



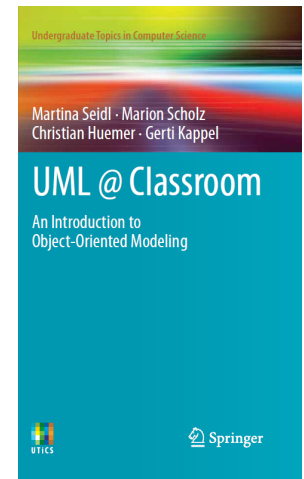
Business Informatics Group

Vienna University of Technology

# Object-Oriented Modeling

First set of diagrams extracted...

Slides accompanying UML@Classroom  
Version 1.0



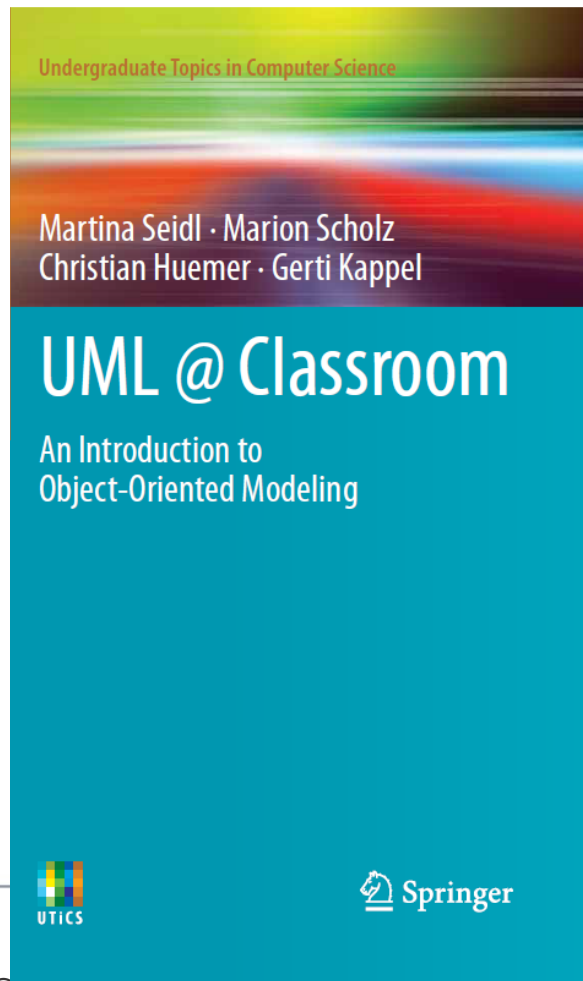
## Business Informatics Group

Institute of Software Technology and Interactive Systems  
Vienna University of Technology

Favoritenstraße 9-11/188-3, 1040 Vienna, Austria  
phone: +43 (1) 58801-18804 (secretary), fax: +43 (1) 58801-18896  
office@big.tuwien.ac.at, www.big.tuwien.ac.at

# Literature

- The lecture is based on the following book:



## **UML @ Classroom: An Introduction to Object-Oriented Modeling**

Martina Seidl, Marion Scholz, Christian Huemer  
and Gerti Kappel

Springer Publishing, 2015

ISBN 3319127411

- **Use Case Diagram**
- Structure Modeling
- State Machine Diagram
- Sequence Diagram
- Activity Diagram



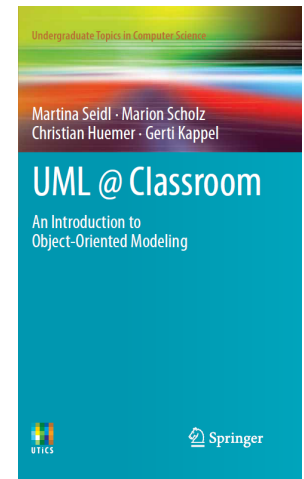
Business Informatics Group

Vienna University of Technology

# Object-Oriented Modeling

## Use Case Diagram

Slides accompanying UML@Classroom  
Version 1.0



### Business Informatics Group

Institute of Software Technology and Interactive Systems  
Vienna University of Technology

Favoritenstraße 9-11/188-3, 1040 Vienna, Austria  
phone: +43 (1) 58801-18804 (secretary), fax: +43 (1) 58801-18896  
office@big.tuwien.ac.at, www.big.tuwien.ac.at

# Content - Simplified

---

- Introduction
- Use cases
- Actors
- Relationships between use cases and actors
- Best practices (extract)
- Typical errors (extract)
- Notation elements

# Introduction

---

- The use case is a fundamental concept of many object-oriented development methods.
- Use case diagrams express the expectations of the customers/stakeholders
  - essential for a detailed design
- The use case diagram is used during the entire analysis and design process.
- We can use a use case diagram to answer the following questions:
  - What is being described? (The system.)
  - Who interacts with the system? (The actors.)
  - What can the actors do? (The use cases.)

# Example: Student Administration System

## ■ System

(what is being described?)

- Student administration system

## ■ Actors

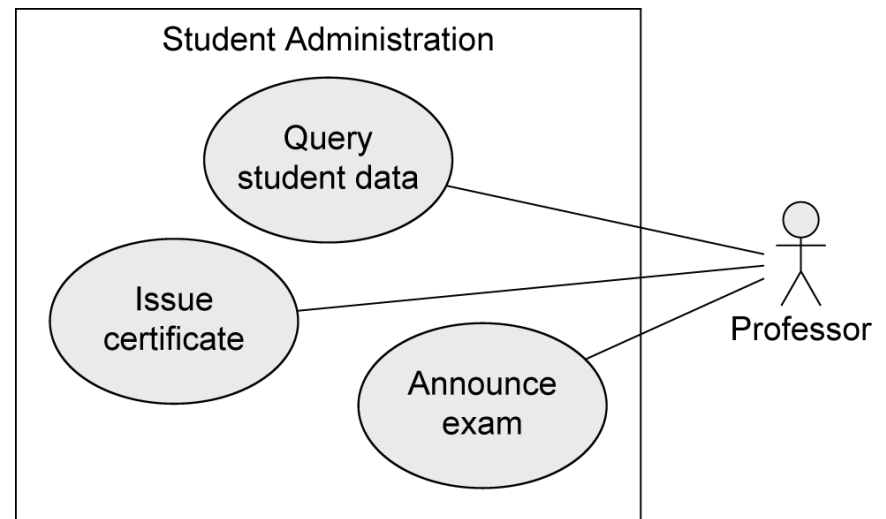
(who interacts with the system?)

- Professor

## ■ Use cases

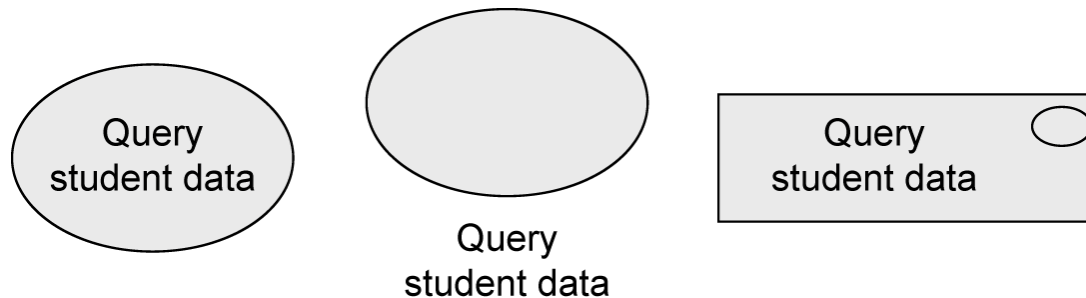
(what can the actors do?)

- Query student data
- Issue certificate
- Announce exam



# Use Case

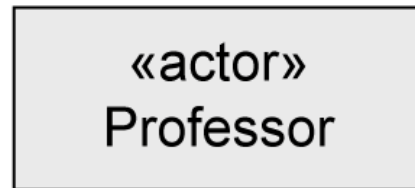
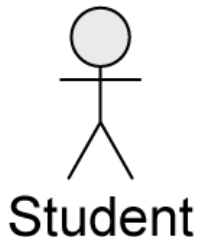
- Describes functionality expected from the system under development.
- Provides tangible benefit for one or more actors that communicate with this use case.
- Derived from collected customer wishes.
- Set of all use cases describes the functionality that a system shall provide.
  - Documents the functionality that a system offers.
- Alternative notations:





## Actor (1/3)

- Actors interact with the system ...
  - by **using** use cases, i.e., the actors initiate the execution of use cases.
  - by **being used** by use cases, i.e., the actors provide functionality for the execution of use cases.
- Actors represent roles that users adopt.
  - Specific users can adopt and set aside multiple roles simultaneously.
- Actors are not part of the system, i.e., they are outside of the system boundaries.
- Alternative notations:



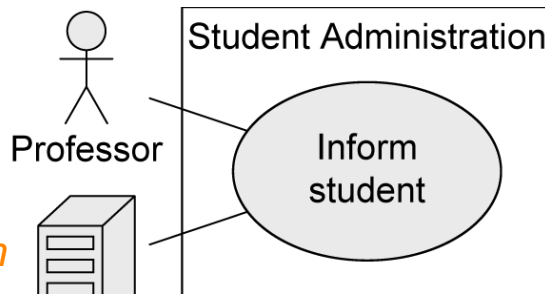




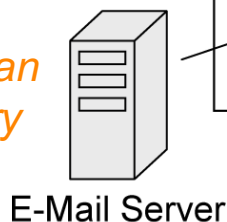
## Actor (3/3)

- **Human**
  - E.g., **Student, Professor**
- **Non-human**
  - E.g., **E-Mail Server**
- **Primary**: has the main benefit of the execution of the use case
- **Secondary**: receives no direct benefit
- **Active**: initiates the execution of the use case
- **Passive**: provides functionality for the execution of the use case
- **Example:**

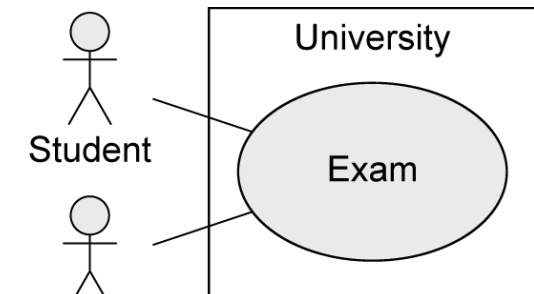
*Human  
Primary  
Active*



*Non-human  
Secondary  
Passive*



*Human  
Primary  
Active*

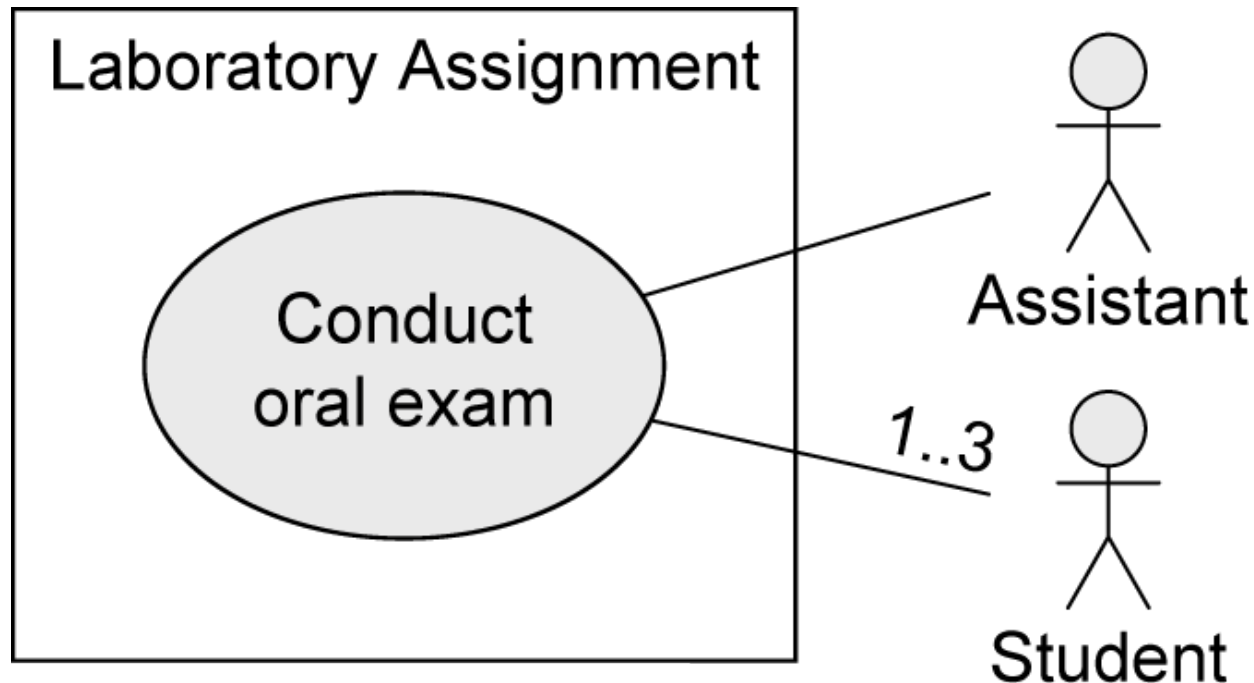


*Human  
Secondary  
Active*



# Relationships between Use Cases and Actors

- Actors are connected with use cases via solid lines (*associations*).
- Every actor must communicate with at least one use case.
- An association is always binary.
- Multiplicities may be specified.



# Best Practices

## Identifying Actors

---

- Who uses the main use cases?
- Who needs support for their daily work?
- Who is responsible for system administration?
- What are the external devices/(software) systems with which the system must communicate?
- Who is interested in the results of the system?

# Best Practices

## Identifying Use Cases

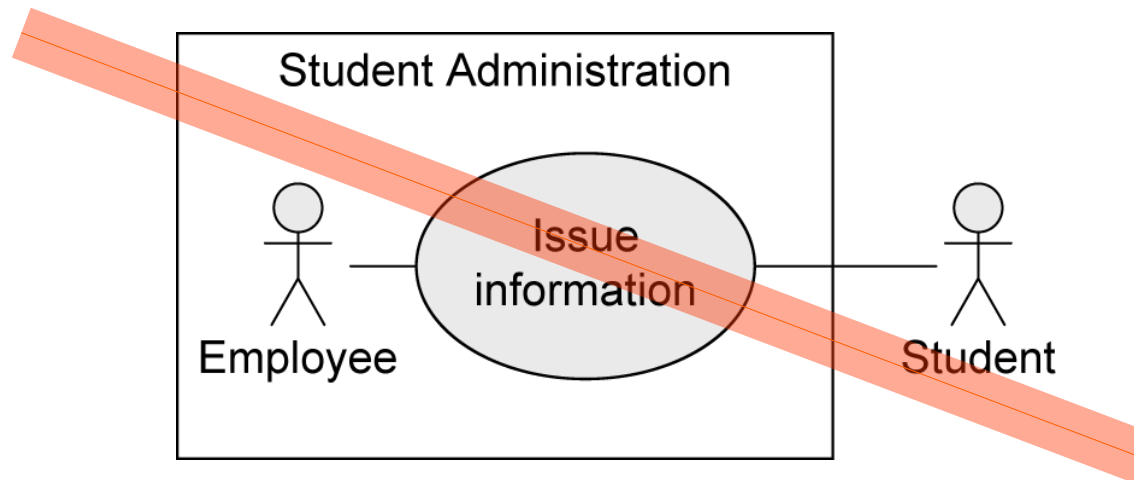
---

- What are the main tasks that an actor must perform?
- Does an actor want to query or even modify information contained in the system?
- Does an actor want to inform the system about changes in other systems?
- Should an actor be informed about unexpected events within the system?

# Best Practices

## Typical Errors To Avoid (2/5)

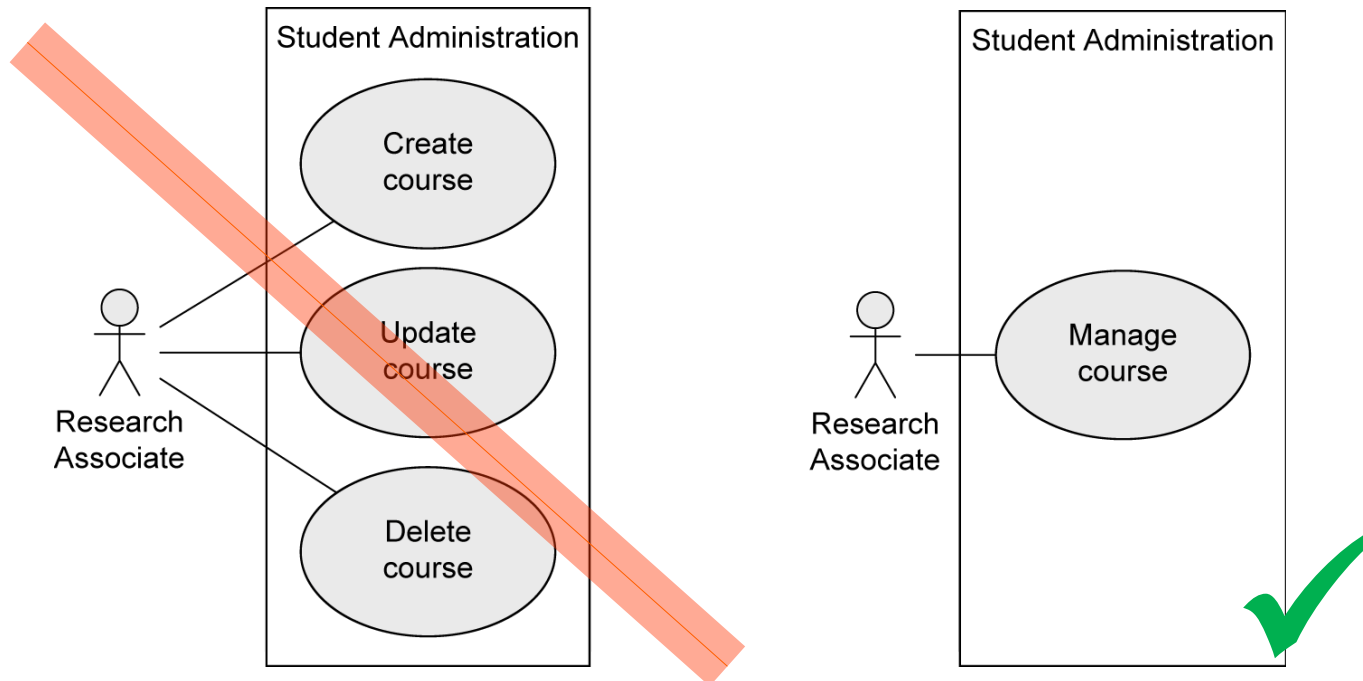
- Actors are not part of the system, hence, they are positioned outside the system boundaries!



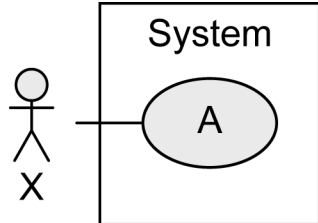

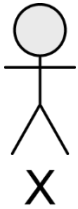
# Best Practices

## Typical Errors To Avoid (4/5)

- Many small use cases that have the same objective may be grouped to form one use case



## Notation Elements (1/2)

Name	Notation	Description
System		Boundaries between the system and the users of the system
Use case		Unit of functionality of the system
Actor		Role of the users of the system





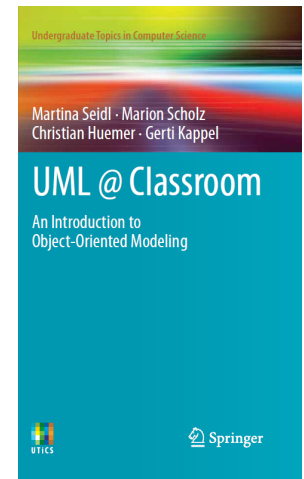
Business Informatics Group

Vienna University of Technology

# Object-Oriented Modeling

## Structure Modeling

Slides accompanying UML@Classroom  
Version 1.0



### Business Informatics Group

Institute of Software Technology and Interactive Systems  
Vienna University of Technology

Favoritenstraße 9-11/188-3, 1040 Vienna, Austria  
phone: +43 (1) 58801-18804 (secretary), fax: +43 (1) 58801-18896  
office@big.tuwien.ac.at, www.big.tuwien.ac.at

# Content

---

- Classes
- Attributes
- Operations
- Relationships
  - Binary Association
  - N-ary Association
  - Association Class
  - Aggregation
  - Generalization
- Creating a class diagram
- Code Generation



# From Object to Class

- Individuals of a system often have identical characteristics and behavior
- A class is a construction plan for a set of similar objects of a system
- Objects are instances of classes
- **Attributes:** structural characteristics of a class
  - Different value for each instance (= object)
- **Operations:** behavior of a class
  - Identical for all objects of a class  
→ not depicted in object diagram

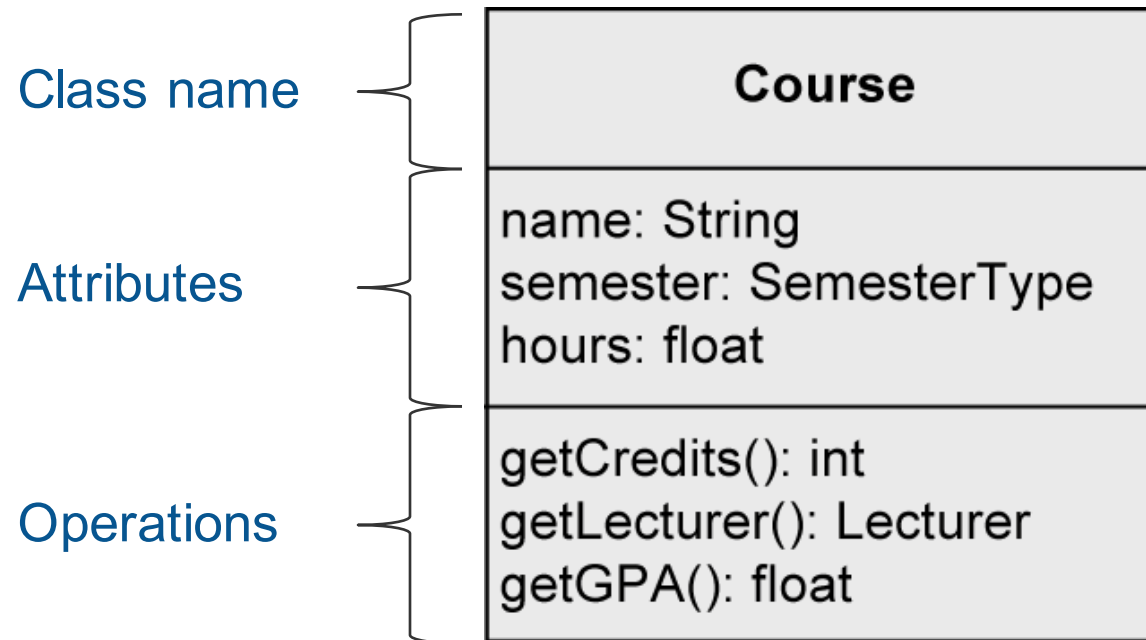
Class

Person
firstName: String lastName: String dob: Date

Object of that class

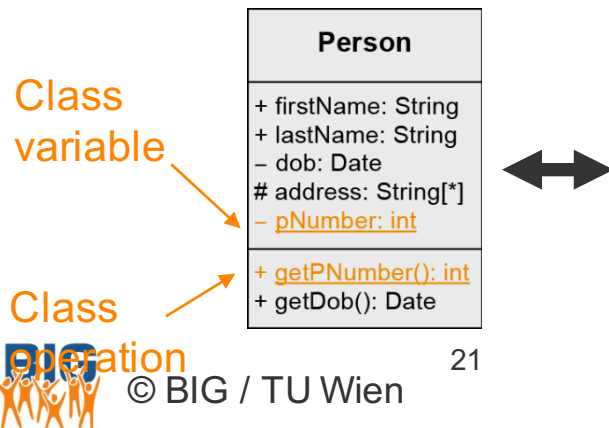
<u>maxMiller:Person</u>
firstName = "Max" lastName = "Miller" dob = 03-05-1973

# Class



# Class Variable and Class Operation

- Instance variable (= instance attribute): attributes defined on instance level
- Class variable (= class attribute, static attribute)
  - Defined only once per class, i.e., shared by all instances of the class
  - E.g. counters for the number of instances of a class, constants, etc.
- Class operation (= static operation)
  - Can be used if no instance of the corresponding class was created
  - E.g. constructors, counting operations, math. functions (sin(x)), etc.
- **Notation:** underlining name of class variable / class operation



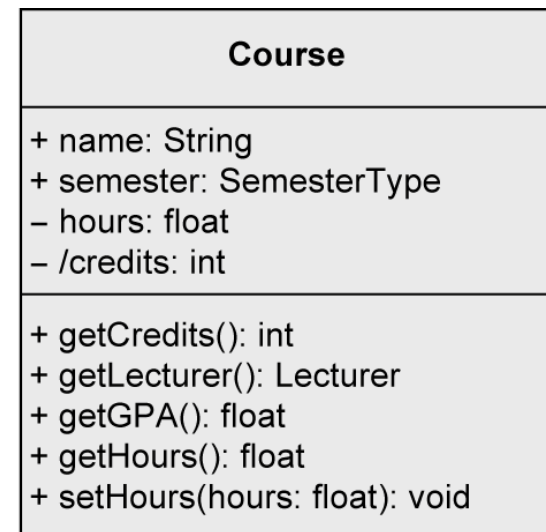
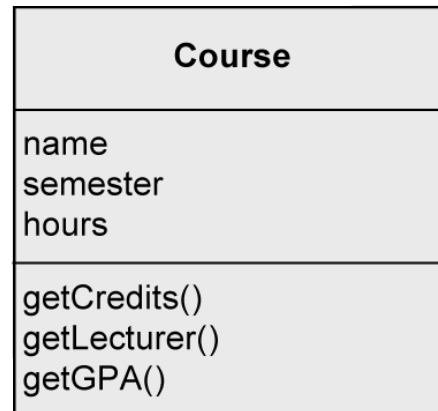
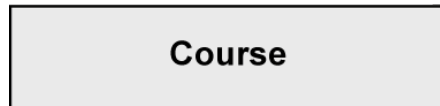
```
class Person {

    public String firstName;
    public String lastName;
    private Date dob;
    protected String[] address;
    private static int pNumber;
    public static int getPNumber() {...}
    public Date getDob() {...}

}
```

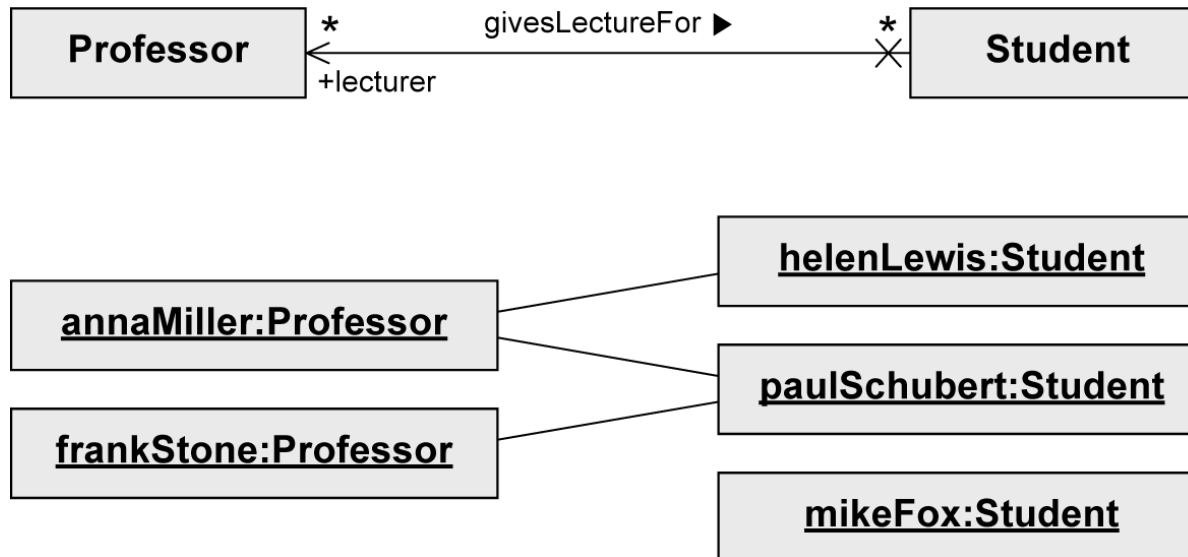


# Specification of Classes: Different Levels of Detail

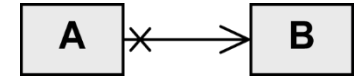


# Association

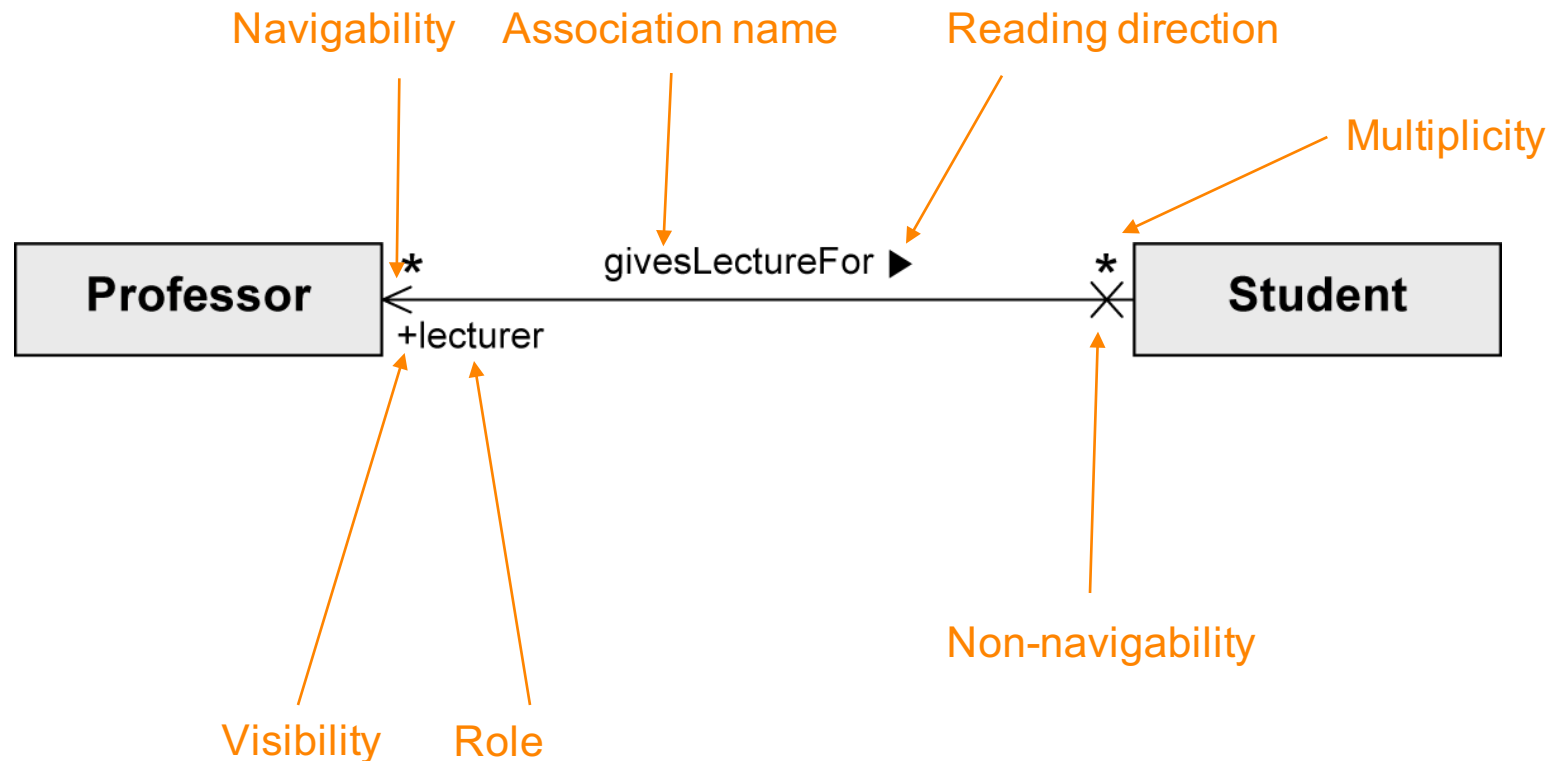
- Models possible relationships between instances of classes



# Binary Association



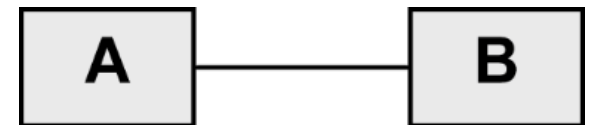
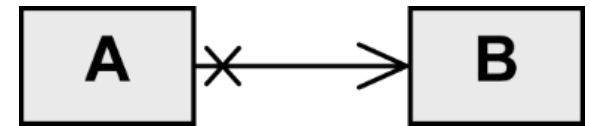
- Connects instances of two classes with one another





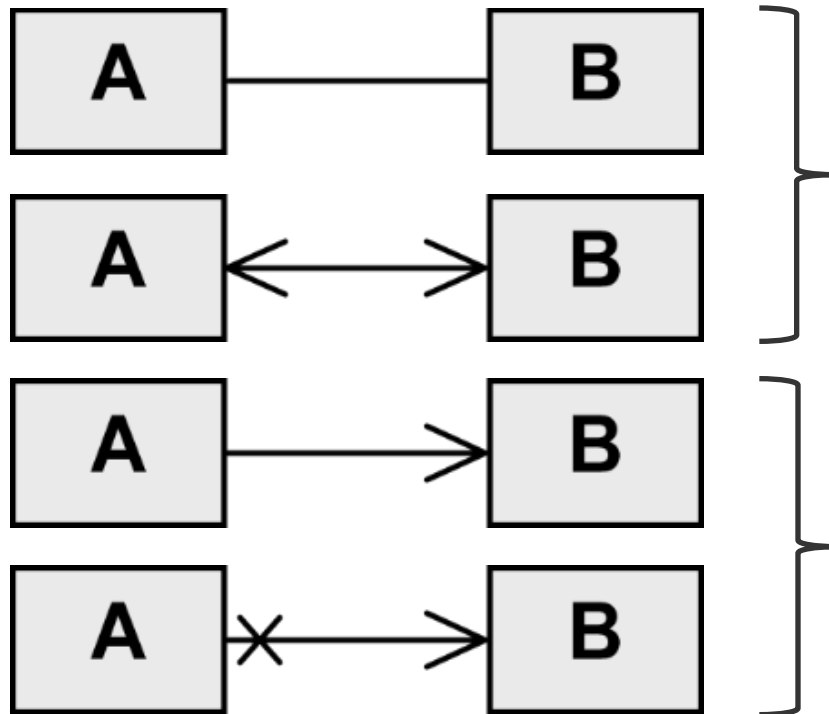
# Binary Association - Navigability

- **Navigability:** an object knows its partner objects and can therefore access their visible attributes and operations
  - Indicated by open arrow head
- **Non-navigability**
  - Indicated by cross
- **Example:**
  - **A** can access the visible attributes and operations of **B**
  - **B** cannot access any attributes and operations of **A**
- **Navigability undefined**
  - Bidirectional navigability is assumed

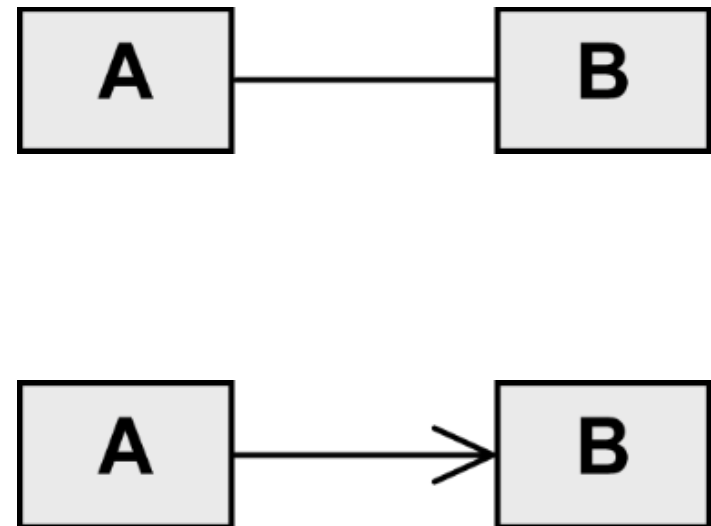


# Navigability – UML Standard vs. Best Practice

UML standard

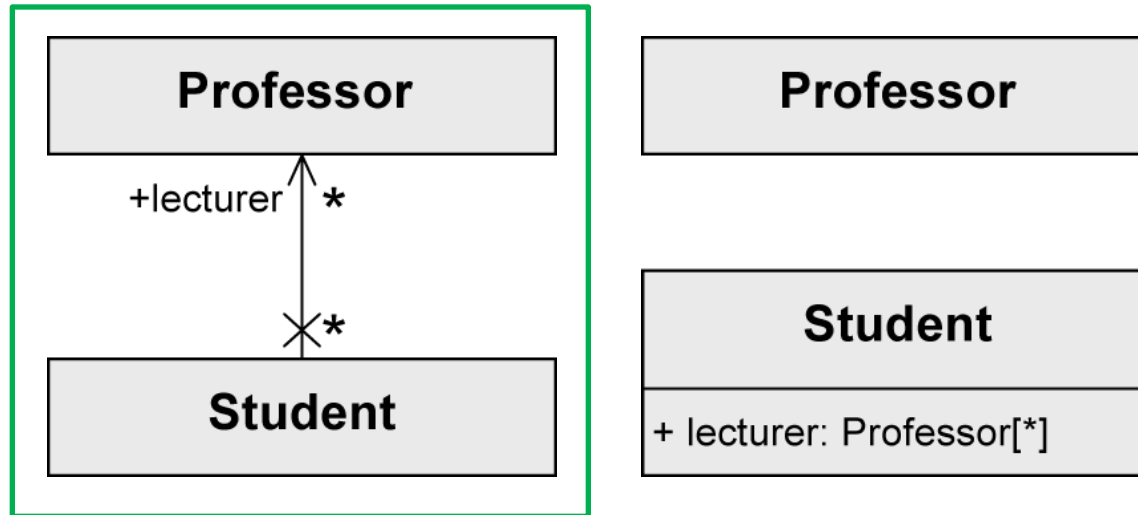


Best practice



# Binary Association as Attribute

Preferable



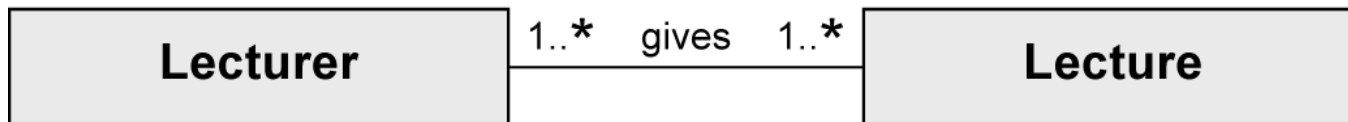
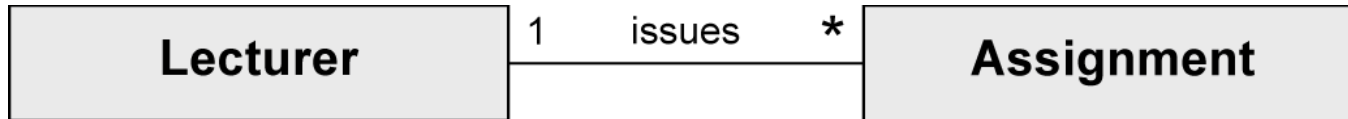
- Java-like notation:

```
class Professor {...}

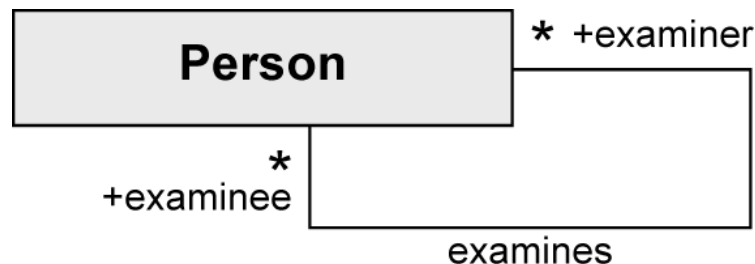
class Student{
    public Professor[] lecturer;
    ...
}
```

# Binary Association – Multiplicity and Role

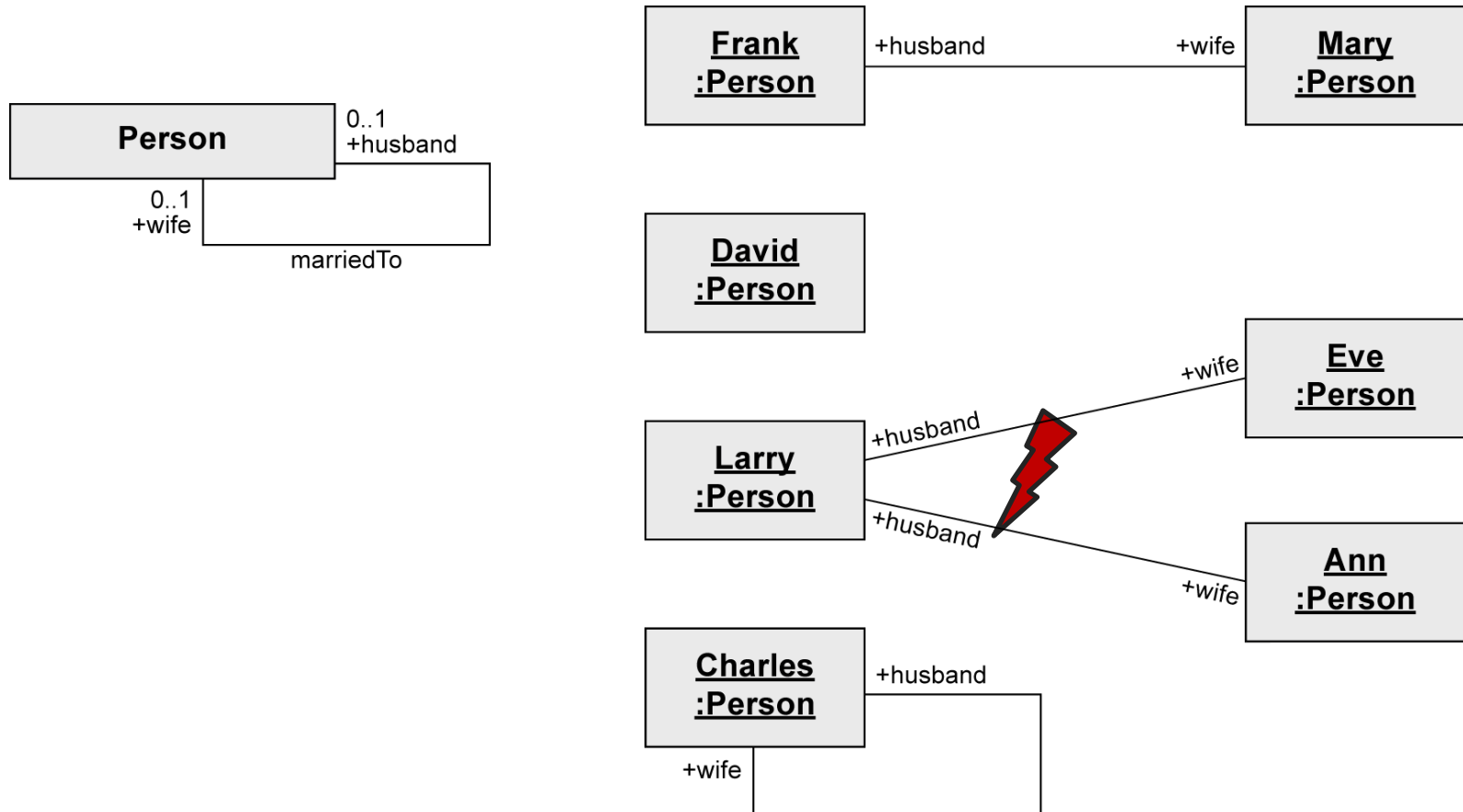
- **Multiplicity:** Number of objects that may be associated with exactly one object of the opposite side



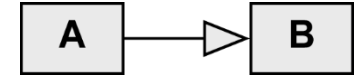
- **Role:** describes the way in which an object is involved in an association relationship



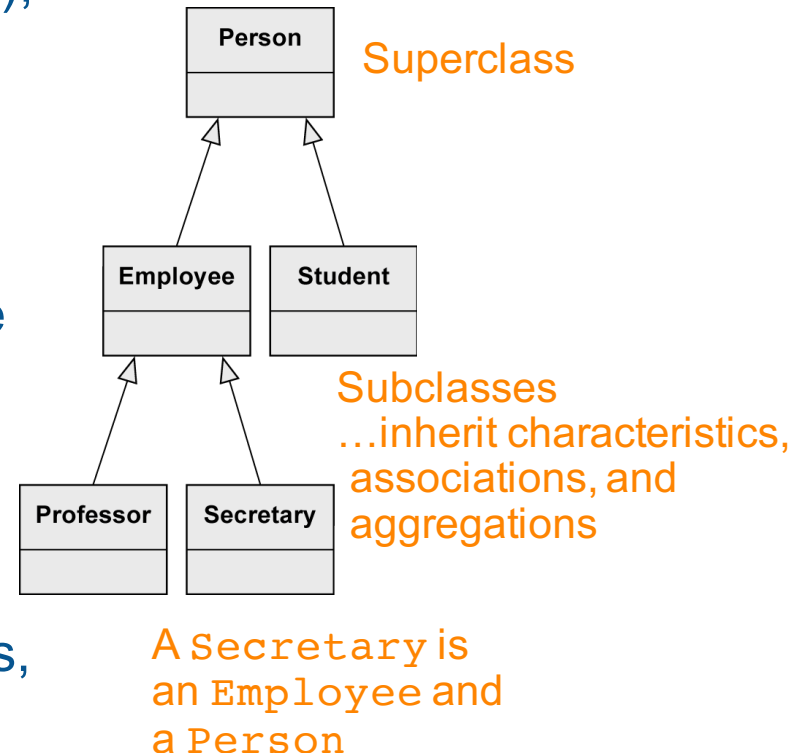
# Unary Association - Example



# Generalization



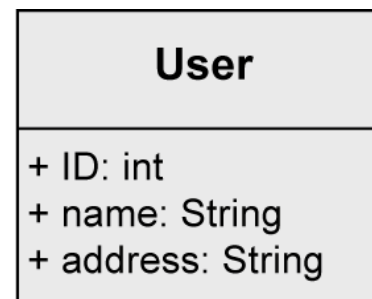
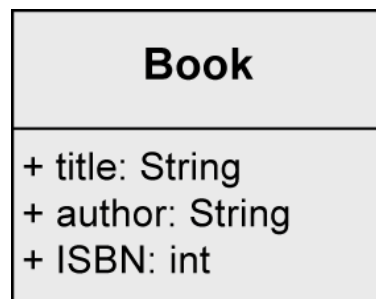
- Characteristics (attributes and operations), associations, and aggregations that are specified for a general class (superclass) are passed on to its subclasses.
- Every instance of a subclass is simultaneously an indirect instance of the superclass.
- Subclass inherits all characteristics, associations, and aggregations of the superclass except private ones.
- Subclass may have further characteristics, associations, and aggregations.
- Generalizations are transitive.



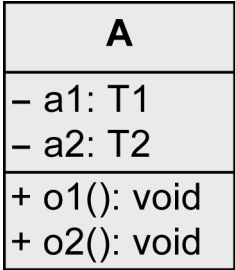
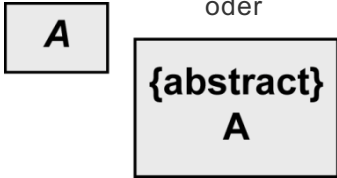
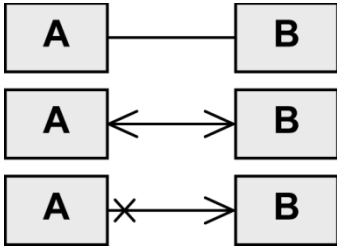
# Creating a Class Diagram

---

- Not possible to completely extract classes, attributes and associations from a natural language text automatically.
- Guidelines
  - Nouns often indicate classes
  - Adjectives indicate attribute values
  - Verbs indicate operations
- **Example:** The library management system stores users with their unique ID, name and address as well as books with their title, author and ISBN number. Ann Foster wants to use the library.



## Notation Elements (1/3)

Name	Notation	Description
Class		Description of the structure and behavior of a set of objects
Abstract class		Class that cannot be instantiated
Association		Relationship between classes: navigability unspecified, navigable in both directions, not navigable in one direction





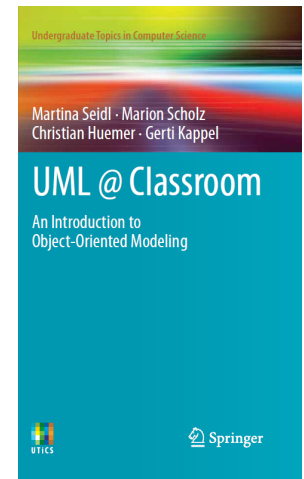
Business Informatics Group

Vienna University of Technology

# Object-Oriented Modeling

## Sequence Diagram

Slides accompanying UML@Classroom  
Version 1.0



### Business Informatics Group

Institute of Software Technology and Interactive Systems  
Vienna University of Technology

Favoritenstraße 9-11/188-3, 1040 Vienna, Austria  
phone: +43 (1) 58801-18804 (secretary), fax: +43 (1) 58801-18896  
office@big.tuwien.ac.at, www.big.tuwien.ac.at

# Content

---

- Introduction
- Interactions and interaction partners
- Messages
- Combined fragments
  - Branches and loops
  - Concurrency and order
  - Filters and assertions
- Further language elements
- Further types of interaction diagrams

# Introduction

---

- Modeling inter-object behavior
  - = interactions between objects
- Interaction
  - Specifies how messages and data are exchanged between interaction partners
- Interaction partners
  - Human (lecturer, administrator, ...)
  - Non-human (server, printer, executable software, ...)
- Examples of interactions
  - Conversation between persons
  - Message exchange between humans and a software system
  - Communication protocols
  - Sequence of method calls in a program
  - ...

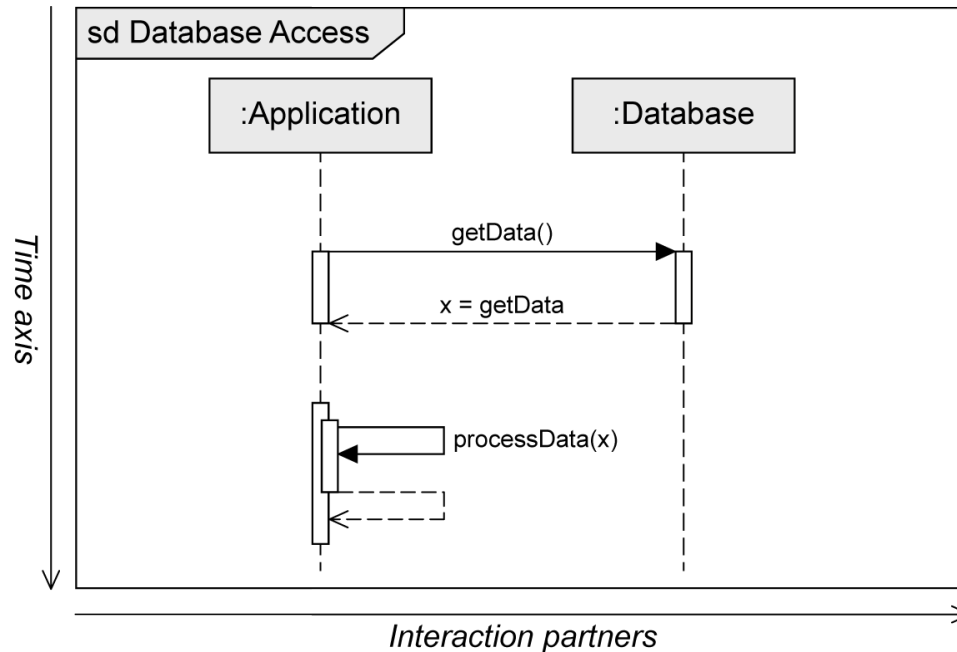
# Interaction Diagrams

---

- Used to specify interactions
- Modeling concrete scenarios
- Describing communication sequences at different levels of detail
- Interaction Diagrams show the following:
  - Interaction of a system with its environment
  - Interaction between system parts in order to show how a specific use case can be implemented
  - Interprocess communication in which the partners involved must observe certain protocols
  - Communication at class level (operation calls, inter-object behavior)

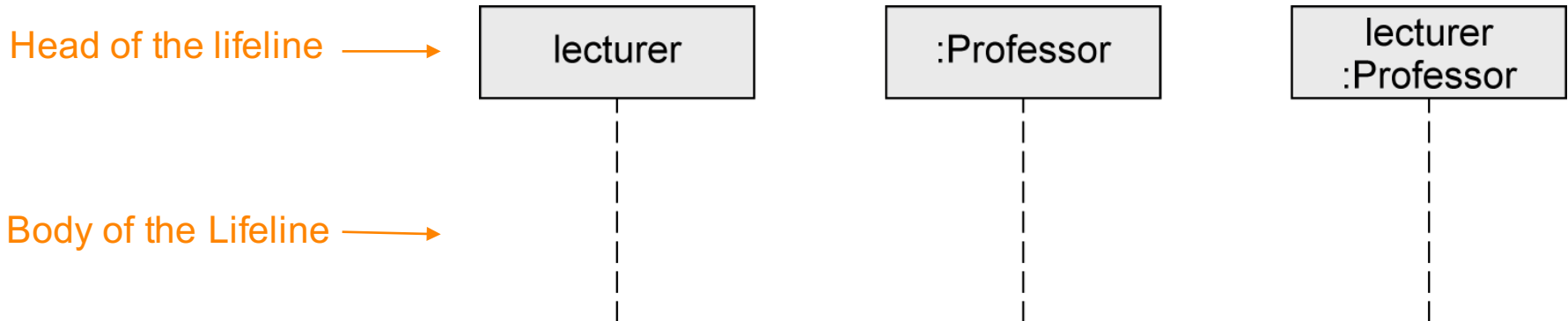
# Sequence Diagram

- Two-dimensional diagram
  - Horizontal axis: involved interaction partners
  - Vertical axis: chronological order of the interaction
- Interaction = sequence of event specifications



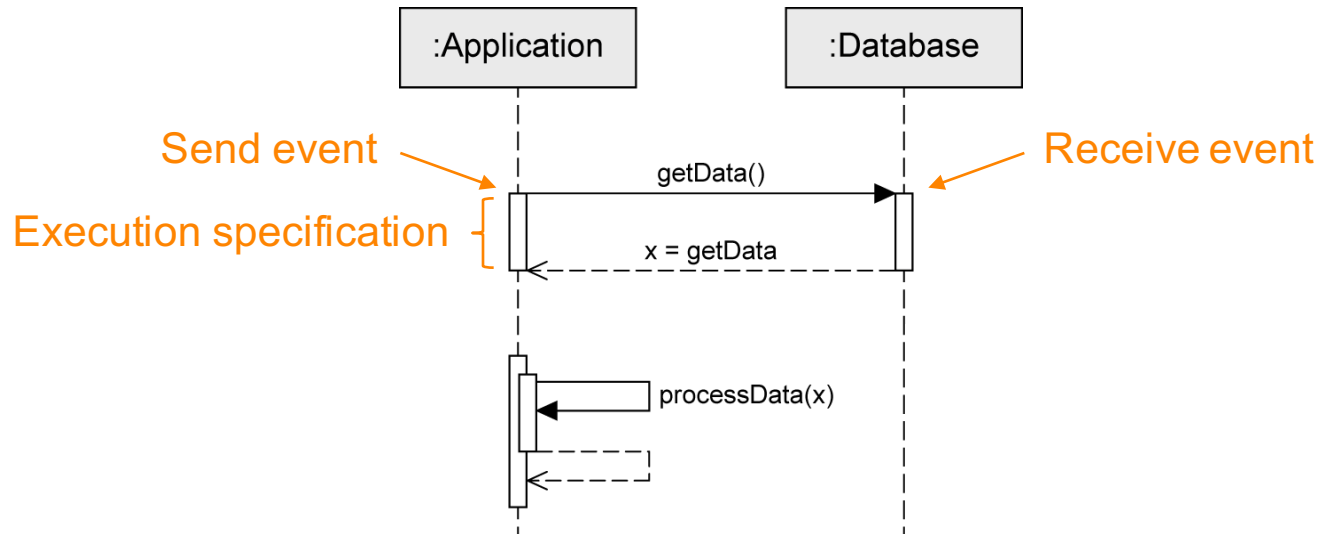
# Interaction Partners

- Interaction partners are depicted as lifelines
- Head of the lifeline
  - Rectangle that contains the expression **roleName:Class**
  - Roles are a more general concept than objects
  - Object can take on different roles over its lifetime
- Body of the lifeline
  - Vertical, usually dashed line
  - Represents the lifetime of the object associated with it



# Exchanging Messages (1/2)

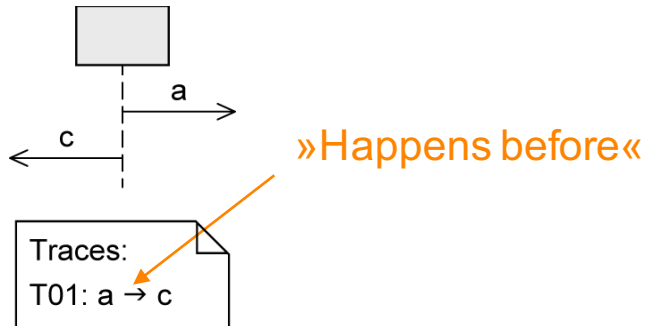
- Interaction: sequence of events
- Message is defined via send event and receive event
- Execution specification
  - Continuous bar
  - Used to visualize when an interaction partner executes some behavior



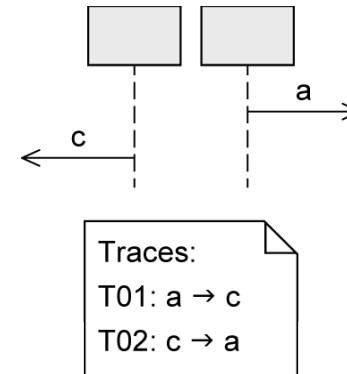
# Exchanging Messages (2/2)

## Order of messages:

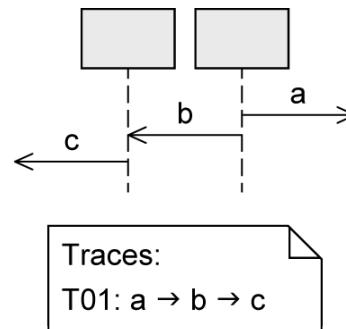
... on one lifeline



... on different lifelines



... on different lifelines which exchange messages

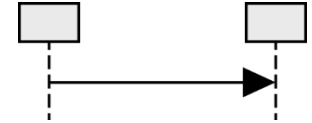




# Messages (1/3)

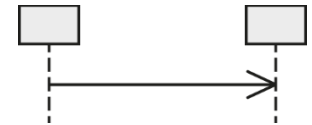
## ■ Synchronous message

- Sender waits until it has received a response message before continuing
- Syntax of message name: **msg(par1, par2)**
  - **msg**: the name of the message
  - **par**: parameters separated by commas



## ■ Asynchronous message

- Sender continues without waiting for a response message
- Syntax of message name: **msg(par1, par2)**



## ■ Response message

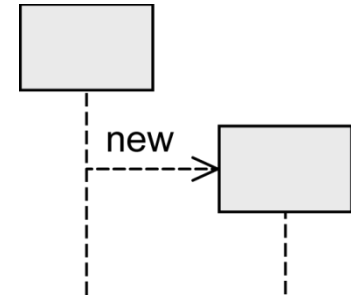
- May be omitted if content and location are obvious
- Syntax: **att=msg(par1, par2):val**
  - **att**: the return value can optionally be assigned to a variable
  - **msg**: the name of the message
  - **par**: parameters separated by commas
  - **val**: return value



## Messages (2/3)

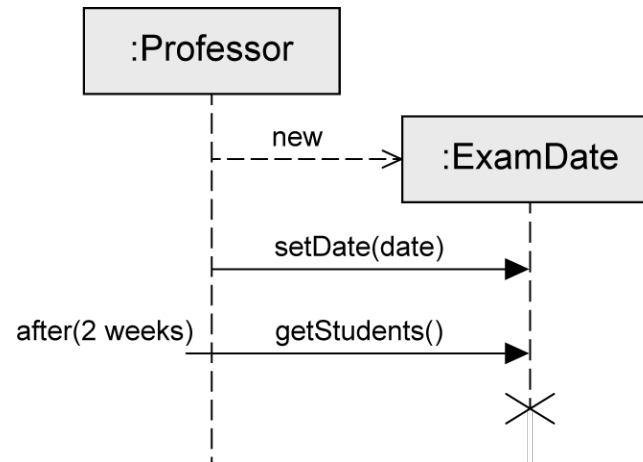
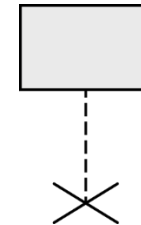
### ■ Object creation

- Dashed arrow
- Arrowhead points to the head of the lifeline of the object to be created
- Keyword **new**



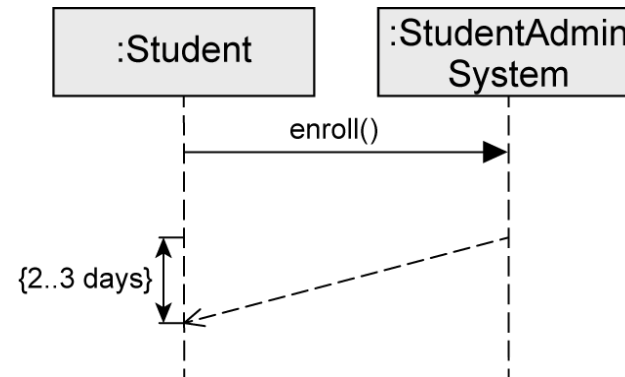
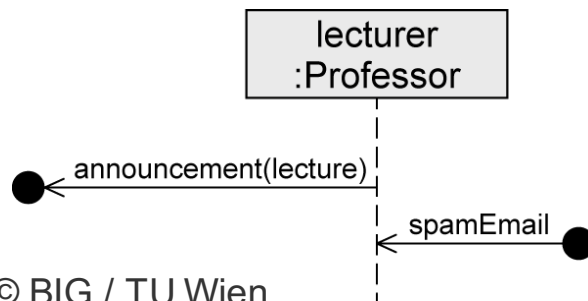
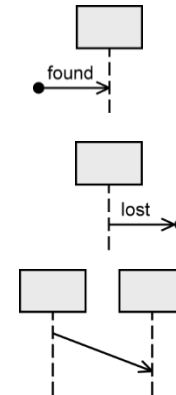
### ■ Object destruction

- Object is deleted
- Large cross ( × ) at the end of the lifeline



# Messages (3/3)

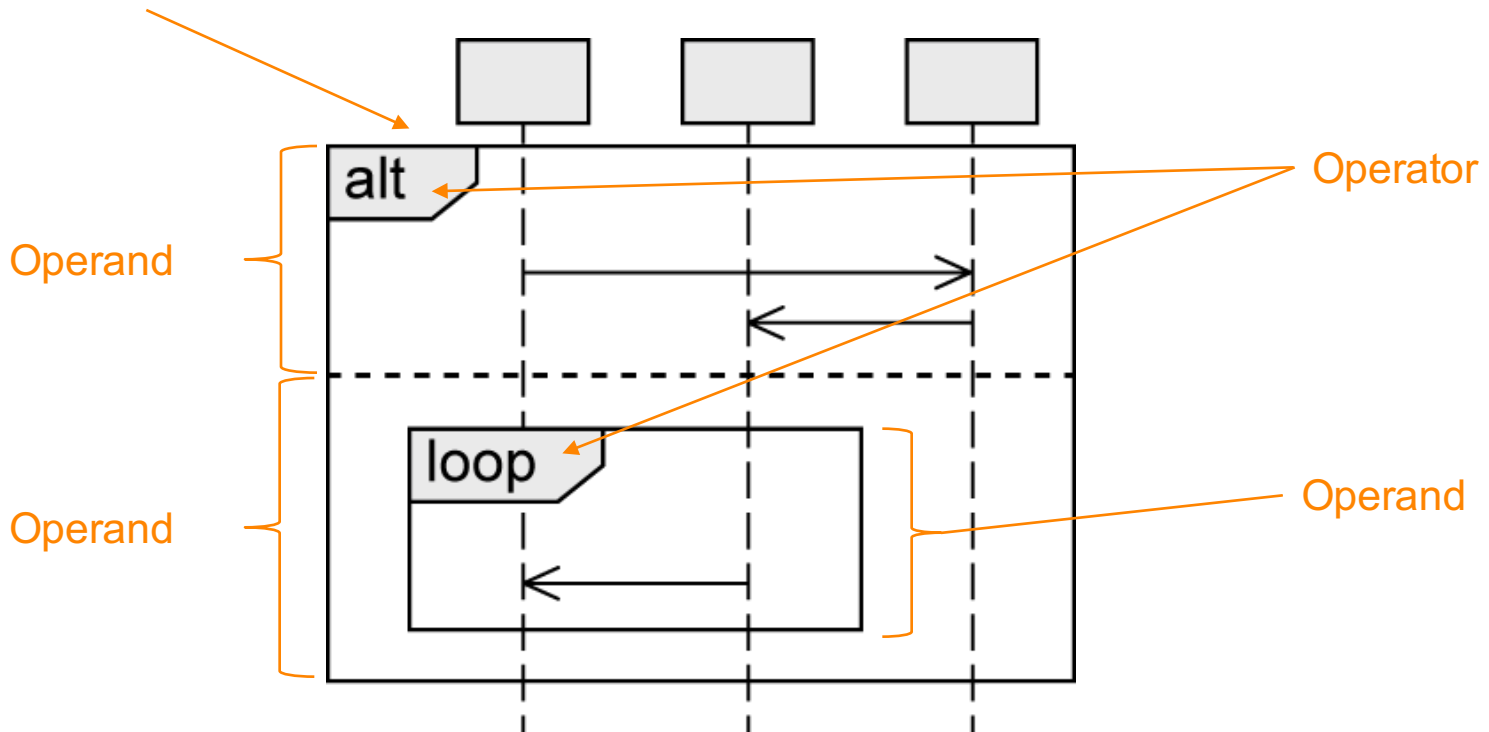
- Found message
  - Sender of a message is unknown or not relevant
- Lost message
  - Receiver of a message is unknown or not relevant
- Time-consuming message
  - "Message with duration"
  - Usually messages are assumed to be transmitted without any loss of time
  - Express that time elapses between the sending and the receipt of a message



# Combined Fragments

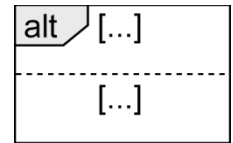
- Model various control structures
- 12 predefined types of operators

Combined Fragment



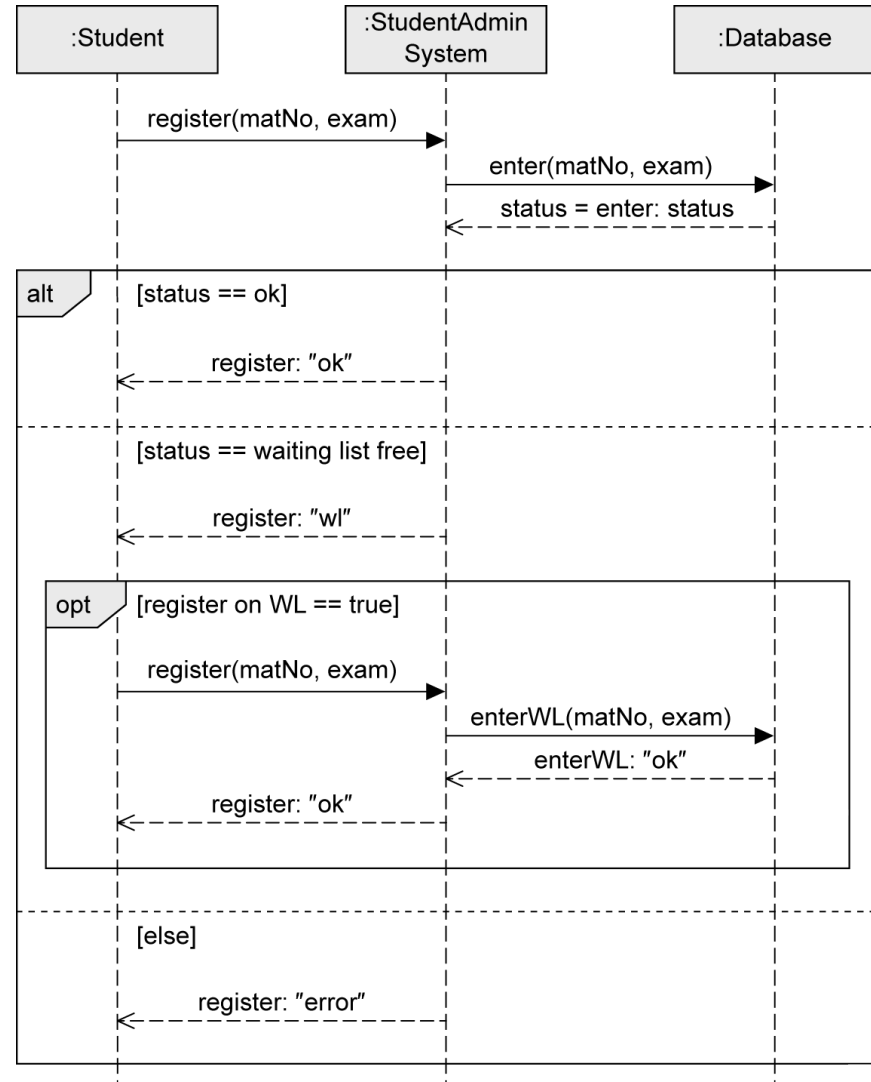
# Types of Combined Fragments

	Operator	Purpose
Branches and loops	<b>alt</b>	Alternative interaction
	<b>opt</b>	Optional interaction
	<b>loop</b>	Repeated interaction
	<b>break</b>	Exception interaction
Concurrency and order	<b>seq</b>	Weak order
	<b>strict</b>	Strict order
	<b>par</b>	Concurrent interaction
	<b>critical</b>	Atomic interaction
Filters and assertions	<b>ignore</b>	Irrelevant interaction
	<b>consider</b>	Relevant interaction
	<b>assert</b>	Asserted interaction
	<b>neg</b>	Invalid interaction



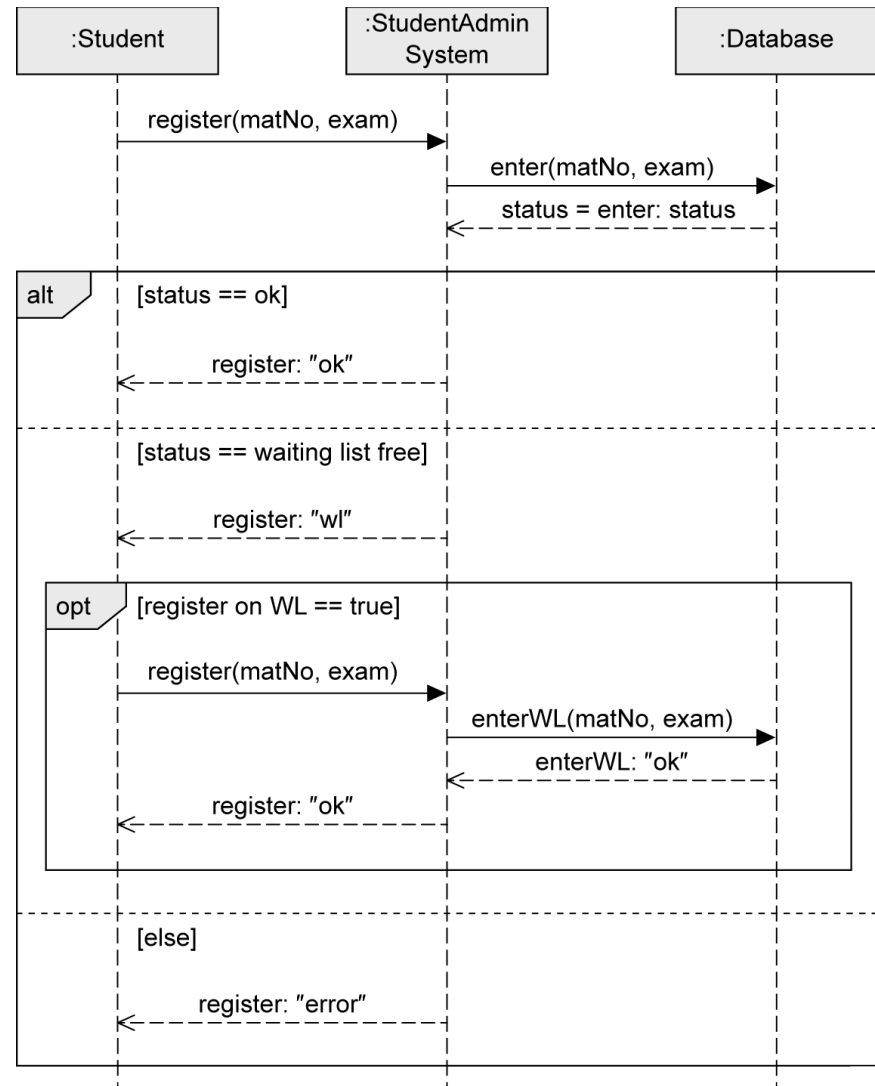
# alt Fragment

- To model alternative sequences
- Similar to switch statement in Java
- Guards are used to select the one path to be executed
- Guards
  - Modeled in square brackets
  - default: `true`
  - predefined: `[else]`
- Multiple operands
- Guards have to be disjoint to avoid indeterministic behavior



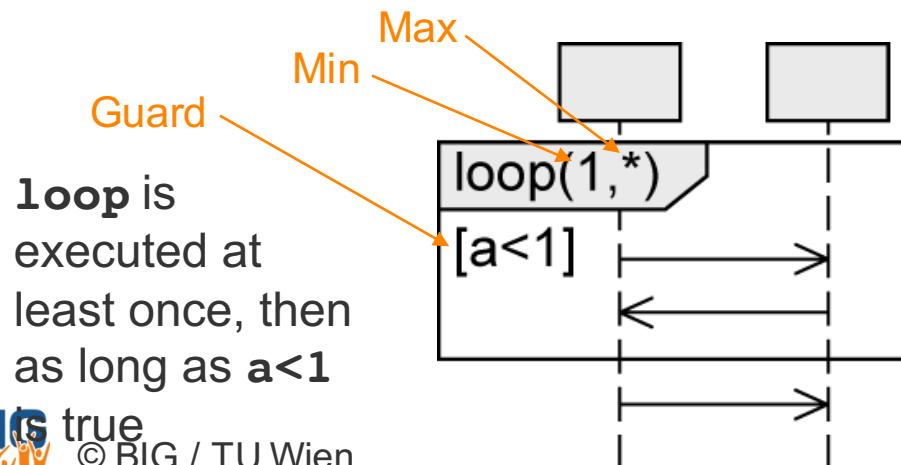
# opt Fragment

- To model an optional sequence
- Actual execution at runtime is dependent on the guard
- Exactly one operand
- Similar to **if** statement without **else** branch
- equivalent to **alt** fragment with two operands, one of which is empty



# loop Fragment

- To express that a sequence is to be executed repeatedly
- Exactly one operand
- Keyword **loop** followed by the minimal/maximal number of iterations (**min..max**) or (**min,max**)
  - default: ( \* ) .. no upper limit
- Guard
  - Evaluated as soon as the minimum number of iterations has taken place
  - Checked for each iteration within the (**min,max**) limits
  - If the guard evaluates to false, the execution of the loop is terminated

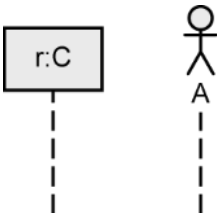
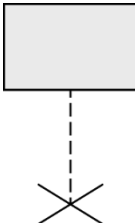
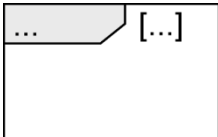


## Notation alternatives:




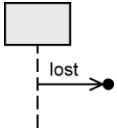
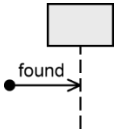
$\text{loop}(3, 8) = \text{loop}(3..8)$   
 $\text{loop}(8, 8) = \text{loop}(8)$   
 $\text{loop} = \text{loop} (*) = \text{loop}(0, *)$



## Notation Elements (1/2)

Name	Notation	Description
Lifeline		Interaction partners involved in the communication
Destruction event		Time at which an interaction partner ceases to exist
Combined fragment		Control constructs

## Notation Elements (2/2)

Name	Notation	Description
Synchronous message		Sender waits for a response message
Response message		Response to a synchronous message
Asynchronous communication		Sender continues its own work after sending the asynchronous message
Lost message		Message to an unknown receiver
Found message		Message from an unknown sender