



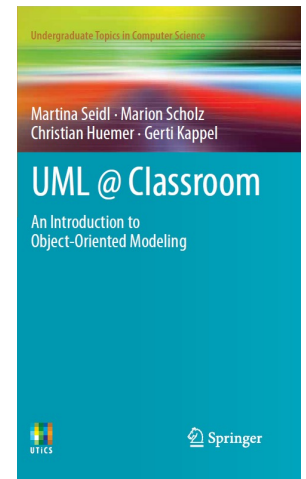
Business Informatics Group

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

Introduction



<https://youtu.be/zid-MVo7M-E>

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

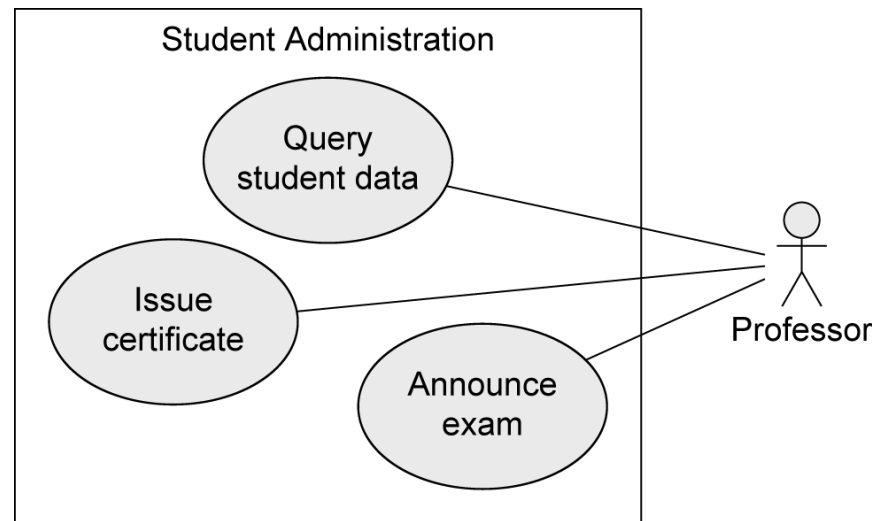
(who interacts with the system?)

- Professor

■ Use cases

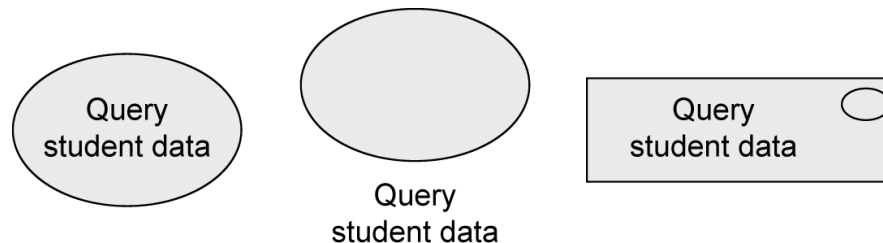
(what can the actors do?)

- Query student data
- Issue certificate
- Announce exam



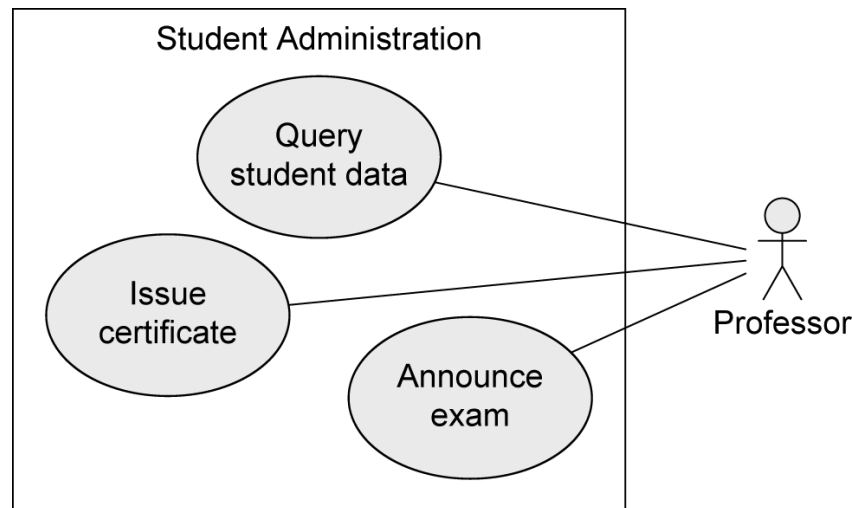
Use Case

- *Describes functionality expected from the system under development.*
- Provides **tangible benefit** for one or more **actors** that communicate with this use case.
- The use case diagram does **not** cover the *internal structure and the actual implementation of a 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:



Example: Student Administration System

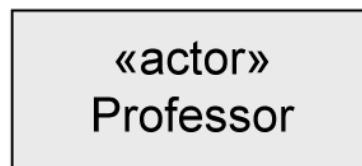
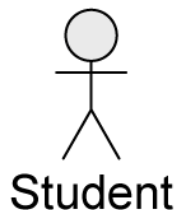
- In general, a use case is *triggered* either by **invocation of an actor** or by a **trigger event**, in short, a **trigger**.
 - An example of a trigger is that the **semester has ended** and hence the use case Issue certificate must be executed.
- The use cases are generally grouped within a **rectangle**. This rectangle symbolizes the **boundaries of the system** to be System described.



Actor

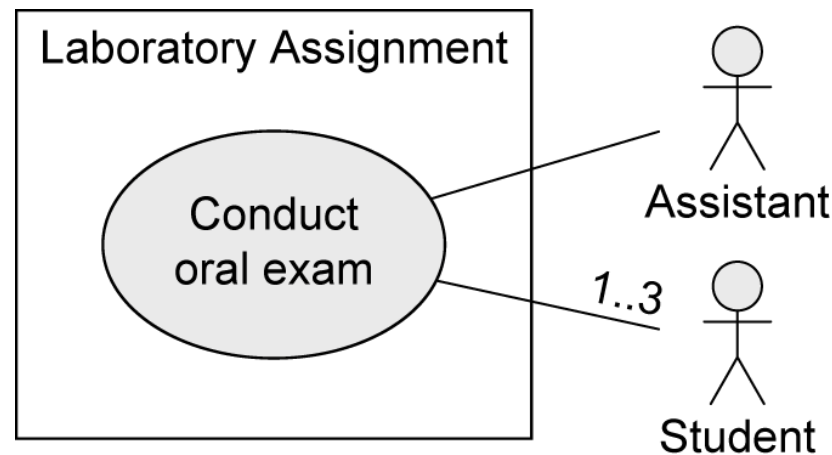


- To describe a system completely, it is **essential** to document not only what the system can do but also *who works and interacts with the system*.
- Actors interact with the system as:
 - **Active Actor** *by using use cases*, i.e., the actors initiate the execution of use cases.
 - **Passive Actor** *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:



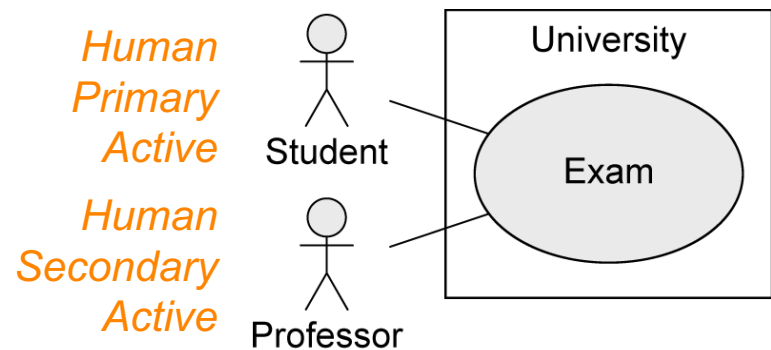
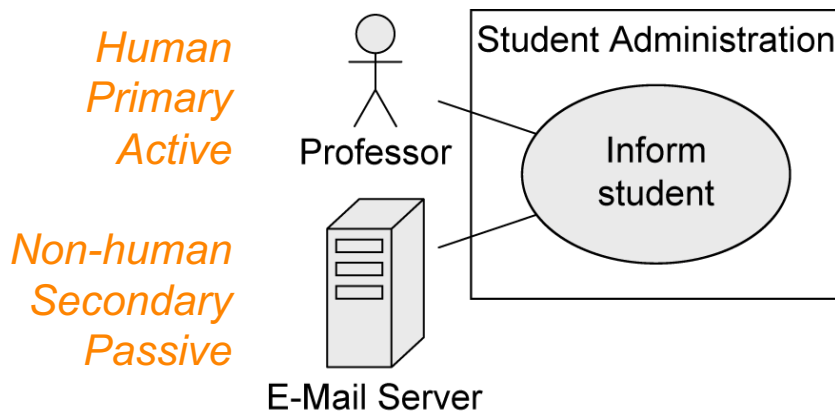
Actor

- Usually, user data is also administered within the system. This data is modeled within the system in the form of objects and classes.
- Example: actor **Assistant**
 - The actor **Assistant** interacts with the system **Laboratory Assignment** by using it.
 - The class **Assistant** describes objects representing user data (e.g., name, ssNr, ...).



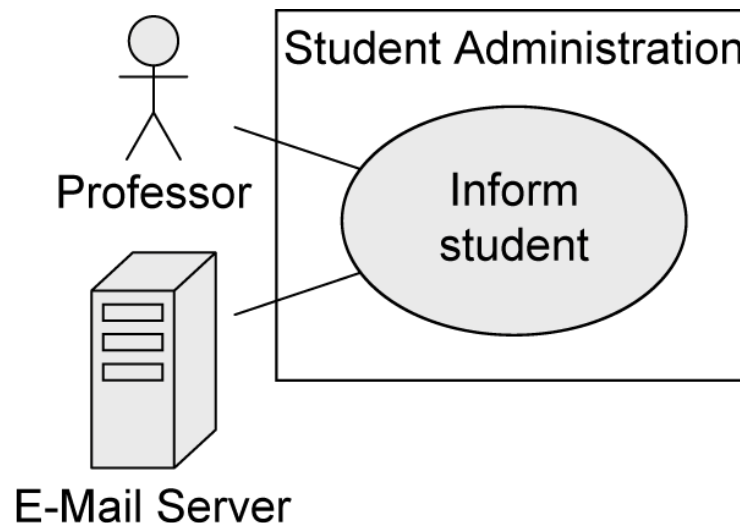
Actor

- **Human, for example: Student, Professor**
- **Non-human, for example: 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
- Graphically, there is **no differentiation** between primary and secondary actors, between active and passive actors, and between human and non-human actors.
- Example:



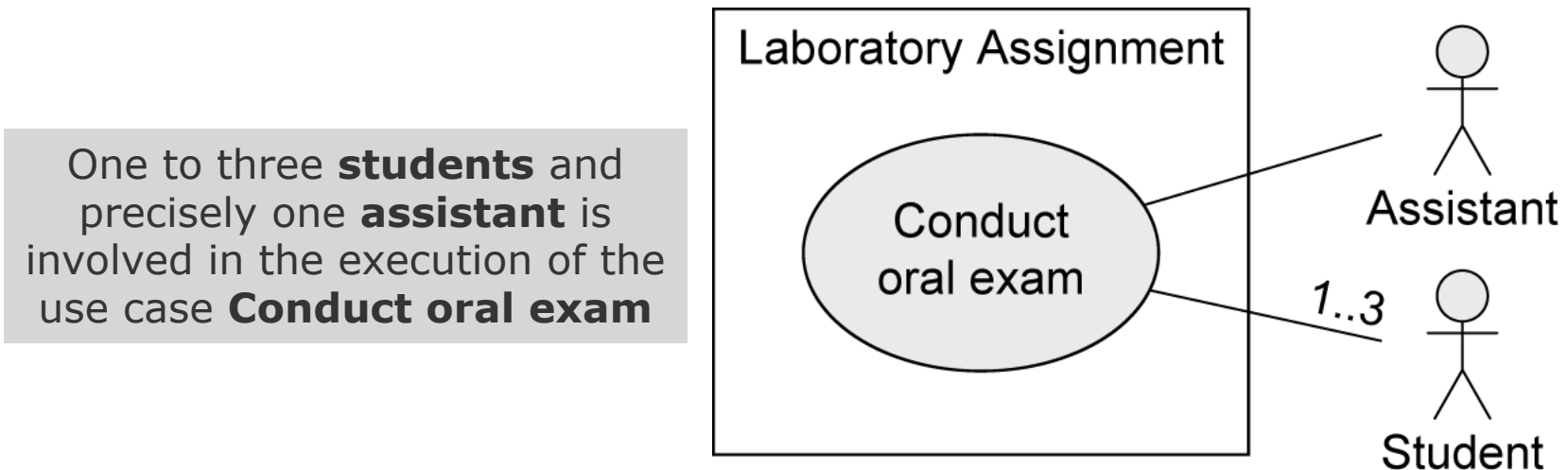
Actor

- The E-Mail Server is an actor—it is not part of the system, but it is necessary for the execution of the use case Inform student.
- However, if no external server is required to execute this use case because the student administration system implements the e-mail functionality itself or has its own server, **the E-Mail Server is no longer an actor.**
 - In that case, only the Professor is required to inform students about various news items.



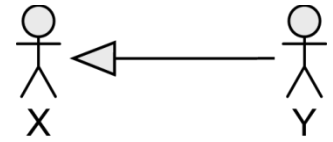
Relationships between Use Cases and Actors

- 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.
 - If a multiplicity greater than 1 is specified for the actor's association end, this means that more than one instance of an actor is involved in the execution of the use case.

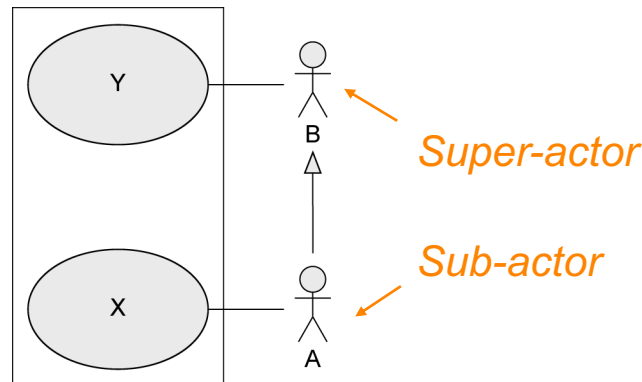


Relationships between Actors

Generalization of Actors

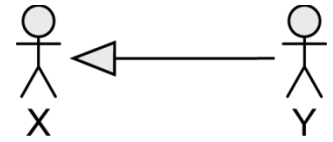


- Actors often have **common properties**, and some use cases can be used by various actors.
 - For example, it is possible that not only **professors** but also **assistants** (i.e., the entire research personnel) are permitted to **view student data**.
- To express this, actors may be depicted in an **inheritance relationship** (**generalization**) with one another.
- When an actor A (sub-actor) inherits from an actor B (super-actor), A is involved with all use cases with which B is involved.
- In simple terms, **generalization** expresses an **“is a” relationship**.
- It is represented with a line from the sub-actor to the super-actor with a large triangular arrowhead at the super-actor end.

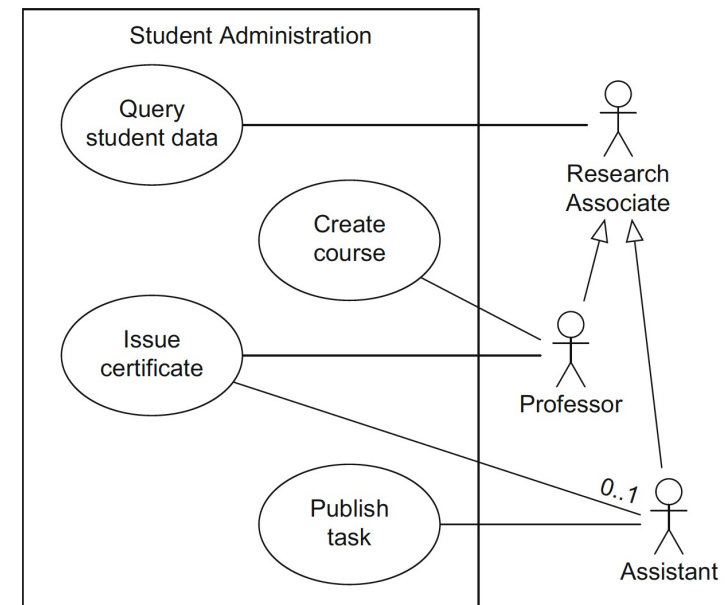


Relationships between Actors

Generalization of Actors

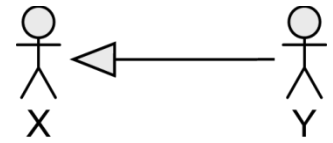


- In the example below, the actors **Professor** and **Assistant** *inherit* from the actor **Research Associate**, which means that every professor and every assistant is a research associate.
- Every research associate can execute the use case **Query student data**. Only professors can create a new course; in contrast, tasks can only be published by assistants.
- To execute the use case **Issue certificate**, an actor **Professor** is required; in addition, an actor **Assistant** can be involved **optionally**, which is expressed by the multiplicity 0..1.

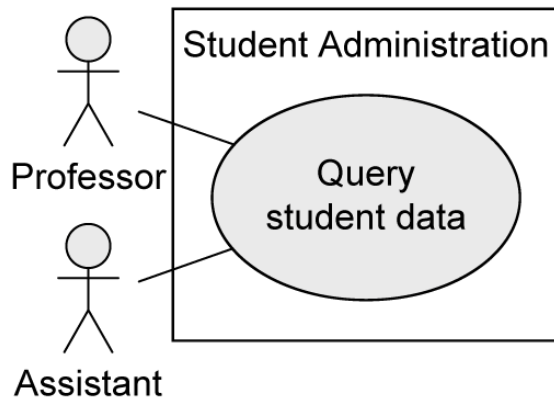


Relationships between Actors

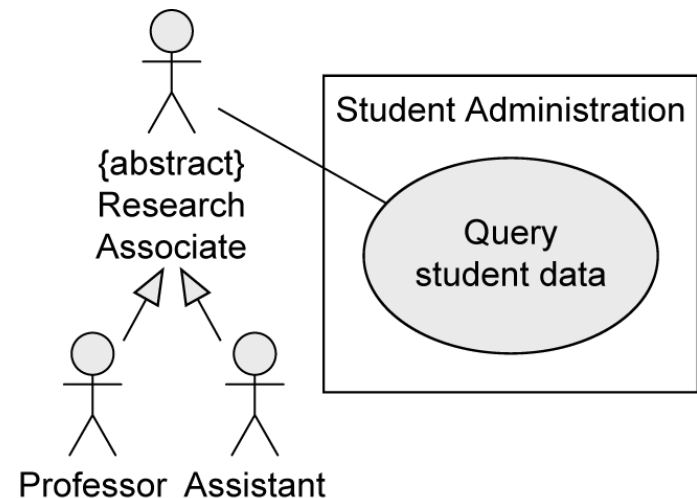
Generalization of Actors



- There is a great difference between two actors participating in a use case themselves and two actors having a common super-actor that participates in the use case.
- In the first case, both actors must participate in the use case.
- In the second case, each of them inherits the association. Then each actor participates in the use case individually
- If there is **no instance of an actor**, this actor can be labeled with the keyword **{abstract}**.



≠

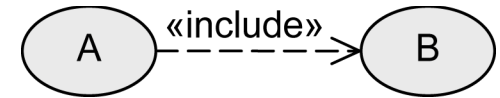


Professor AND Assistant needed
for executing **Query student data**

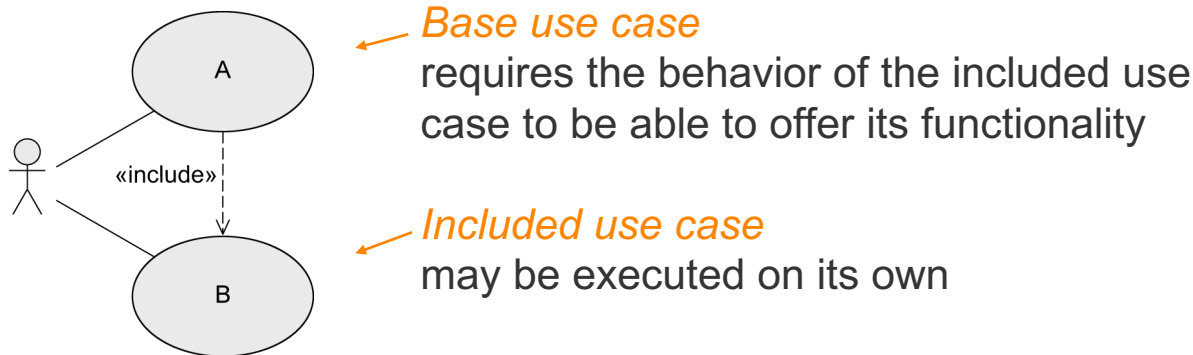
Professor OR Assistant needed
for executing **Query student data**

Relationships between Use Cases

«include» - Relationship

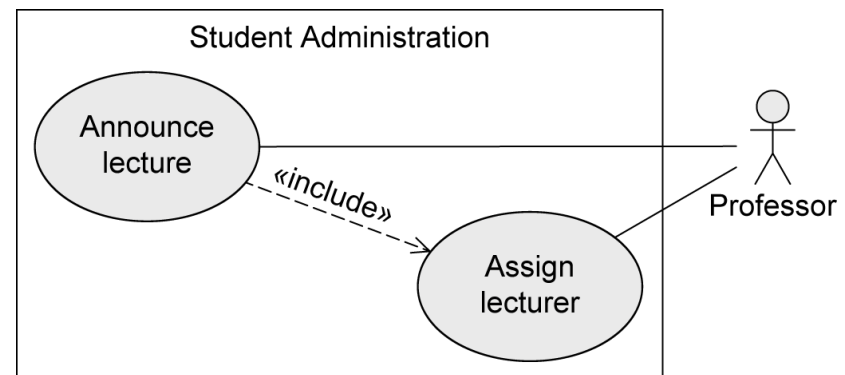


- The behavior of one use case (**included** use case) is **integrated in the** behavior of another use case (**base** use case)



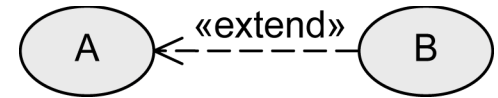
- The use of «include» is analogous to **calling a subroutine** in a procedural programming language.
- One use case may include/included by **multiple other use cases**. In such situations, it is important to ensure that **no cycle arises**.

Whenever a new lecture is announced, the use case Assign lecturer must also be executed, and further lecturers can also be assigned to an existing lecture.

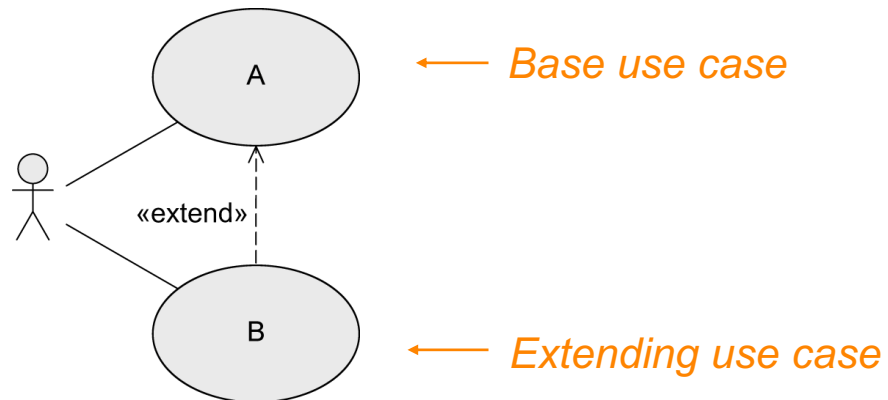


Relationships between Use Cases

«extend» - Relationship

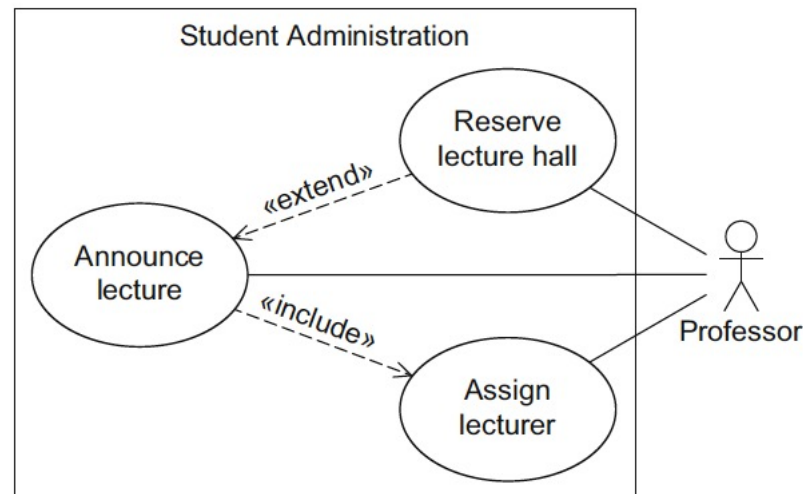


- The behavior of one use case (**extending** use case) may be integrated in the behavior of another use case (**base** use case) but does not have to.
- Both use cases may also be executed independently of each other.
- A decides if B is executed.



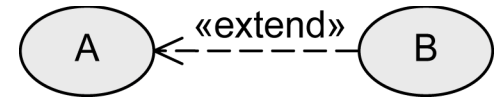
When a new lecture is announced, it is possible (but not mandatory) to reserve a lecture hall.

A use case can act as an extending use case several times or can itself be extended by several use cases. Again, no cycles may arise.

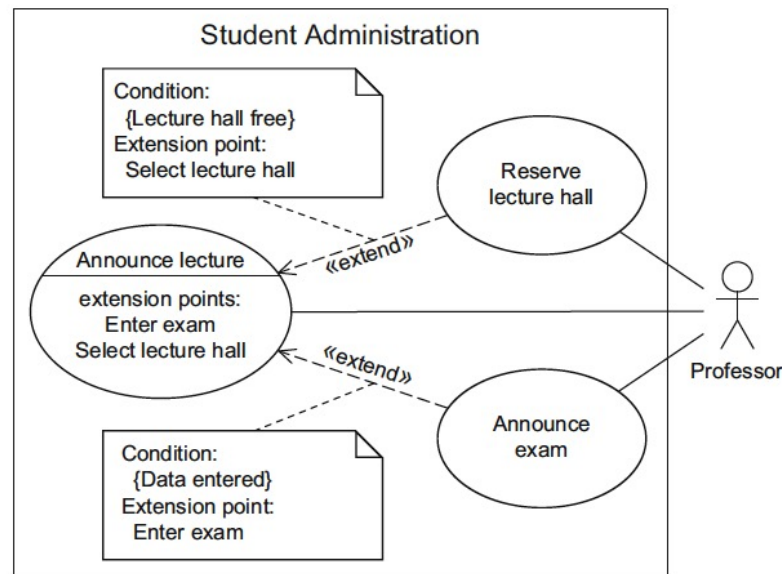


Relationships between Use Cases

«extend» - Relationship

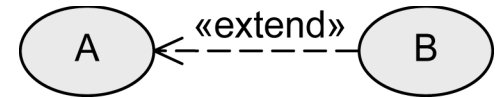


- Extension points define at which point the behavior is integrated.
- Conditions define under which circumstances the behavior is integrated.
- A condition that must be fulfilled for the **base use case** to insert the behavior of the **extending use case**.
- The condition can be specified for every «extend» relationship within curly brackets, in a note that is connected with the corresponding «extend» relationship. A condition is indicated by the preceding keyword Condition followed by a colon.

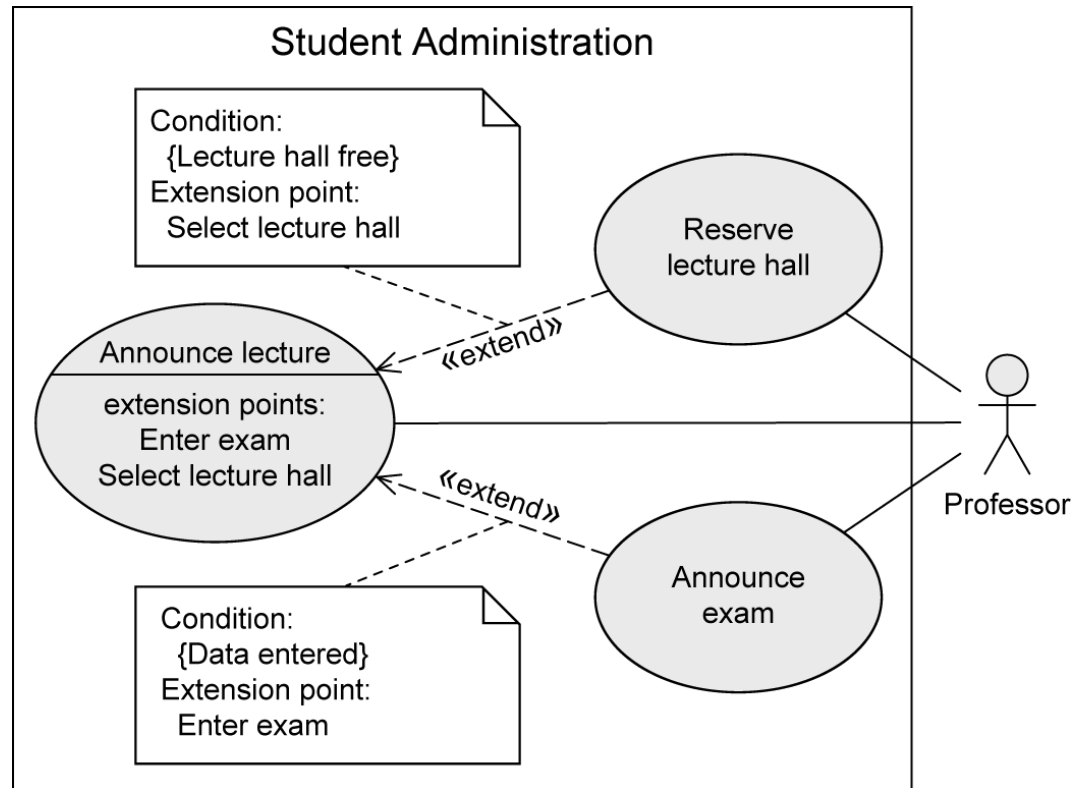


Relationships between Use Cases

«extend» - Relationship: Extension Points



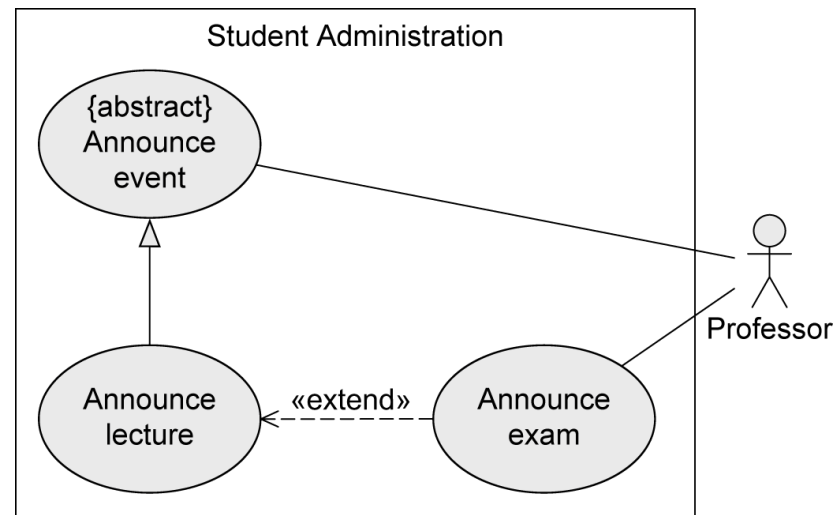
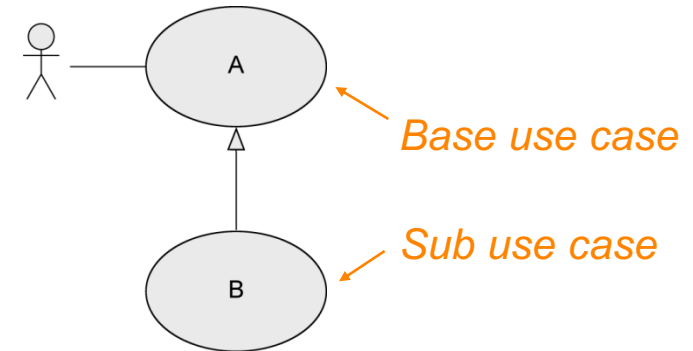
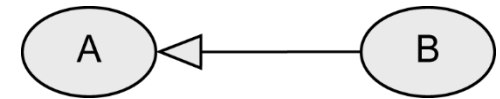
- By using extension points, you can define the point at which the behavior of the extending use cases must be inserted in the base use case.
- The extension points are written directly within the use case
- Specification of multiple extension points is possible.



Relationships between Use Cases

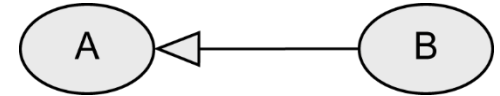
Generalization of Use Cases

- Use case **A** generalizes use case **B**.
- **B** inherits the behavior of **A** and may either extend or overwrite it.
- **B** also inherits all relationships from **A**.
- **B** adopts the basic functionality of **A** but decides itself what part of **A** is executed or changed.
- **A** may be labeled **{abstract}**
 - Cannot be executed directly
 - Only **B** is executable
- Example:

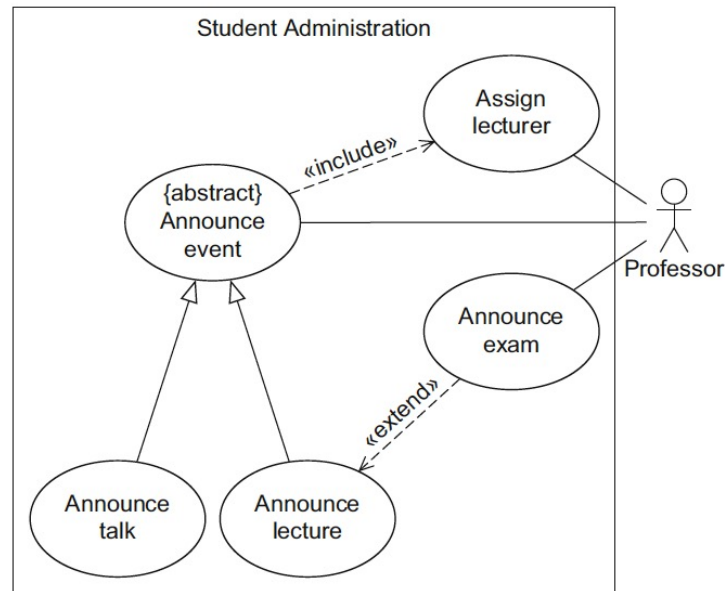


Relationships between Use Cases

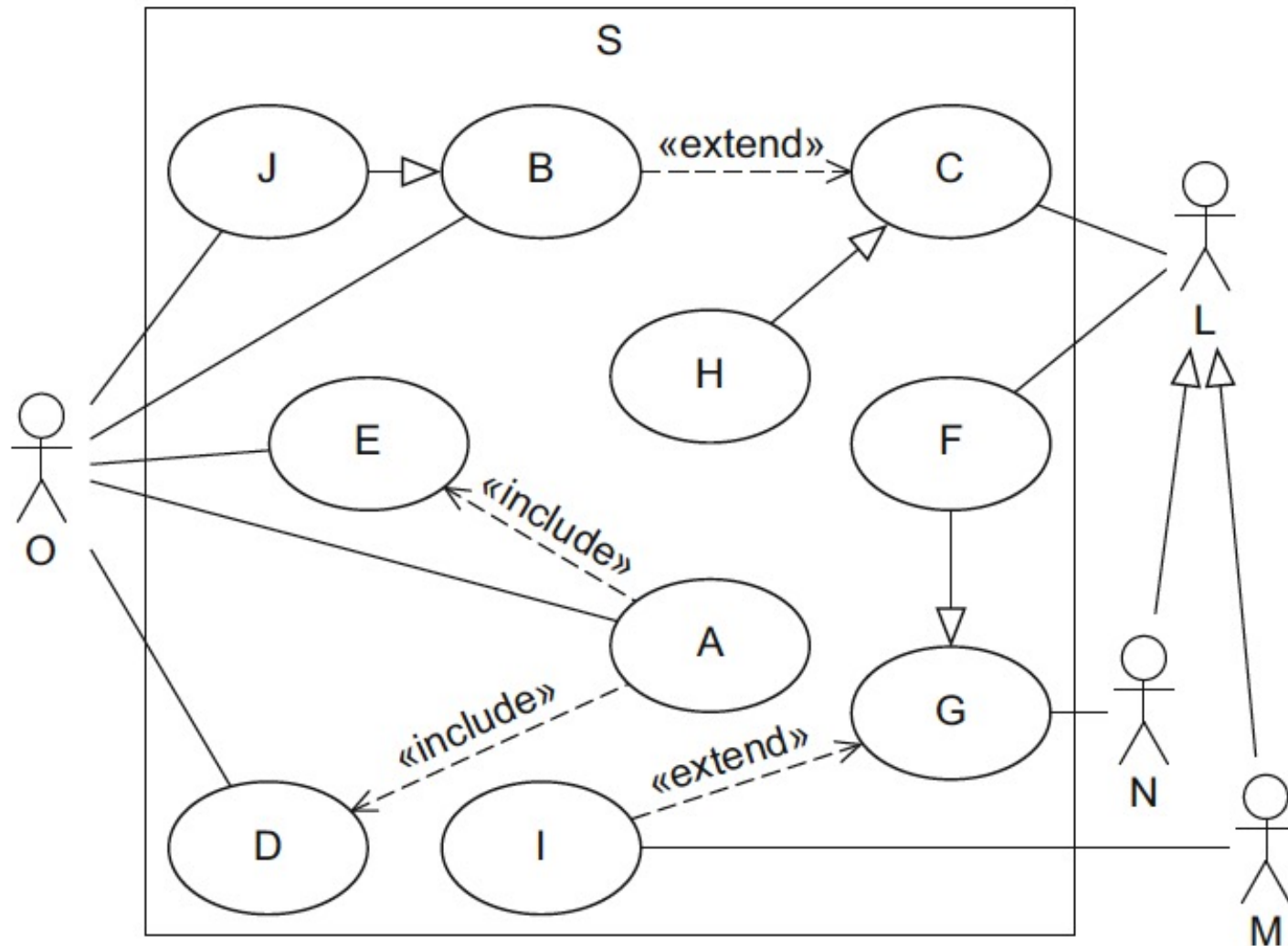
Generalization of Use Cases



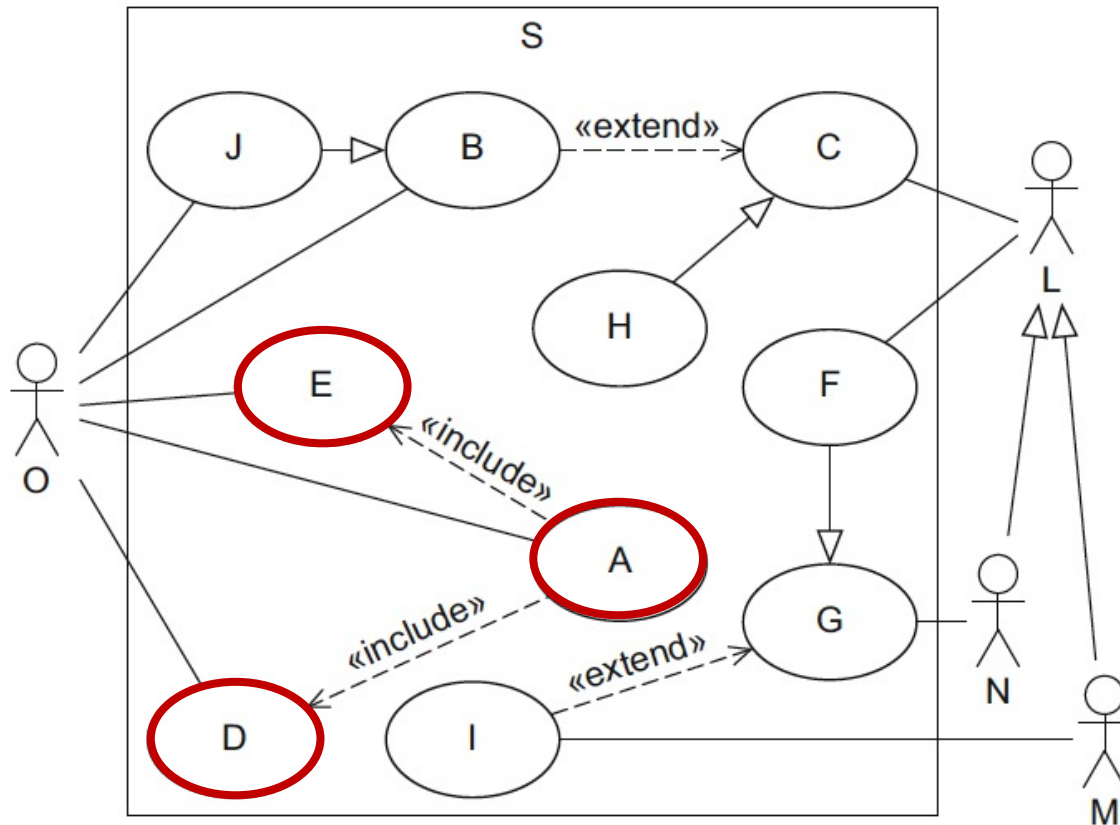
- The abstract use case **Announce event** passes on its properties and behavior to the use cases **Announce lecture** and **Announce talk**. As a result of an «include» relationship, both use cases must execute the behavior of the use case **Assign lecturer**.
- When a lecture is announced, an exam can also be announced at the same time.
- Both use cases inherit the relationship from the use case **Announce event** to the actor **Professor**.



In Class Activity: Examples of Relationships

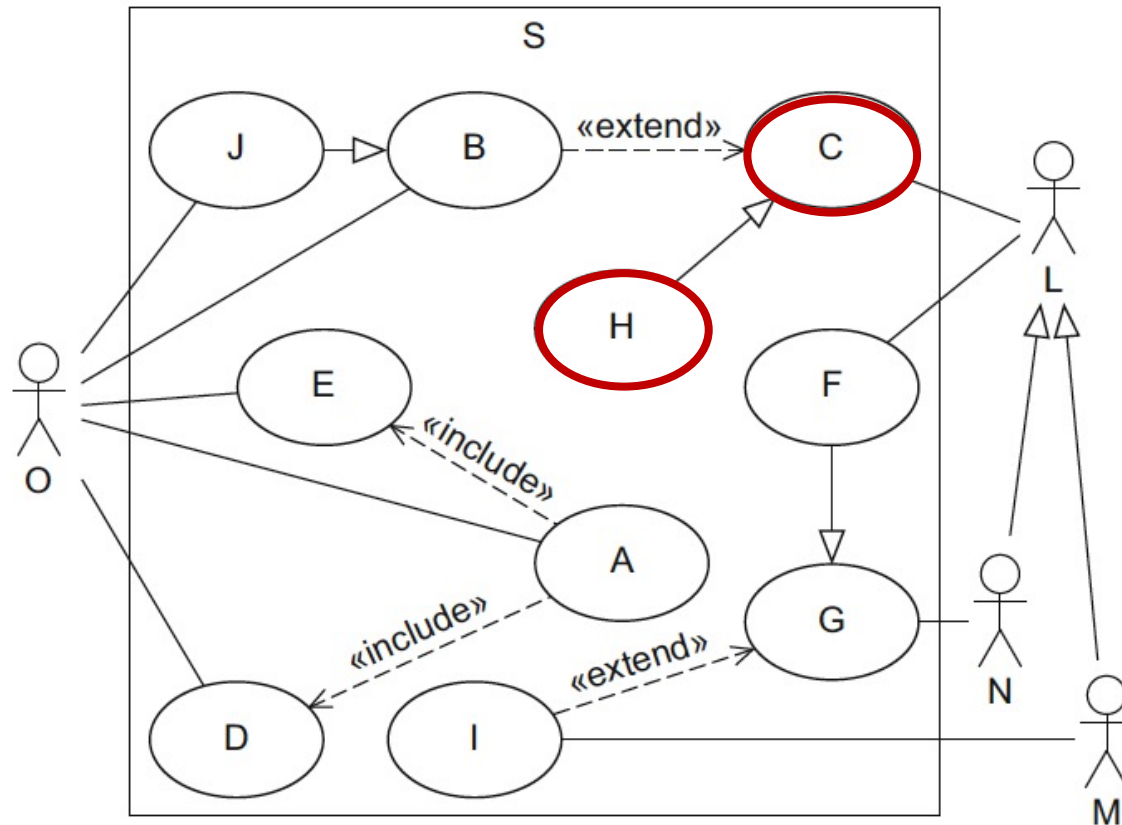


Examples of Relationships



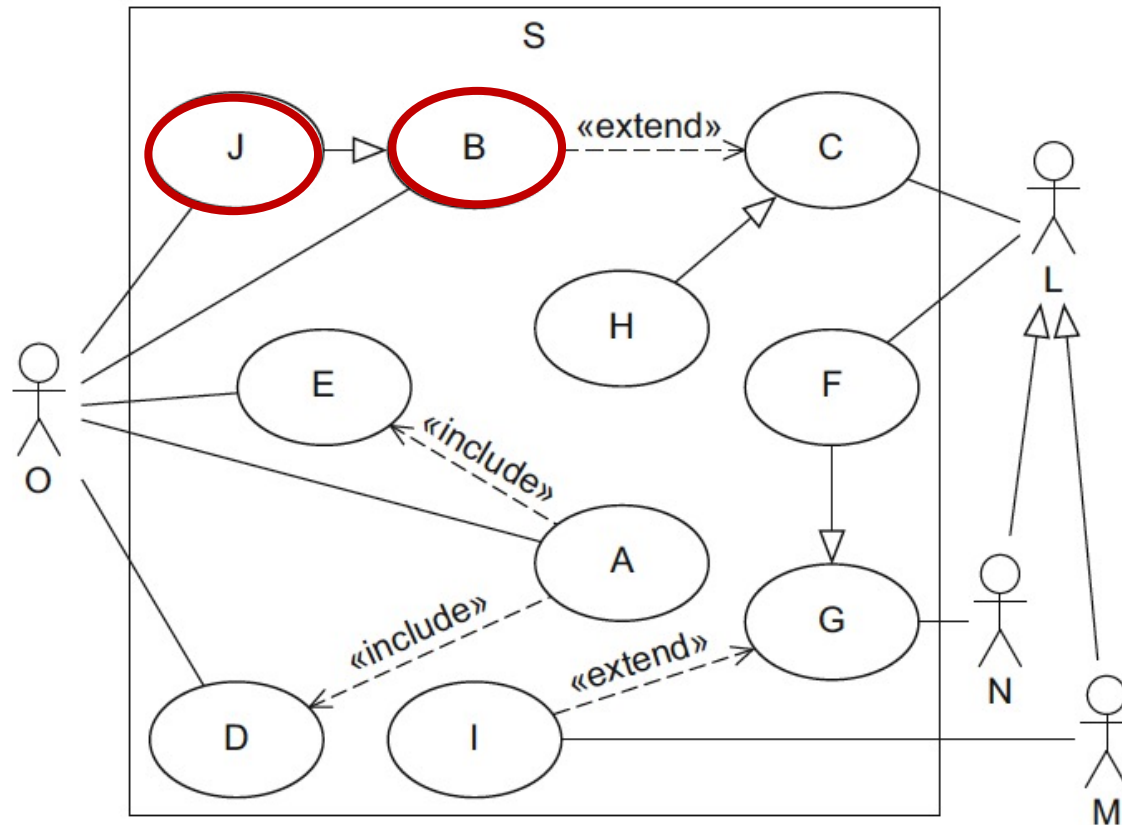
The use case A includes the use cases E and D. An actor O is involved in all three use cases. There is no specification of whether this is the same user or different users, that is, different instances of O.

Examples of Relationships



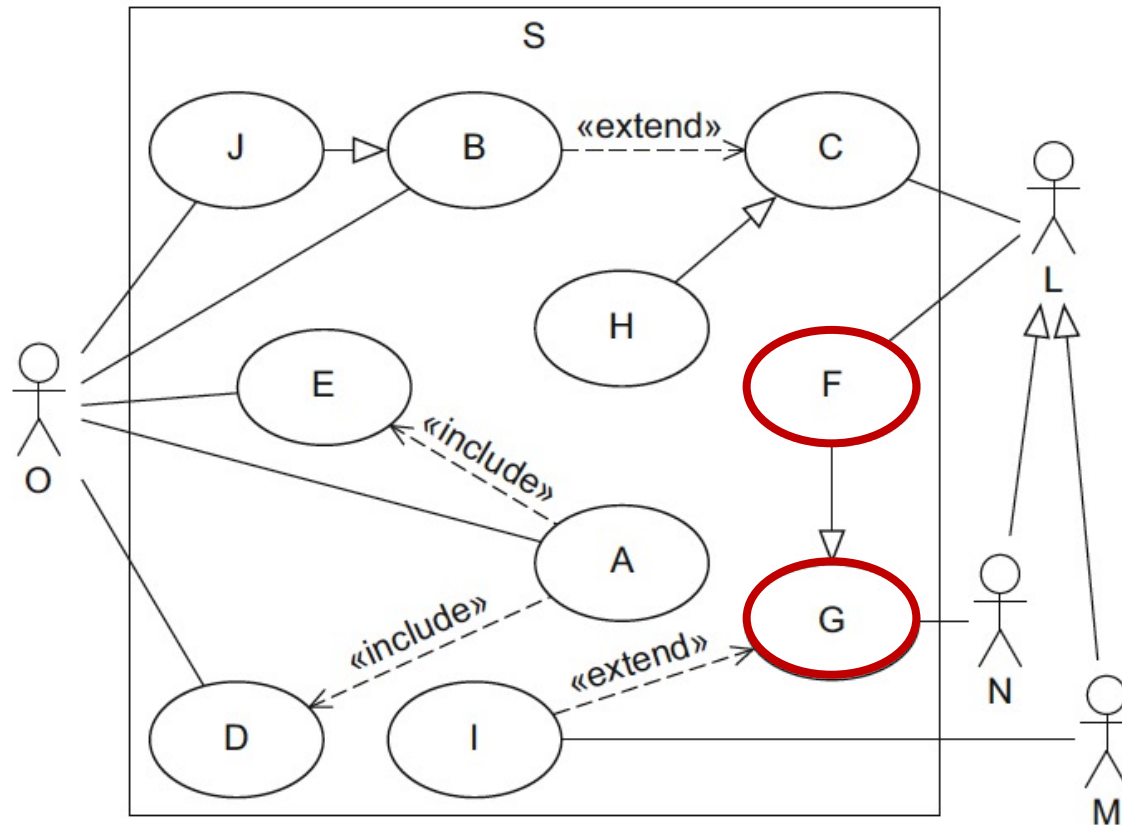
The use case H inherits from the use case C. As use case C is executed by the actor L, an actor L must also be involved in the execution of H. The actors N and M inherit from L. Therefore, both use cases C and H can also be executed by an actor M or N.

Examples of Relationships



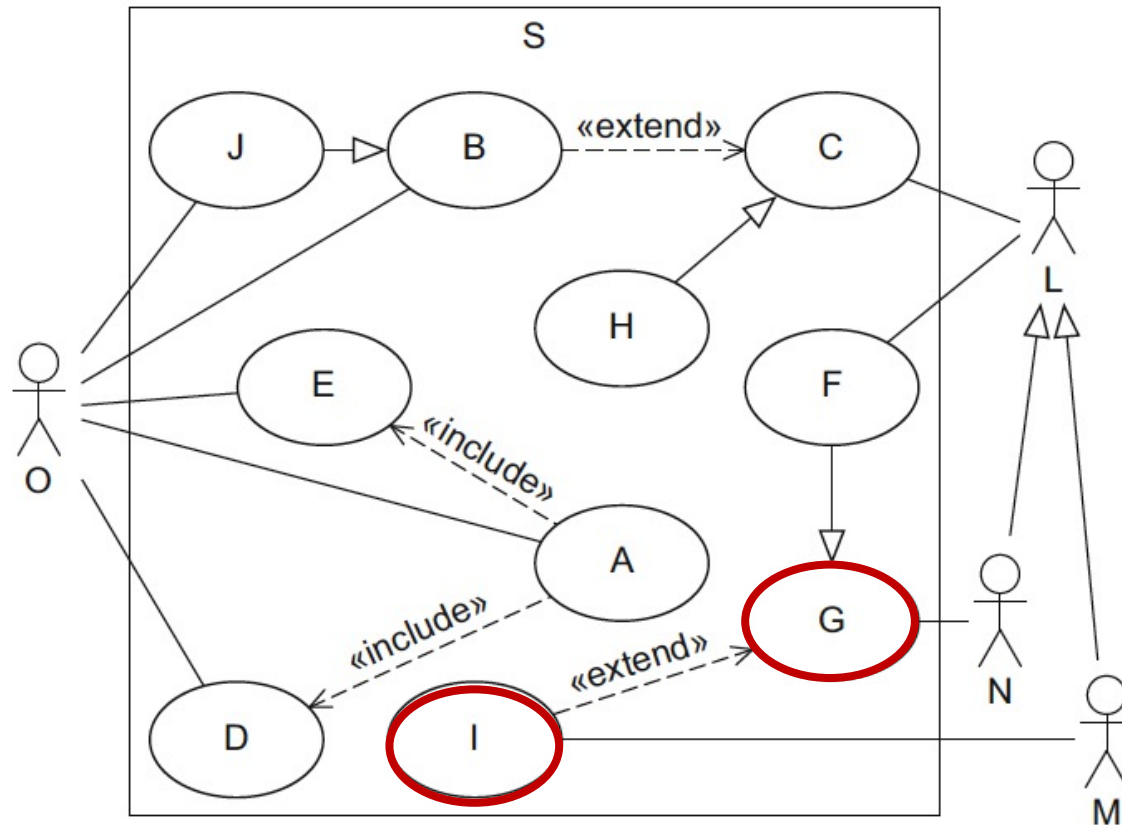
The use case J inherits from the use case B. As a result of the inheritance relationship, an actor O is involved in the execution of use case J. However, an association with O is also modeled for J directly. The consequence of this is that two actors in the role O are involved in the execution of J. Note that these two actors can coincide.

Examples of Relationships



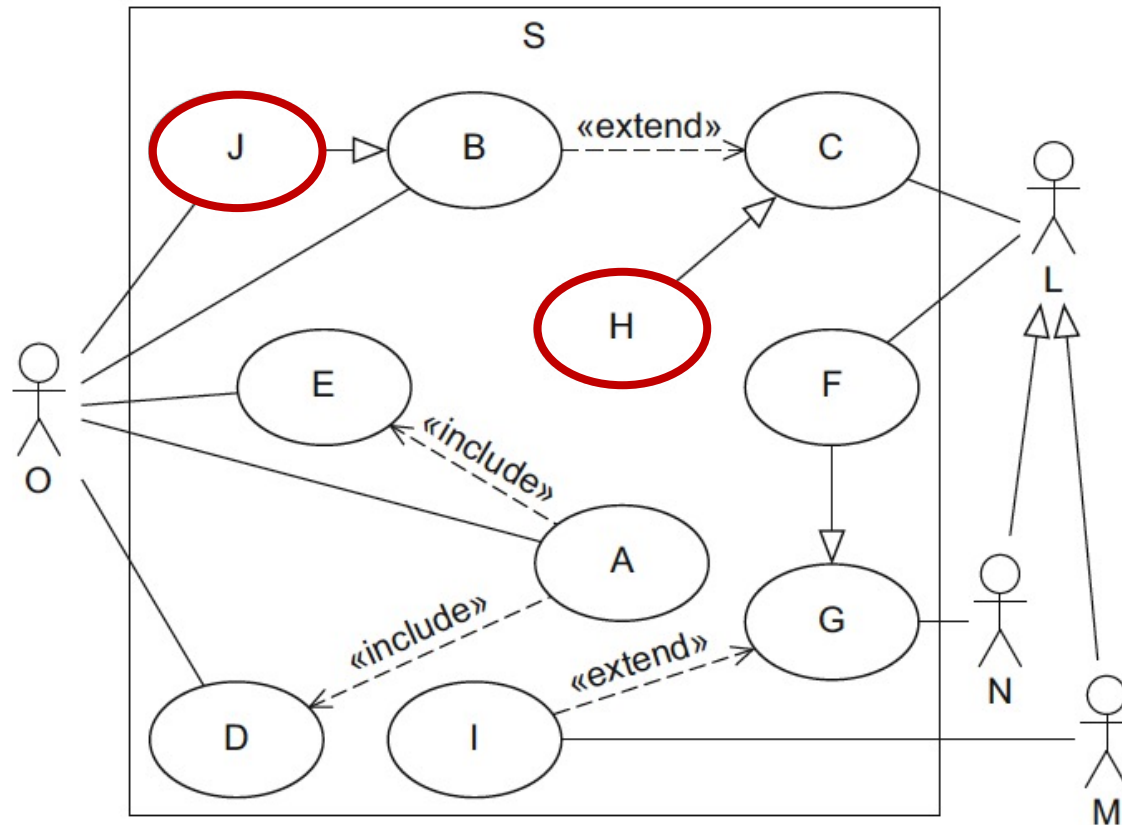
The use case F inherits from the use case G. As a result of the inheritance relationship, an actor N is involved in the execution of use case F. For F, an association with the actor L is also modeled directly. Therefore, an actor N and, due to the inheritance relationship of the actors L, N, and M, either an actor L or an actor M or an additional actor N is involved in the execution of F. If two actors N are involved, they may coincide.

Examples of Relationships



The use case I extends the use case F. As use case F inherits from use case G and as I extends use case G, this relationship is passed on to F. If G and I were in an «include» relationship, this relationship would also be passed on to F in the same way.

Examples of Relationships



The use case J extends the use case H. This is as a result of the inheritance relationships from B to J and from C to H.

Creating a Use Case Diagram

- First you must **identify actors** and **use cases** and then place them in **relationships** with one another. You then **describe** the use cases in detail.
- At first glance, this diagram seems to be simple due to the low number of concepts involved. But in fact, use case diagrams are often created incorrectly with a lot of errors.

Identifying Actors and Use Cases

- There are two ways to identify use cases for prospective system design:
 - Analysis of requirements documents
 - Analysis of the expectations of future users
- Requirements documents are generally natural language specifications that explain what the customer expects from a system. They should document relatively precisely who will use the system and how they will use it. If you follow the second approach for finding use cases, you must first identify the future users—that is, the actors.
- To identify the actors that appear in a use case diagram, you must answer the following questions:
 - 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?

Identifying Actors and Use Cases

- Once you know the actors, you can derive the use cases by asking the following questions about the actors:
 - 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?

Identifying Actors and Use Cases

- In many cases, you model use cases **iteratively** and **incrementally**. In doing so, you often start with the “top level” requirements that reflect the business objectives to be pursued with the software.
- You then continue to refine them until, at a technical level, you have specified what the system should be able to do.
- For example, a “top level” requirement for a university administration system could be that the system can be used for *student administration*. If we refine this requirement, we define that new students should be able to **register at the university** and **enroll for studies**.

Description of Use Cases

- To ensure that even large use case diagrams remain clear, it is extremely important to select **short, concise names** for the use cases.
- When situations arise in which the intention behind the use case and its interpretation are not clear, you must also **describe the use cases**.
- It is important to ensure that you describe the use cases clearly and concisely, as otherwise there is a risk that readers will only skim over the document.

Description of Use Cases

- Structured approach
 - Name
 - Short description
 - **Precondition**: prerequisite for successful execution
 - **Postcondition**: system state after successful execution
 - **Error situations**: errors relevant to the problem domain
 - System state on the occurrence of an error
 - Actors that communicate with the use case
 - **Trigger**: events which initiate/start the use case
 - **Standard process**: individual steps to be taken
 - **Alternative processes**: deviations from the standard process

[A. Cockburn: Writing Effective Use Cases, Addison Wesley, 2000]

Description of Use Cases – Example*

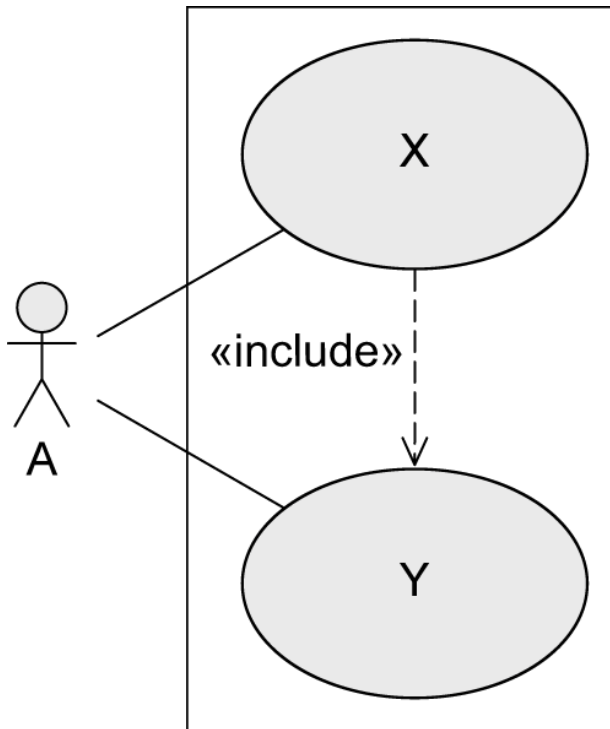
Name:	Reserve lecture hall
Short description:	An employee reserves a lecture hall at the university for an event.
Precondition:	The employee is authorized to reserve lecture halls. Employee is logged in to the system.
Postcondition:	A lecture hall is reserved.
Error situations:	There is no free lecture hall.
System state in the event of an error:	The employee has not reserved a lecture hall.
Actors:	Employee
Trigger:	Employee requires a lecture hall.
Standard process:	(1) Employee selects the lecture hall. (2) Employee selects the date. (3) System confirms that the lecture hall is free. (4) Employee confirms the reservation.
Alternative processes:	(3') Lecture hall is not free. (4') System proposes an alternative lecture hall. (5') Employee selects the alternative lecture hall and confirms the reservation.

* The description is extremely simplified but fully sufficient for our purposes.

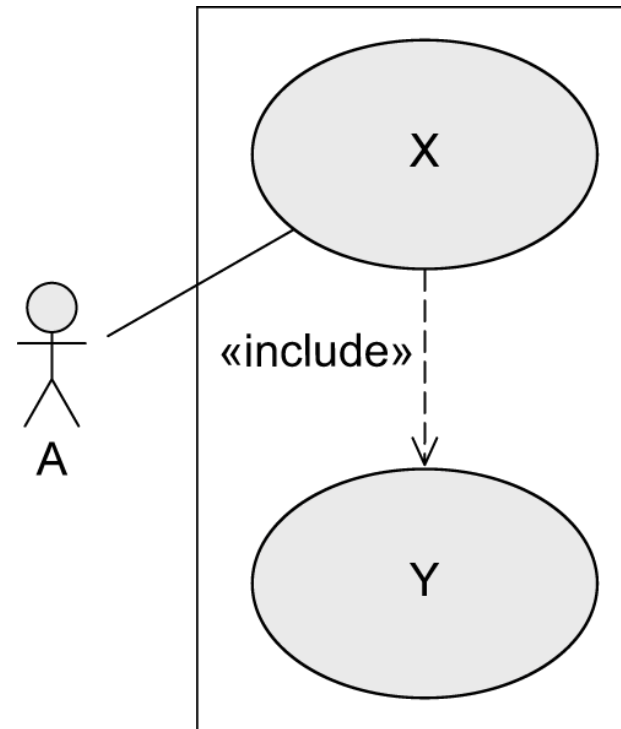
Best Practices

«include»

UML standard



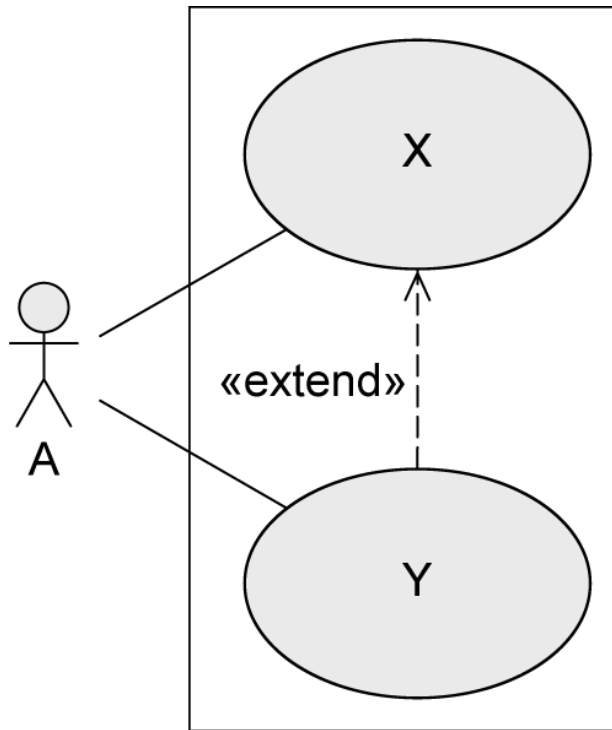
Best practice



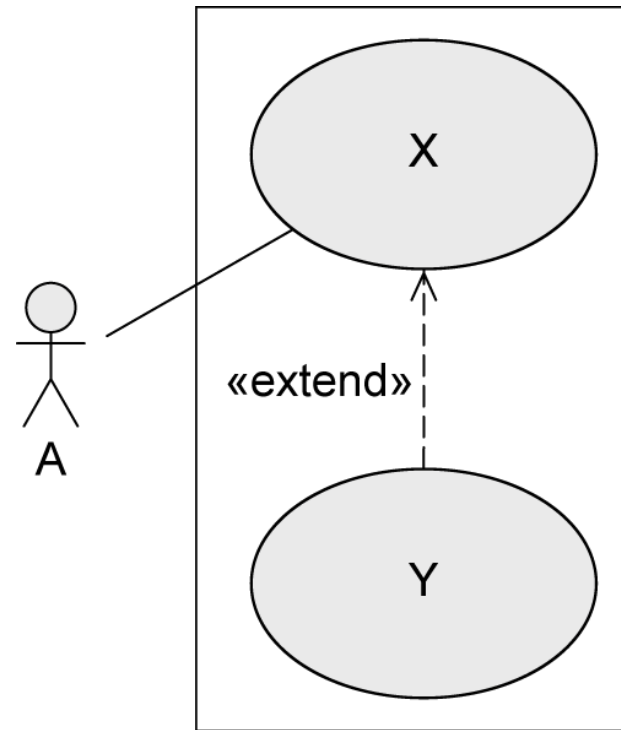
Best Practices

«extend»

UML standard



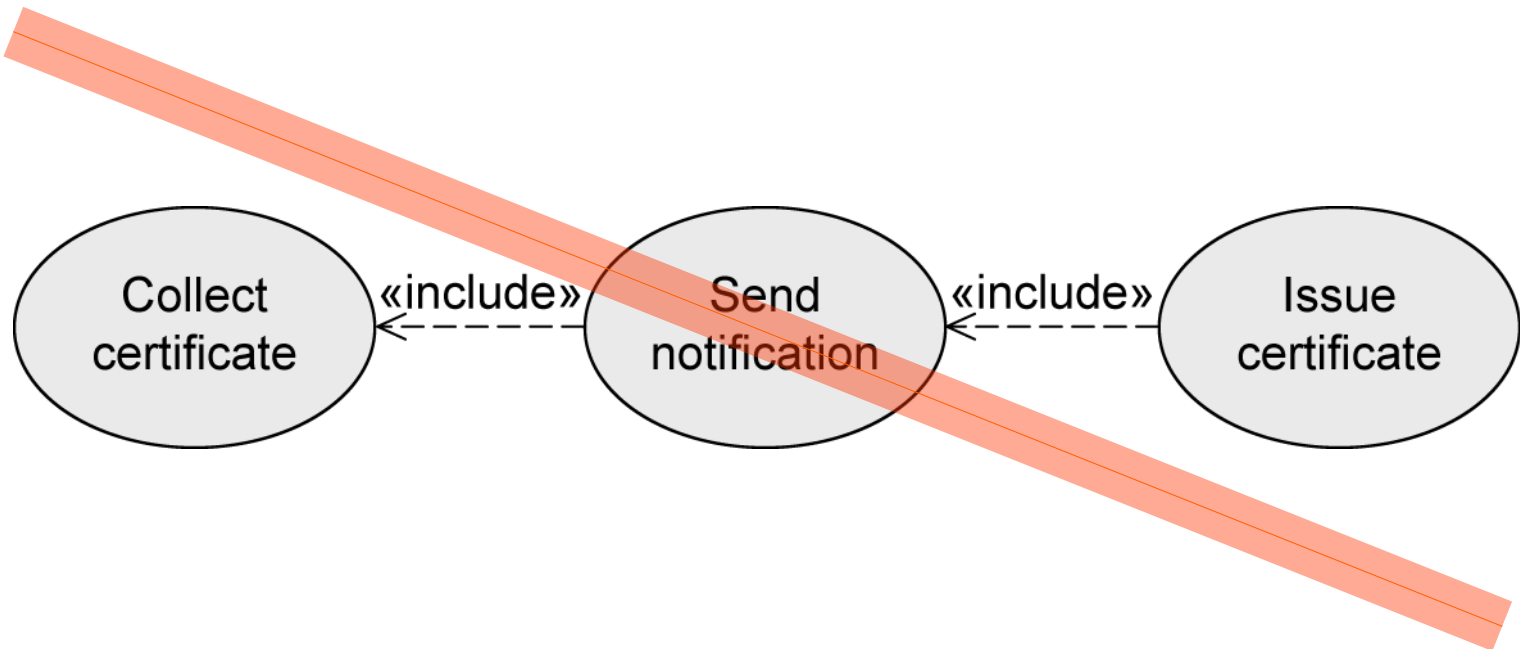
Best practice



Best Practices

Error 1: Modeling processes

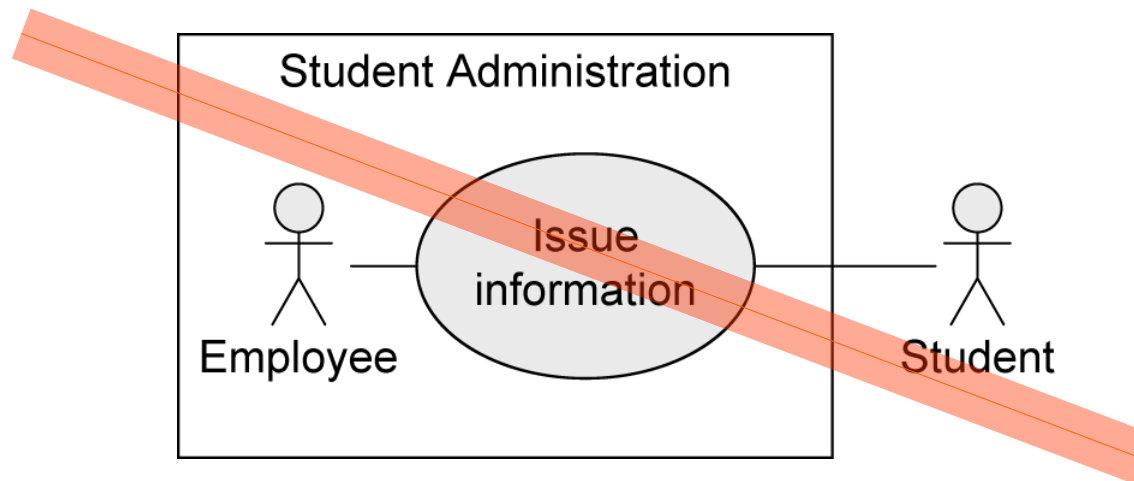
- Use case diagrams do not model processes/workflows!
- The functionality that one of these use cases offers is **not** part of the functionality that another use case offers, hence the use cases must be used independently of one another.



Best Practices

Error 2: Setting system boundaries incorrectly

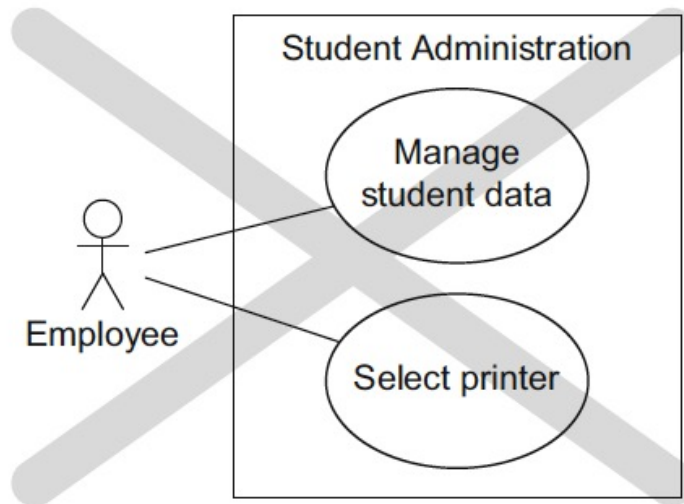
- When modeling a use case diagram, you must consider very carefully where to **draw the boundaries of the diagram**.
- Actors are not part of the system; hence, they are positioned outside the system boundaries!



Best Practices

Error 3: Mixing abstraction levels

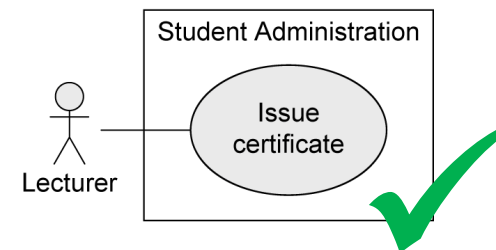
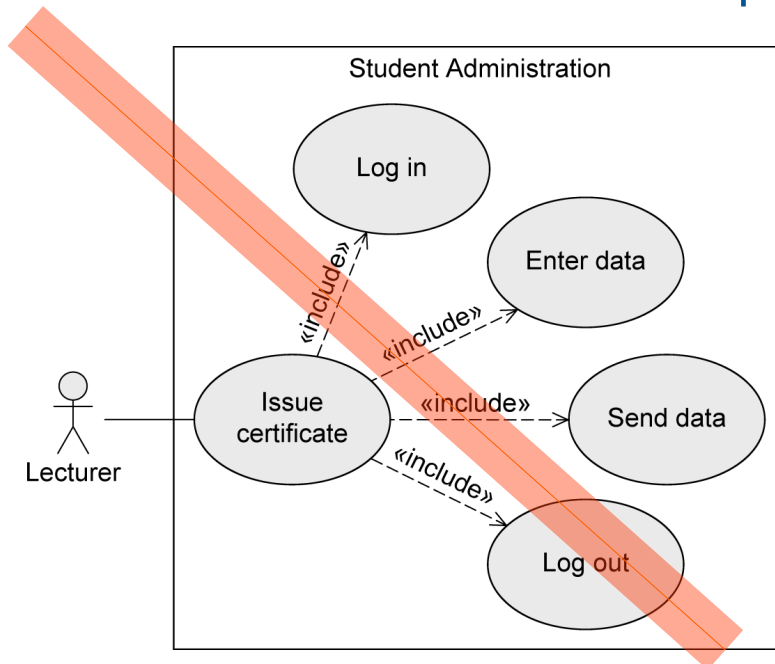
- When identifying use cases, you must always ensure that they are located on the **same abstraction level**.
- Avoid representing “top level” use cases with technically oriented use cases in the same diagram.



Best Practices

Error 4: Functional decomposition

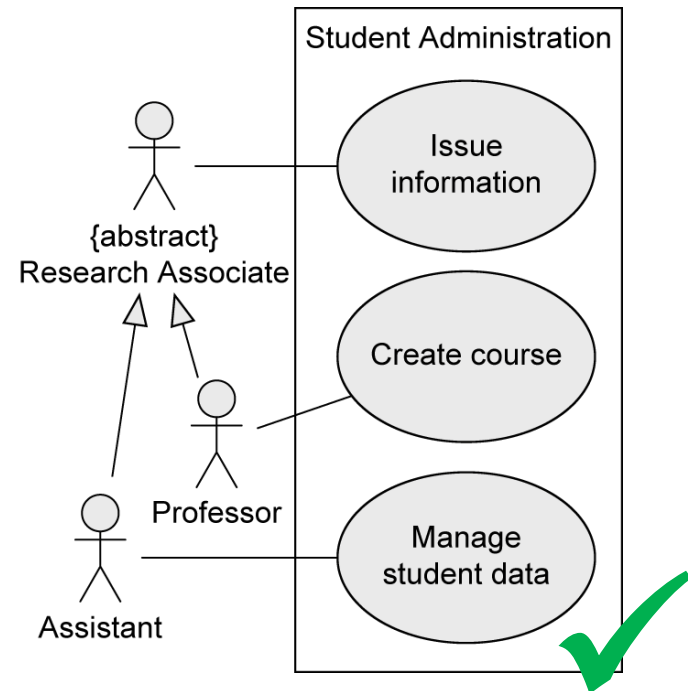
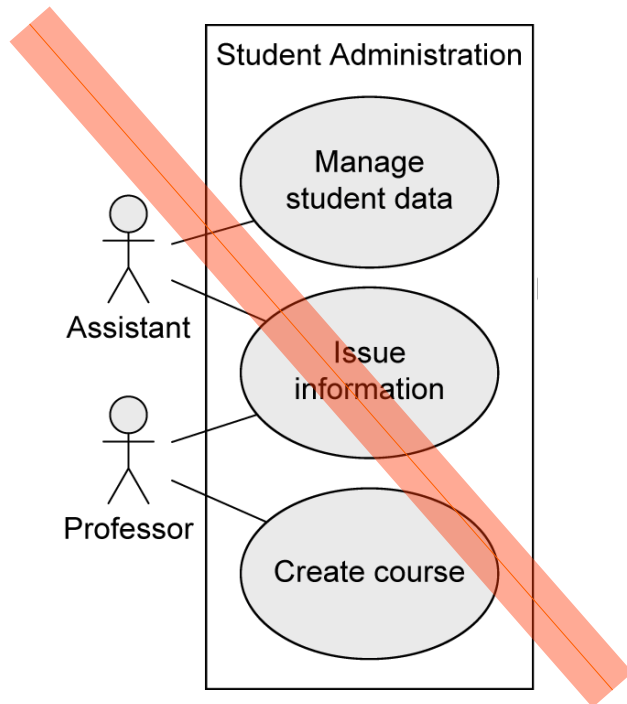
- Use cases—even included or extending use cases—can always be executed independently. If they can only be executed within the scope of another use case and not independently, they are not use cases and must not be depicted as such.
 - Their functionality must then be covered in the description of the use case that uses them.
- The various steps are part of the use cases, **not separate use cases themselves!** -> NO functional decomposition



Best Practices

Error 5: Incorrect associations

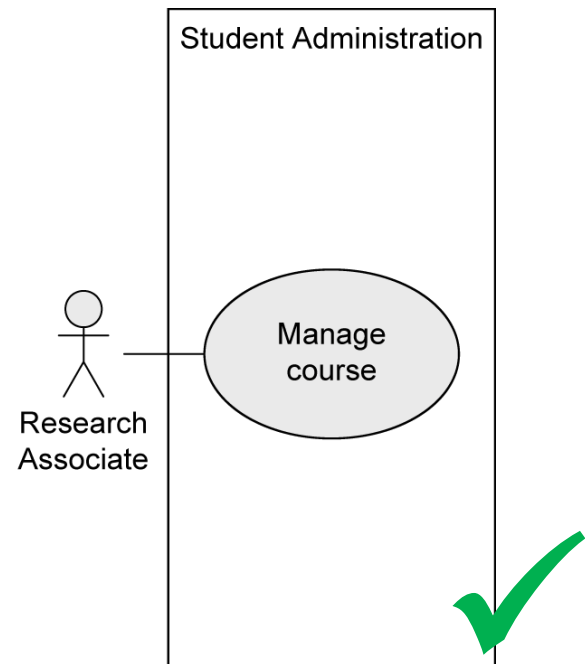
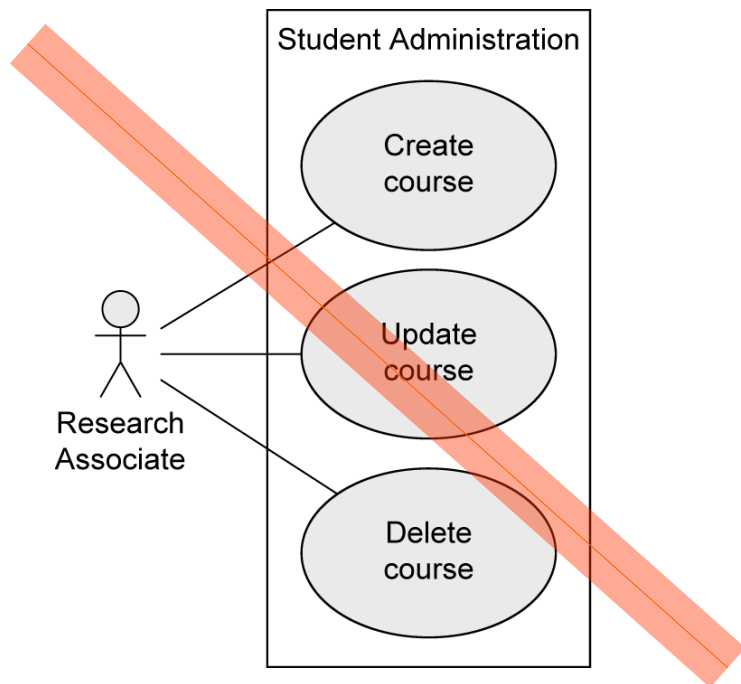
- If a use case is **associated with two actors**, this does not mean that either one or the other actor is involved in the execution of the use case: it means that both are **necessary for its execution**.
- Use case **Issue information** needs **EITHER** one actor **Assistant** **OR** one actor **Professor** for execution



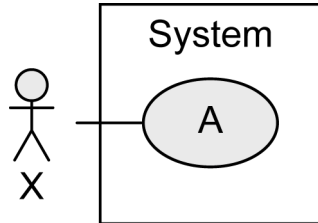

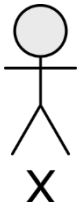
Best Practices

Error 6: Modeling redundant use cases

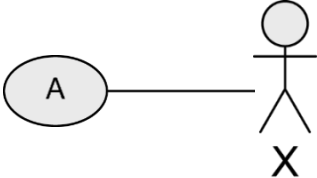
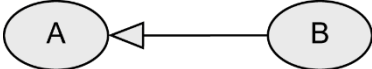
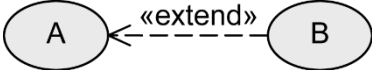
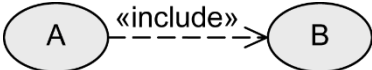
- When modeling use cases, it is very tempting to create a separate use case for each possible situation. When modeling a real application, the diagram would very quickly become unmanageable.
- Many small use cases that have the same objective may be grouped to form one use case.
- The individual steps are then specified in the description of the standard process.



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

Notation Elements (2/2)

Name	Notation	Description
Association		Relationship between use cases and actors
Generalization		Inheritance relationship between actors or use cases
Extend relationship		B extends A: optional use of use case B by use case A
Include relationship		A includes B: required use of use case B by use case A

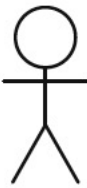
In Class Activity: Information System of a Student Office

- Many important administrative activities of a university are processed by the student office. Students can register for studies (matriculation), enroll, and withdraw from studies here. Matriculation involves enrolling, that is, registering for studies.
- Students receive their certificates from the student office. The certificates are printed out by an employee. Lecturers send grading information to the student office. The notification system then informs the students automatically that a certificate has been issued.
- There is a differentiation between two types of employees in the student office: a) those that are exclusively occupied with the administration of student data (service employee, or ServEmp), and b) those that fulfill the remaining tasks (administration employee, or AdminEmp), whereas all employees (ServEmp and AdminEmp) can issue information.
- Administration employees issue certificates when the students come to collect them. Administration employees also create courses. When creating courses, they can reserve lecture halls

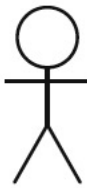
In Class Activity: Identifying Actors

- Many important administrative activities of a university are processed by the student office. **Students** can register for studies (matriculation), enroll, and withdraw from studies here. Matriculation involves enrolling, that is, registering for studies.
- **Students** receive their certificates from the student office. The certificates are printed out by an **employee**. **Lecturers** send grading information to the student office. The **notification system** then informs the **students** automatically that a certificate has been issued.
- There is a differentiation between two types of **employees** in the student office: a) those that are exclusively occupied with the administration of student data (**service employee**, or **ServEmp**), and b) those that fulfill the remaining tasks (**administration employee**, or **AdminEmp**), whereas all **employees** (**ServEmp** and **AdminEmp**) can issue information.
- **Administration employees** issue certificates when the **students** come to collect them. **Administration employees** also create courses. When creating courses, they can reserve lecture halls.

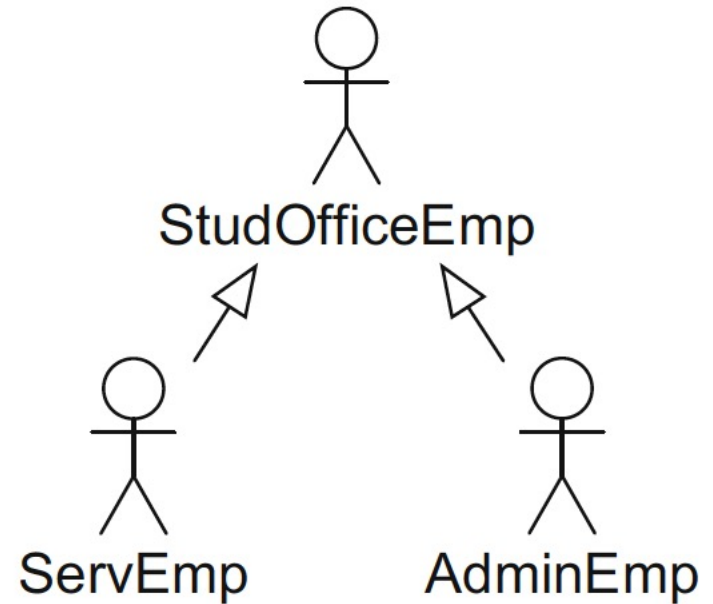
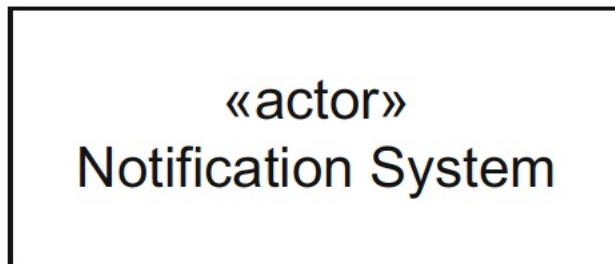
In Class Activity: Identifying Actors



Lecturer



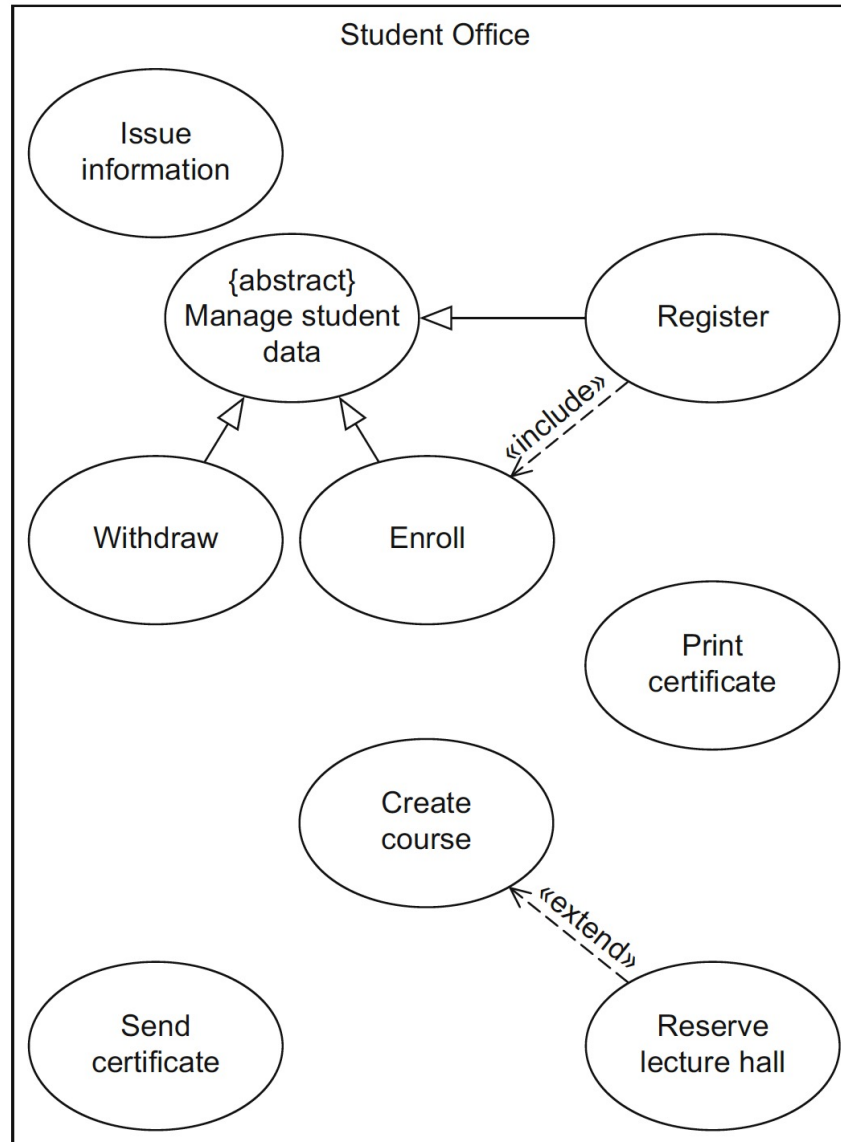
Student



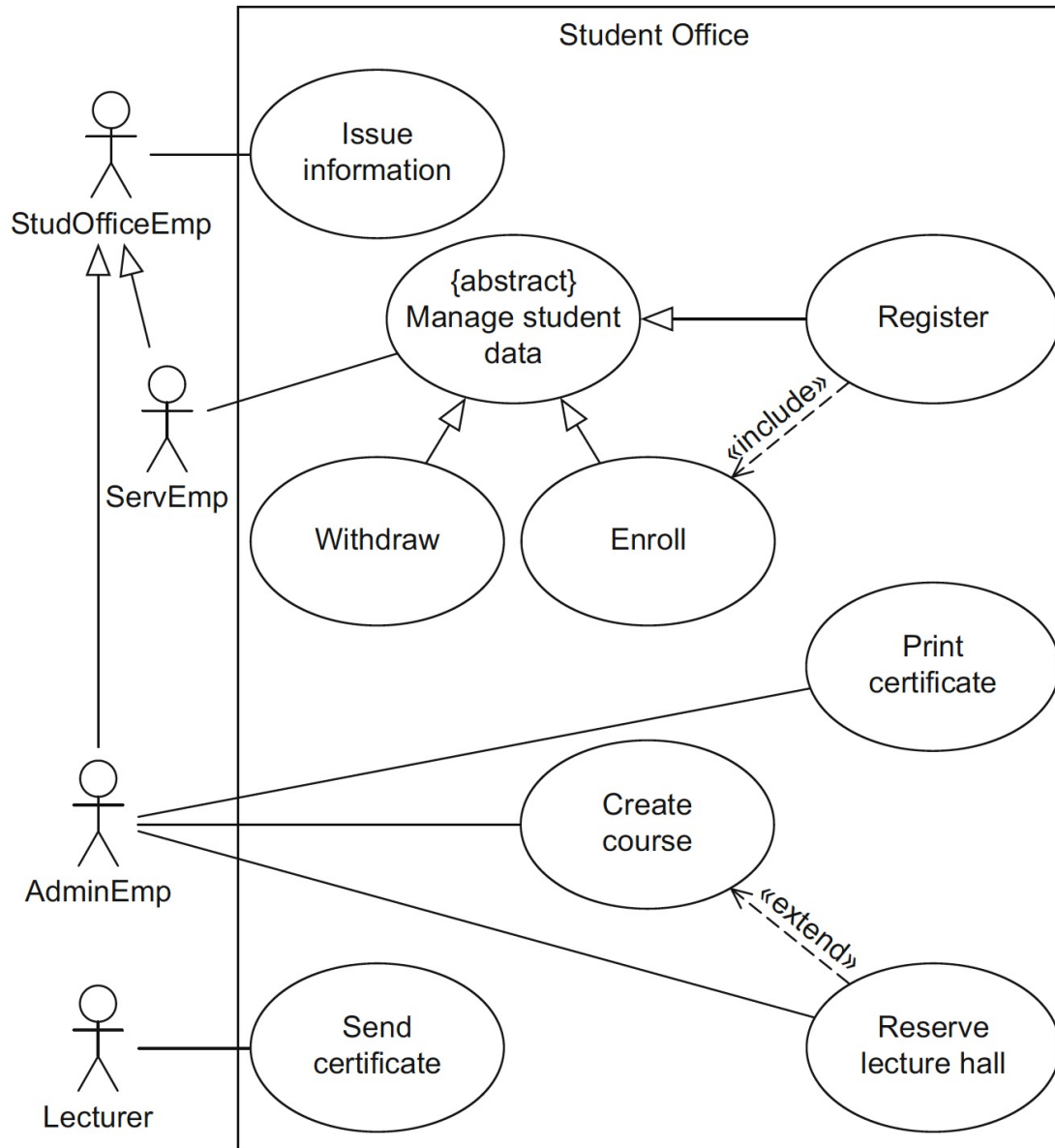
In Class Activity: Identifying Use Cases

- Many important administrative activities of a university are processed by the student office. **Students** can *register for studies* (matriculation), *enroll*, and *withdraw* from studies here. *Matriculation involves enrolling, that is, registering for studies.*
- **Students** *receive their certificates* from the student office. The certificates are *printed out* by an **employee**. **Lecturers** *send grading information* to the student office. The **notification system** then informs the **students** automatically that a certificate has been issued.
- There is a differentiation between two types of **employees** in the student office: a) those that are exclusively occupied with the administration of student data (**service employee**, or **ServEmp**), and b) those that fulfill the remaining tasks (**administration employee**, or **AdminEmp**), whereas all **employees** (**ServEmp** and **AdminEmp**) can issue information.
- **Administration employees** *issue certificates* when the **students** come to collect them. **Administration employees** also *create courses*. When creating courses, they can *reserve lecture halls*.

In Class Activity: Identifying Use Cases



In Class Activity: Identifying Associations



In Class Activity: Describing the Use Cases

Name:	Print certificate
Short description:	On request from a student, an employee prints the student's certificate for a course on paper.
Precondition:	All data relevant for the certificate has been sent and the student has been graded.
Postcondition:	Certificate is available to the student in printed form.
Error situations:	Printer is not working.
System state in the event of an error:	Certificate is not printed.
Actors:	AdminEmp
Trigger:	Student requests printed certificate.
Standard process:	(1) Student enters the student office and requests a certificate. (2) AdminEmp enters the student's matriculation number. (3) AdminEmp selects the certificate. (4) AdminEmp enters the print command. (5) System confirms that the certificate was printed. (6) Certificate is handed over to the student.
Alternative processes:	(1') Student requests certificate via e-mail. (2-5) As above (6') Certificate is sent by post.