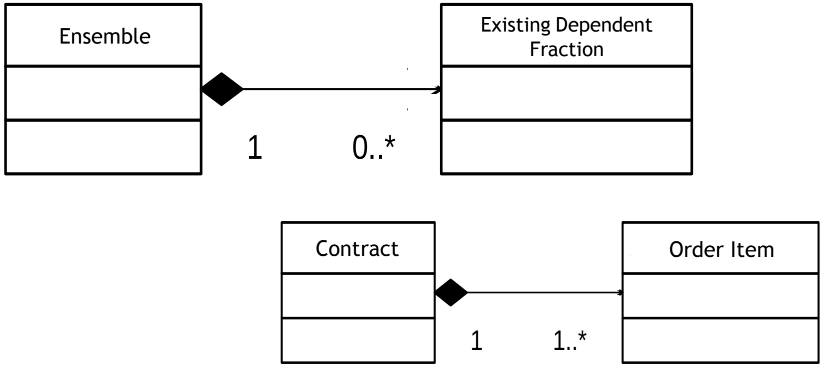
(*) denotes any number of objects.

Associations

- Aggregation
  - Denotes a „has a“ relationship

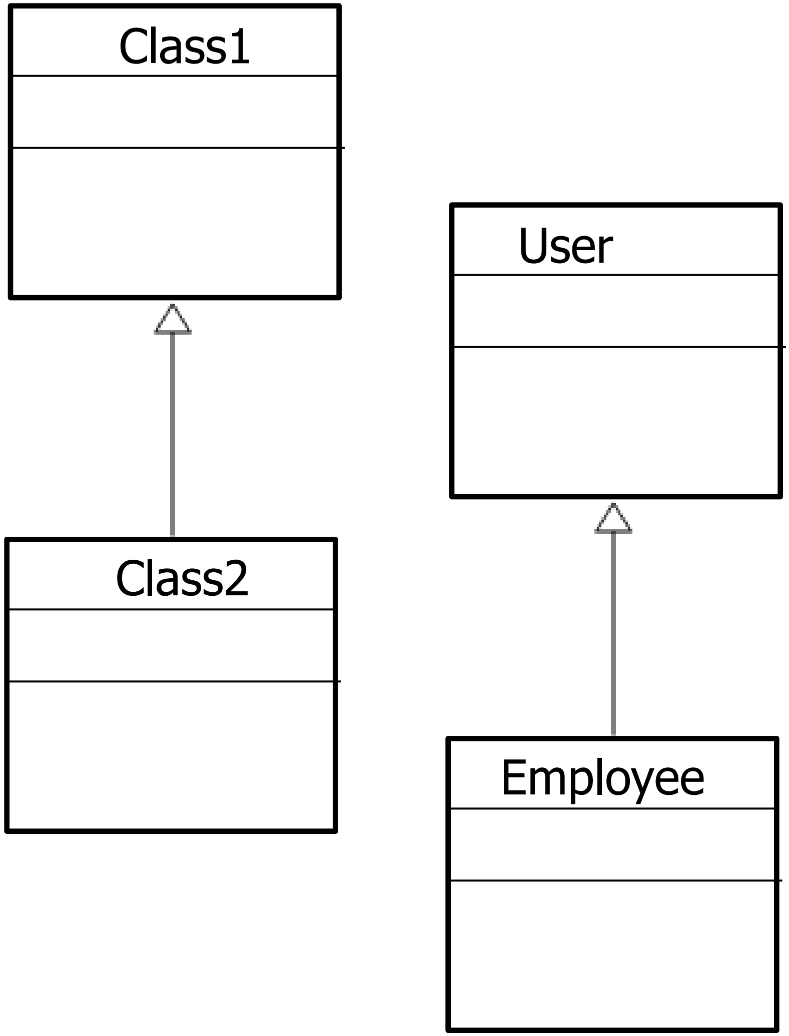


- Composition
  - Composition is a stronger variant of the aggregation
  - Denotes an “owns a” relationship



Inheritance

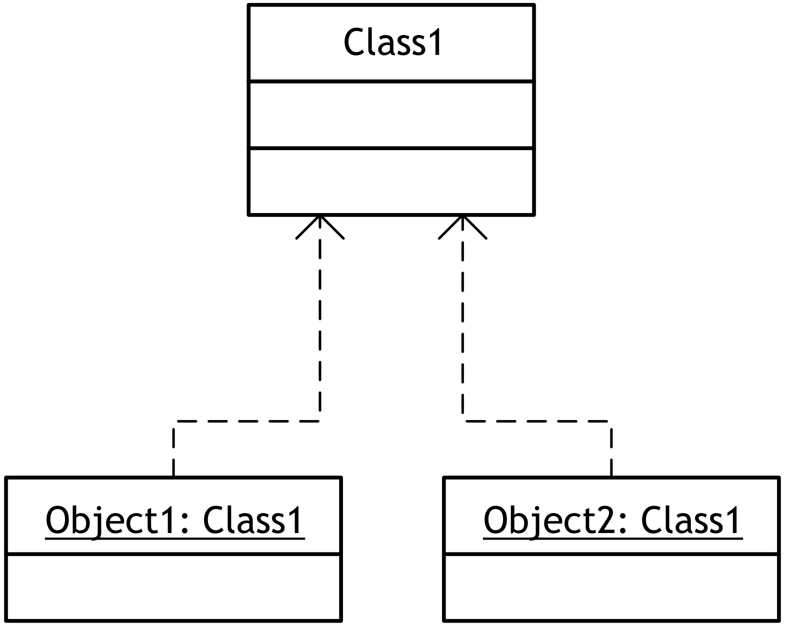
- Denotes an relation between parent class and subclass
- Is represented by a line with an empty arrow at the end, pointing towards the parent class
- Class2 inherits from Class1.
- Purpose:
  - Reuse code, by objects which can be based on previously created objects



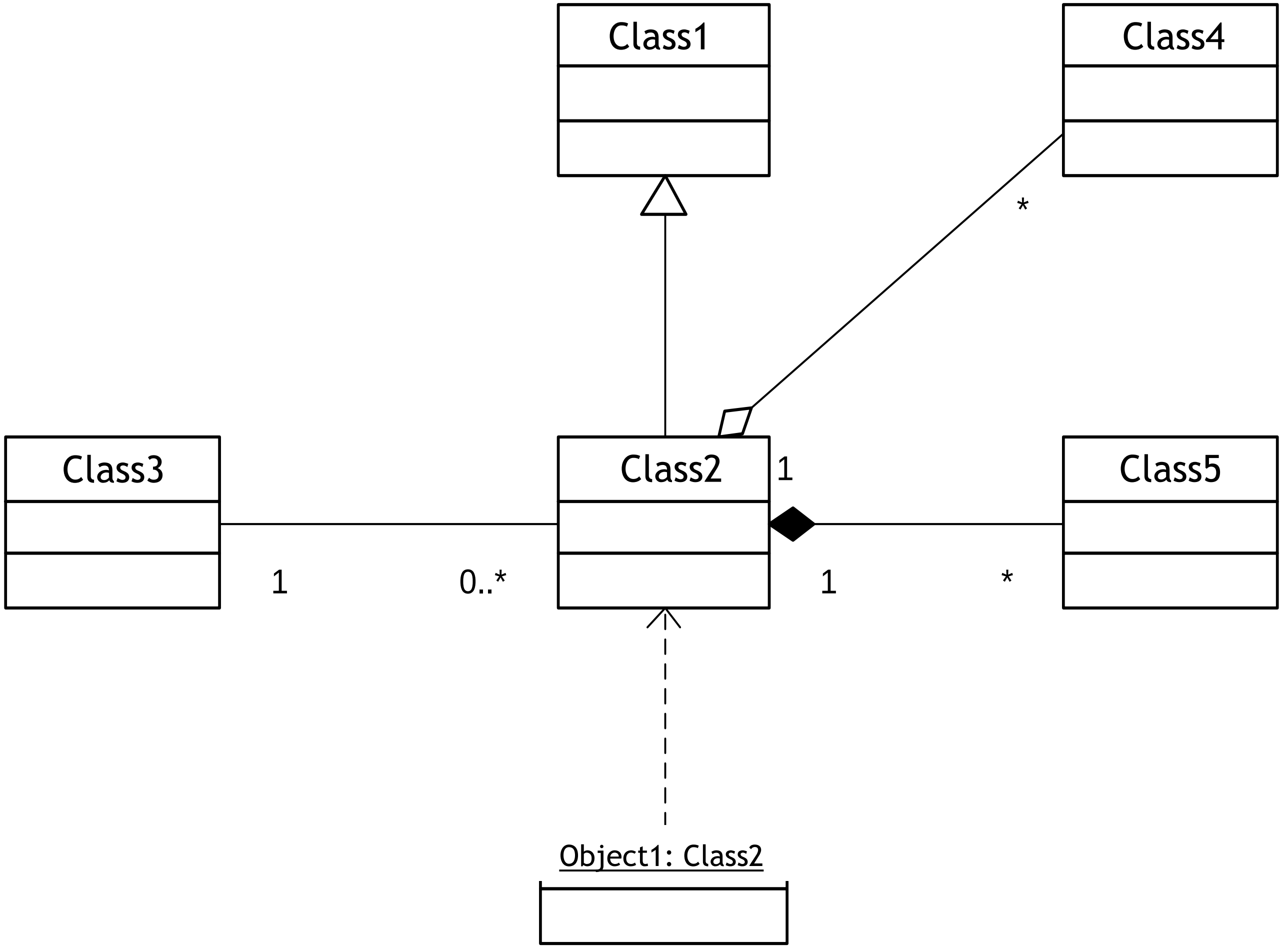
Instantiation

- Representation of the relation “class-object“
- An object is an instance of a class.

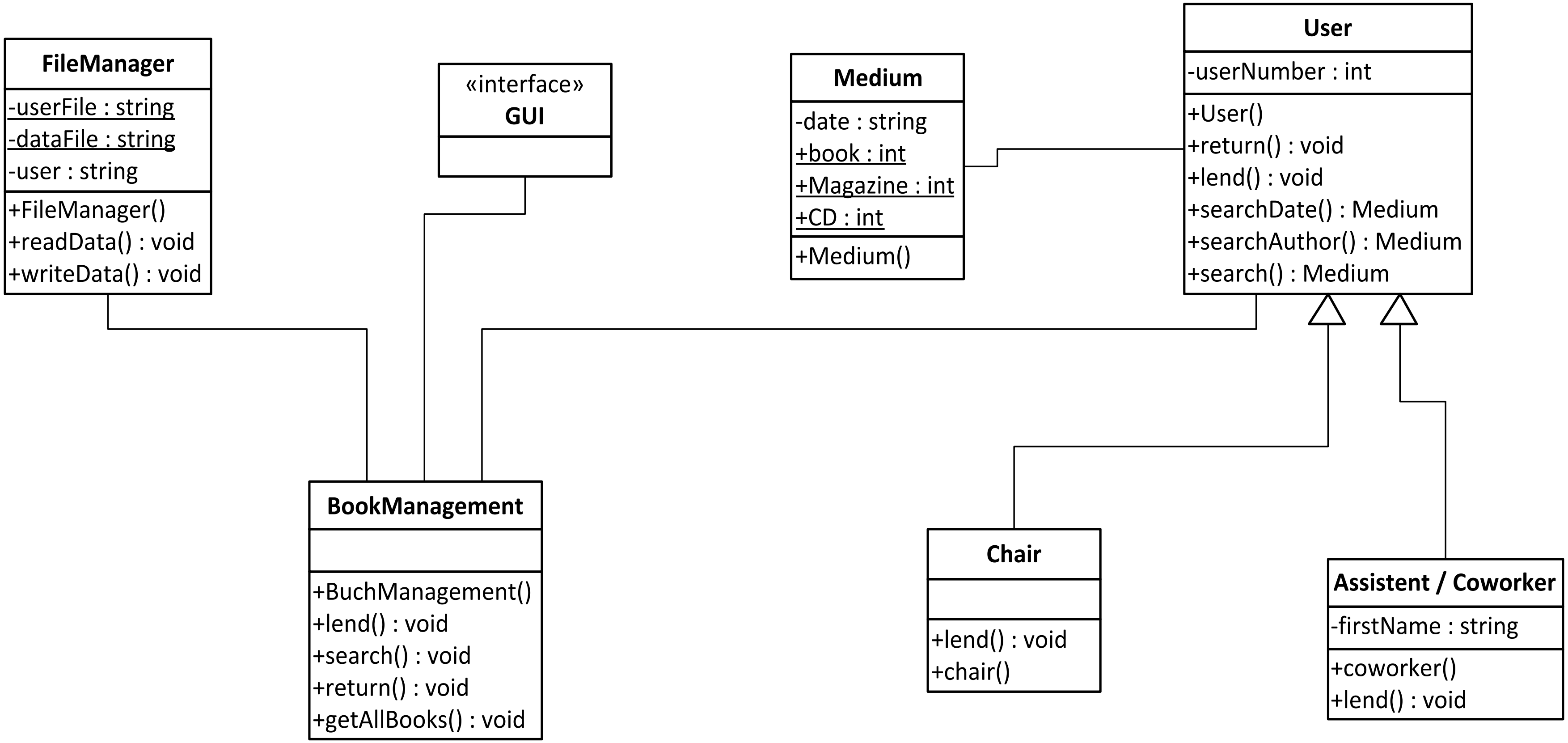
- Class
  - Attributes
  - Methods
- Object
  - Attribute values
  - Messages



Class Diagram



Class Diagram (Example)



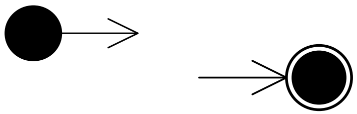
Activity Diagram

- Activity diagrams are used to model workflows in a system.
- Central element “Activity”: An activity is any kind of action.
- Activities are structured by responsibilities.
- Different views:
  - Conceptional View
    - e.g. business processes
  - Implementation View
    - e.g. methods of objects

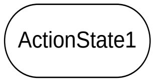
Activity Diagram Notation Elements

Notation elements

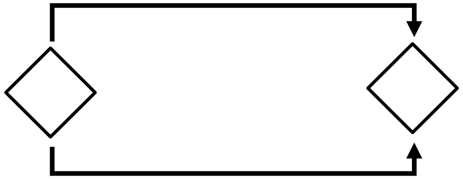
- Initial state/final state



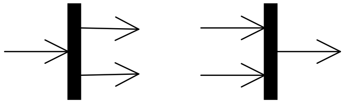
- Activity



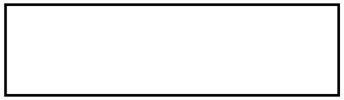
- Decision



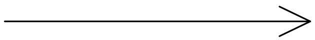
- Split/join



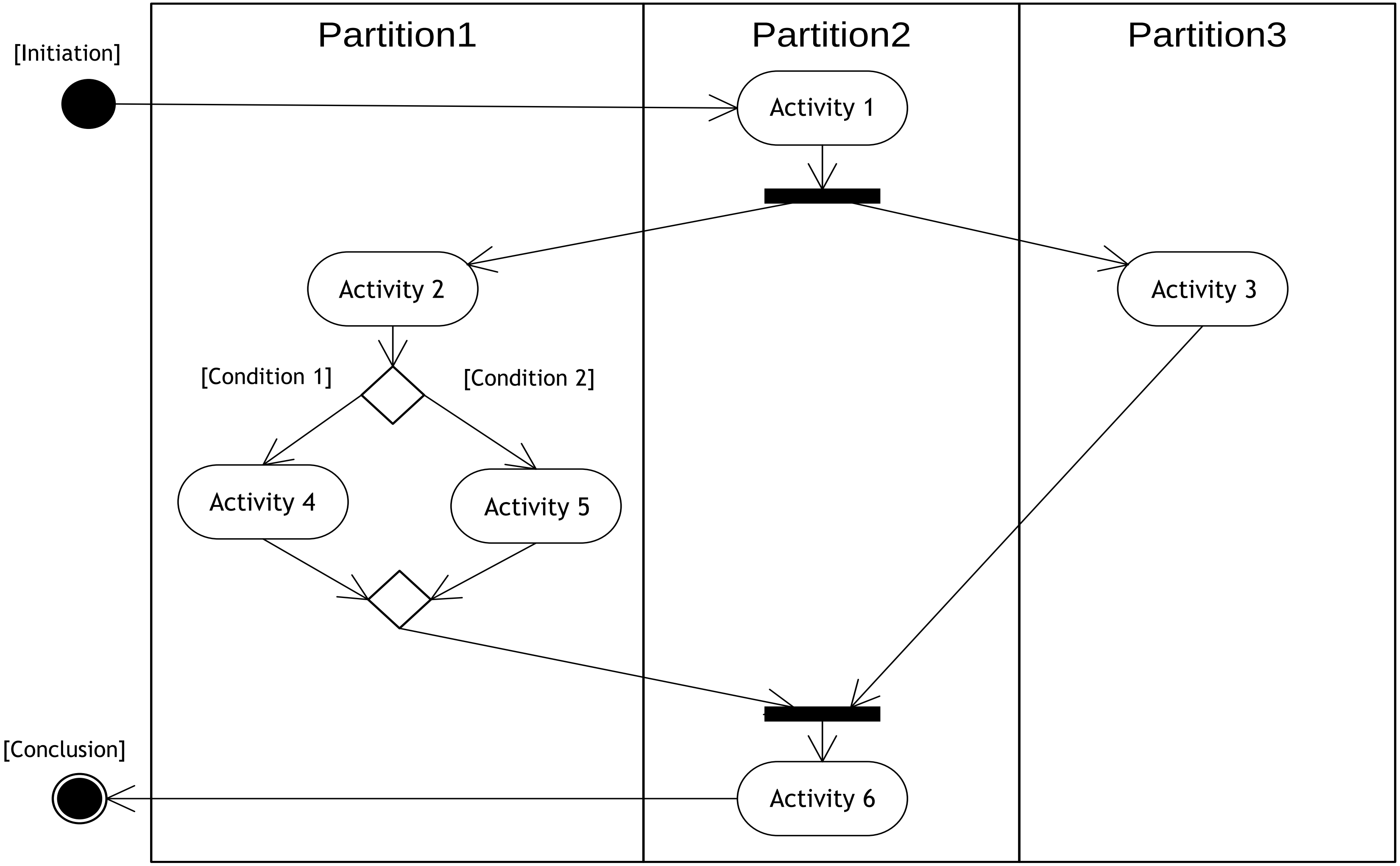
- Responsibility



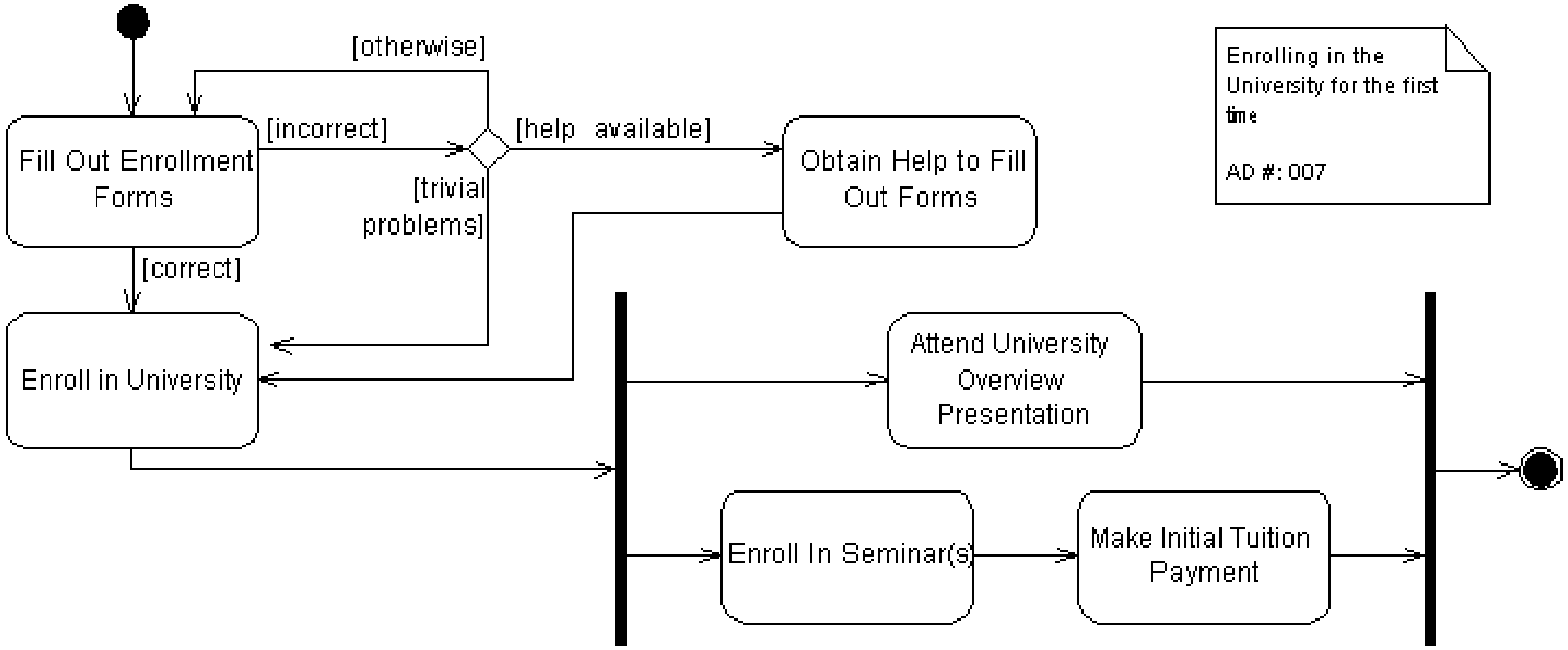
- Activity flow



Activity Diagram



Activity Diagram (Example)


[otherwise]
[incorrect]
[help available]
Fill Out Enrollment Forms
[trivial problems]
Obtain Help to Fill Out Forms
[correct]
Enroll in University
Attend University Overview Presentation
Enroll In Seminar(s)
Make Initial Tuition Payment

Enrolling in the University for the first time

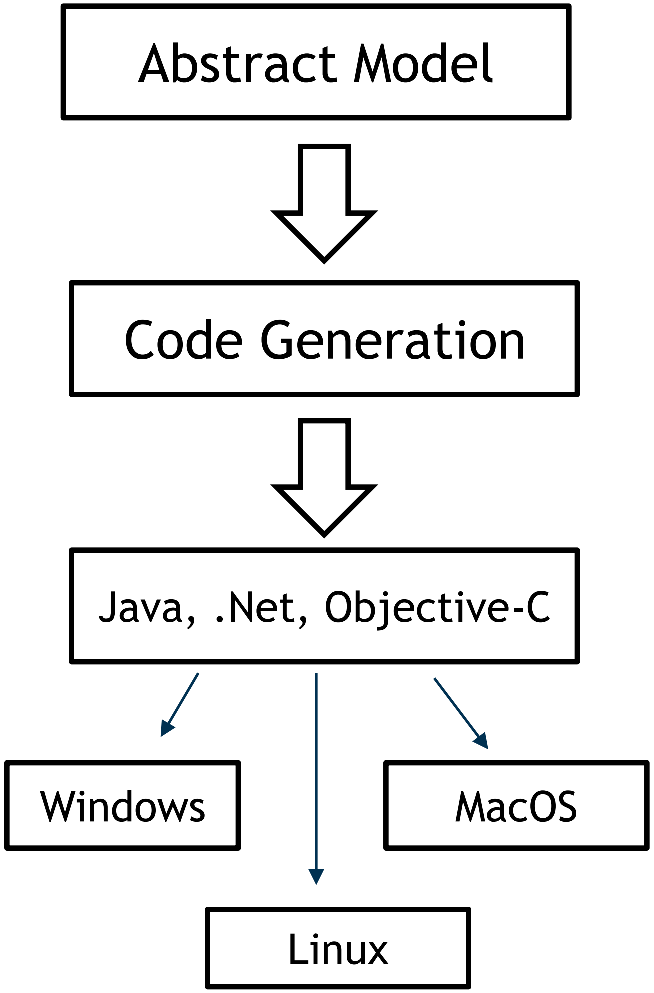
AD #: 007

Agenda

- Object-Oriented Approach
- Unified Modelling Language (UML)
- Model-Driven Development and Architectures

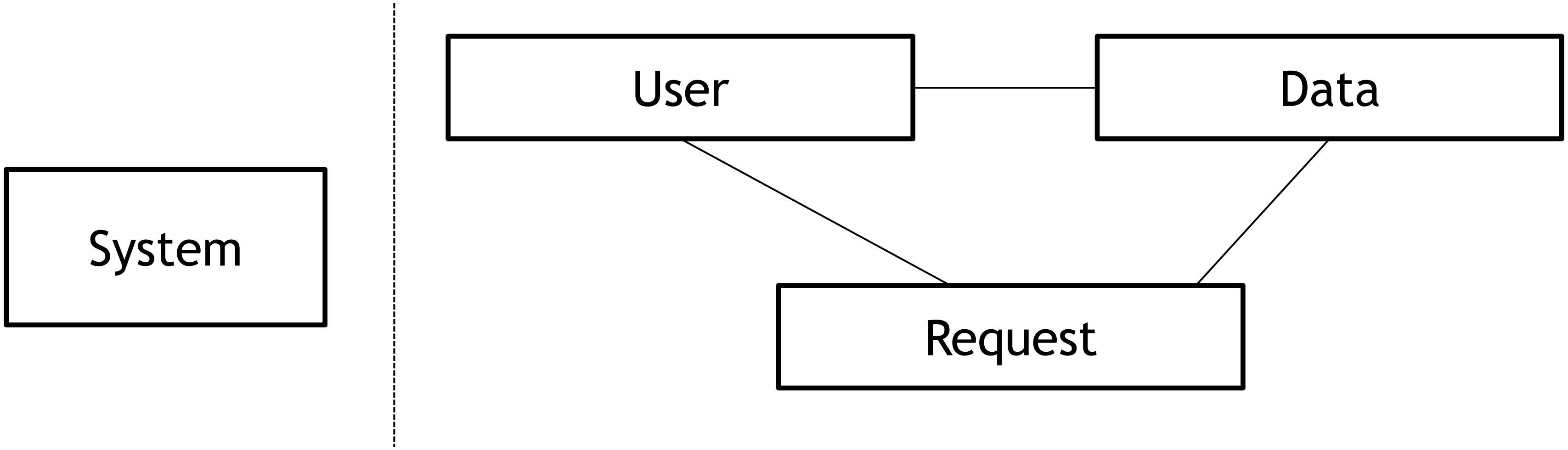
Model-driven Development (MDD)

- MDD is a concept for the development of software
- The software system is described by an abstract model (e.g. based on UML)
- The abstract model is typically independent from the target programming language, OS platform or other any underlying technology
- The abstract model allows an automatic transformation into code for multiple target OS platforms
- The resulting code may vary from skeleton classes to complete software products



What is an Abstract Model?

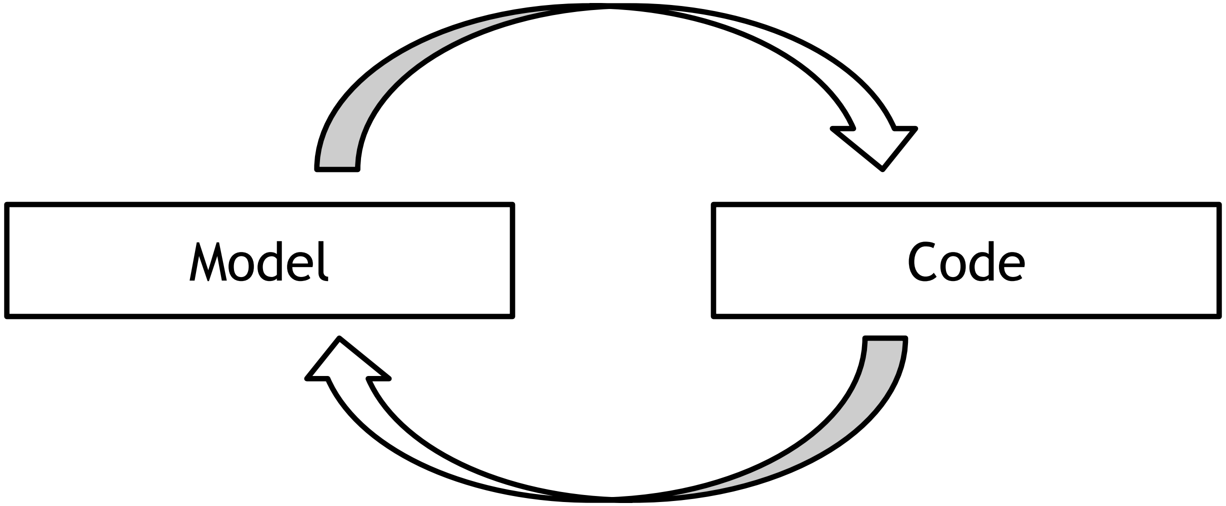
- Abstraction of the real software system (not the real world)
- Comprised of only the relevant aspects of a system – irrelevant ones are ignored
- Different abstraction levels are possible



Round-Trip Engineering

- Modifications to the model can automatically be transformed into code and vice versa.

Forward Engineering



Reverse Engineering

Automation in the Development Process

- MDD promotes automation within the development process.
- Automated analysis and verification of model
  - Since models do not contain implementation details they are easier to analyse.
- Automated code generation from model, which guarantees the conformance to the model
- Runtime monitoring based on a model
  - Runtime monitoring makes sure that the implementation follows the behaviour specified in the model.
- Automated test generation
  - Models can be used to generate test cases for the implementation.

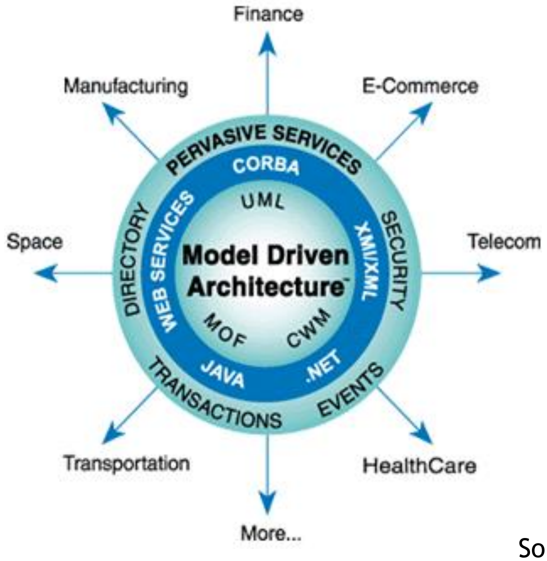
Benefits of MDD

- Reduced development time
- The model is timeless: It will age with the domain and not with the technology.
- Improved documentation of the software system
  - A model is a better documentation than code
  - Improved readability – especially by non IT-personnel
  - Because of automated generation always consistent with the code
- The system can be adjusted more easily.
- Platform and programming language independence
- …

Source: Based on Scheier Software Engineering 2011

Model-Driven Architecture (MDA)

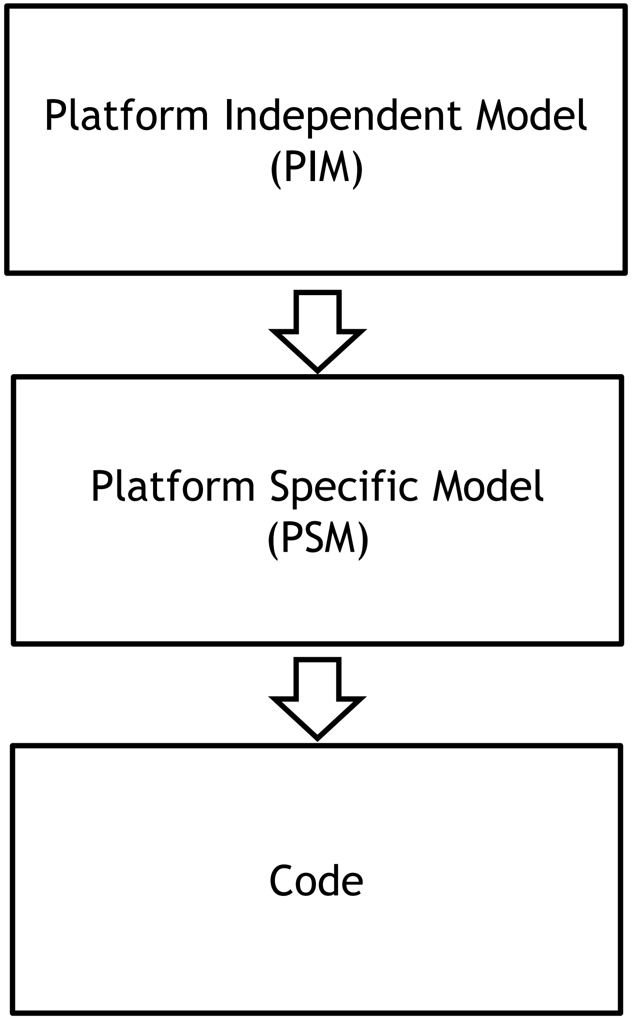
- MDA was introduced by the Object Management Group (OMG).
- MDA separates the business and application logic from the underlying implementation platform.
- MDA is a forward engineering approach where first abstract model diagrams are developed which are later transformed to code.
- The goal of MDA is to separate the conceptual design from the implementation architecture.



Source: OMG, 2011

Model-Driven Architecture Development Process

- Developers develop platform independent models (PIM) for the software (e.g. readable design models or UML).
- The platform independent models document the business functionality of a software – independent from the technology-specific code.
- After the target implementation platform was chosen, the platform independent models can automatically be translated to platform specific models (PSM).
- The platform specific models are used to guide the implementation for the chosen platform.



Platform Independent Model (PIM)

Platform Specific Model (PSM)

Code

MDA Benefits for the Software Lifecycle

- *Implementation:* MDA enables the integration of new target software platforms based on the existing design models.
- *Integration:* Integration is easier since both the implementation and the design models exists at the time of integration.
- *Maintenance:* The availability of the design in a machine-readable form gives developers direct access to the specification of the system, making maintenance much simpler.
- *Testing and simulation:* The design models can be validated against existing requirements and executable models can be used to simulate the behaviour of the system.

Literature

- Booch, G.; Rumbaugh, J.; Jacobson, I. (1999): Das UML-Benutzerhandbuch. Addison-Wesley
- Hitz et al. (2005): UML@Work: Objektorientierte Modellierung mit UML 2, d.punkt Verlag
- Johannes Scheier: Software Engineering, www.jug.ch/events/slides/061018_johannes_scheier.pdf
- OMG (2011): http://www.omg.org/gettingstarted/specintro.htm#MDA
- Stellmann, A.; Greene, J. (2011): Applied Software Project Management, O'Reilly Media Inc