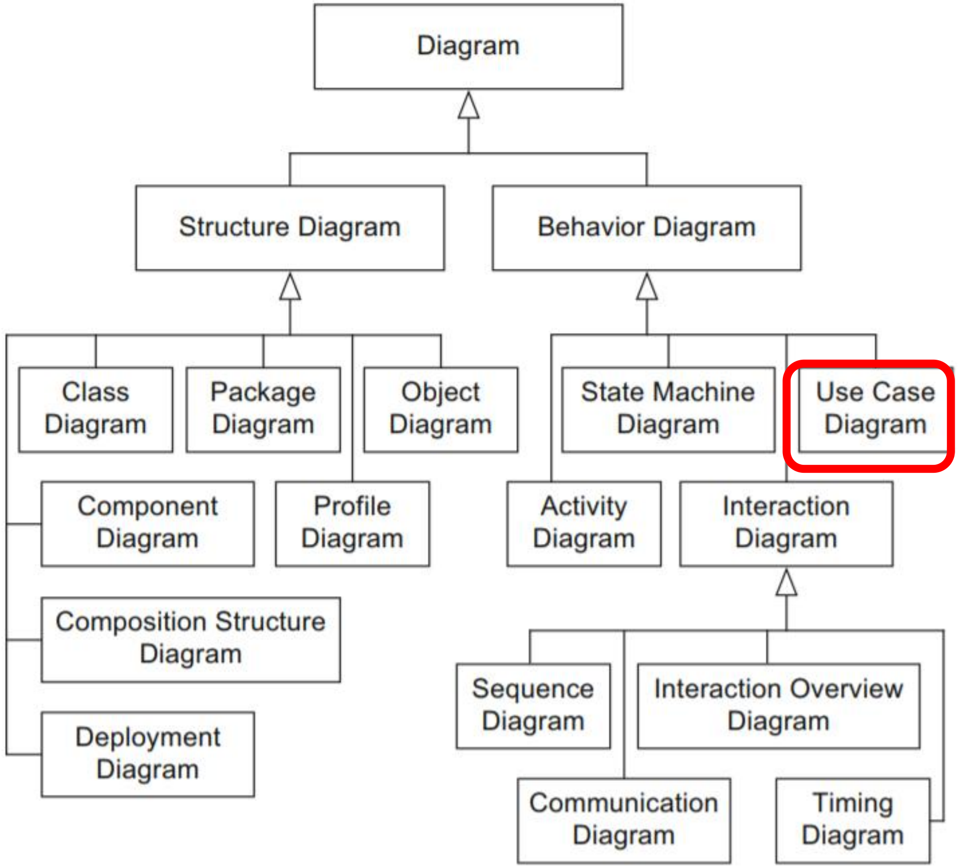
UML Diagrams



Use Cases

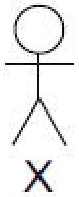
- The customer’s requirements of the system (Functional)
- Who’s interacting with the system?
- How they are using it?

- Use case

  - Verb + Noun
  - i.e. Check deposit
- Use case needs to benefit the actor
  - i.e. Withdraw money is a use case, fill in the withdraw form is not

Actor

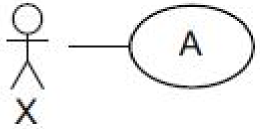
- Actor



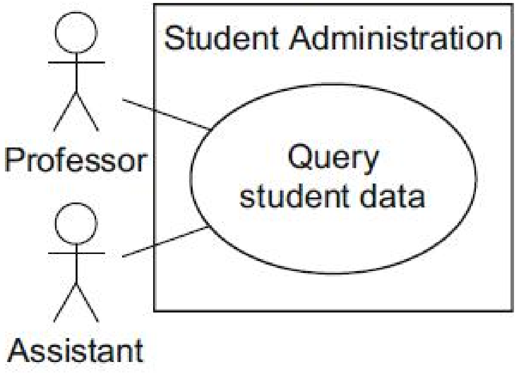
  - Don’t confuse with stakeholder
  - Not all stakeholders are actors in certain use cases
  - Not necessarily human. i.e. server
- Active actor: who initiates the use case
- Primary/secondary actor:
  - Who benefits from executing the use case
- Represents roles, not user
  - One user can perform different role, therefore represents multiple actors

Associations

- Associate actors with use cases

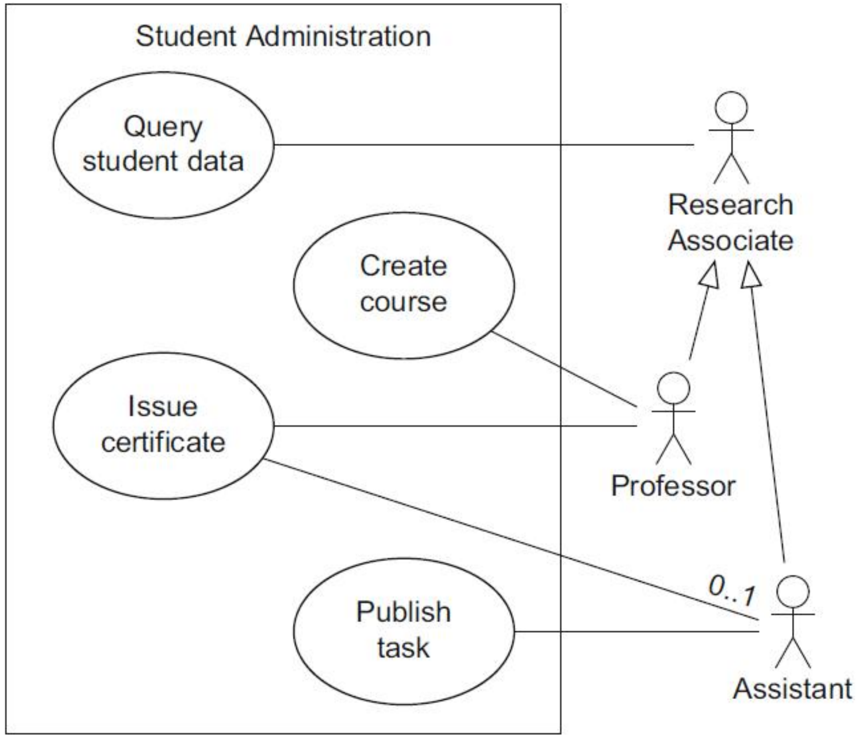


- Every actor should interact with at least one use case
- Every use case should have at least one associated actor
- Actors are outside of system boundary
- Some stakeholders are inside the system as business workers

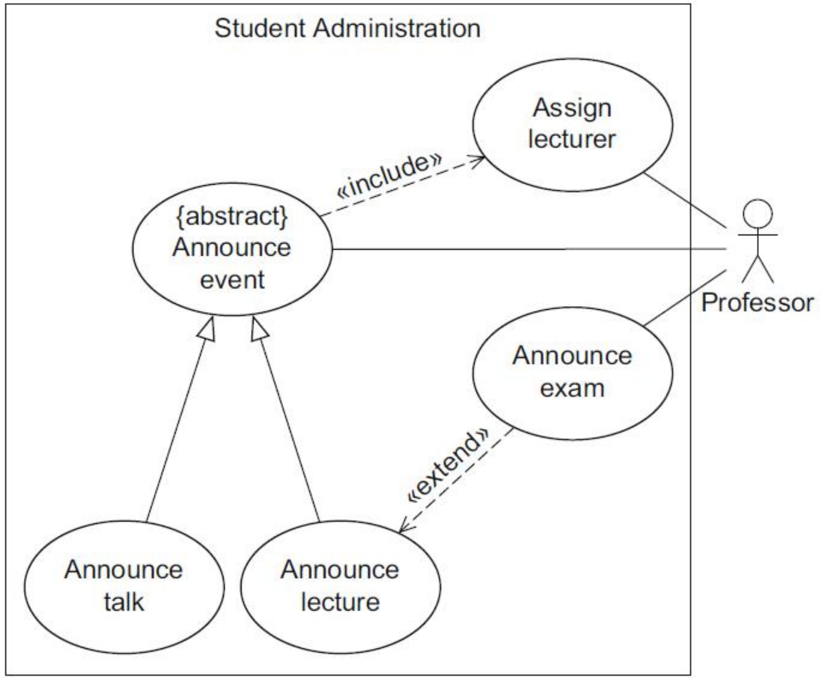


Relationships between Actors

- Generalization/inheritance
- Abstract actor
  - No instance

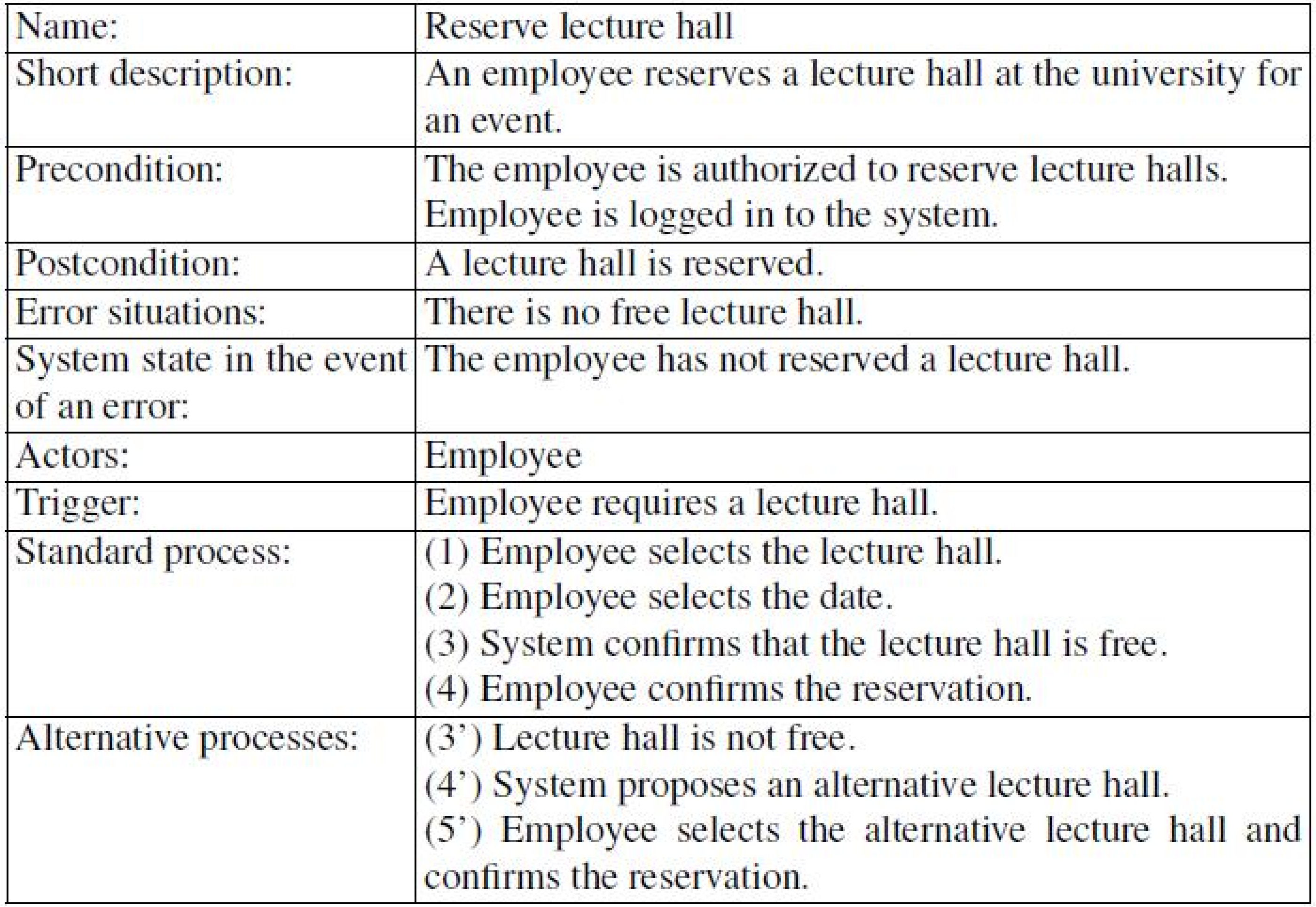


Relationships between Use Cases



- Include
  - The included use case is part of base use case
  - Like a subroutine
  - The included use case can be executed individually
- Extend
  - Optional
  - The extending use case and the base use case both can execute individually
  - The extending use case is triggered by a condition at an extension point
- Generalization

Use Case Document 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.

Identify the actors

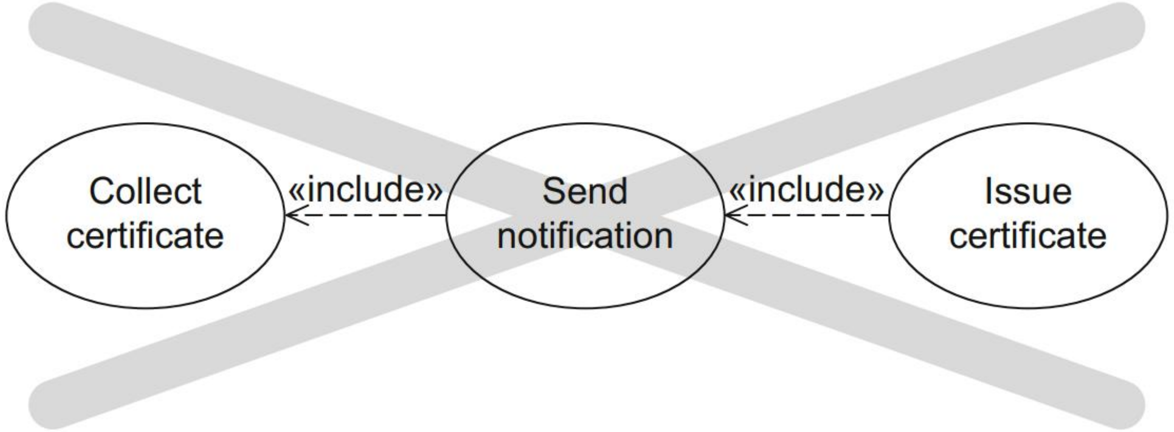
- Who uses the main use cases?
- Who needs support for their daily work?
- Who is responsible for system administration?
- What are the external devices/(software) systems with which the system must communicate?
- Who has an interest in the results of the system?

Derive the use cases

- What are the main tasks that an actor must perform?
- Does an actor want to query or even modify information contained use cases 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?

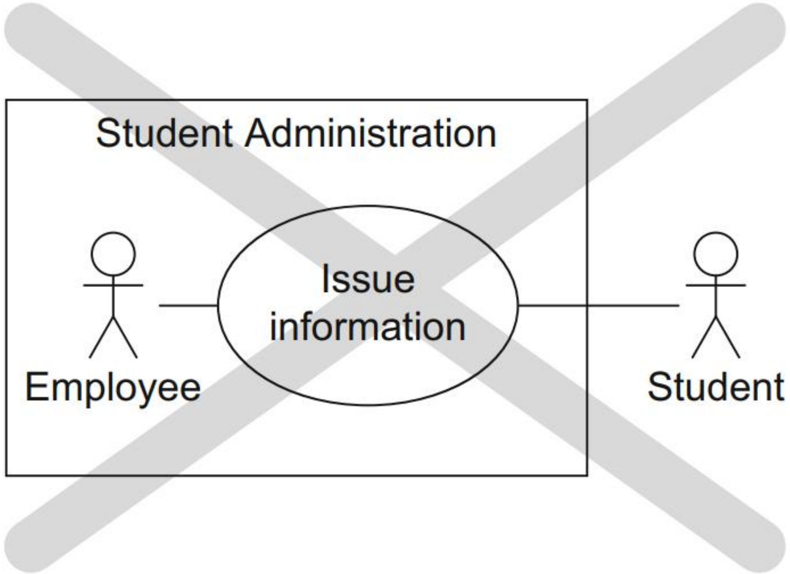
Common Problems

1. Modeling Processes



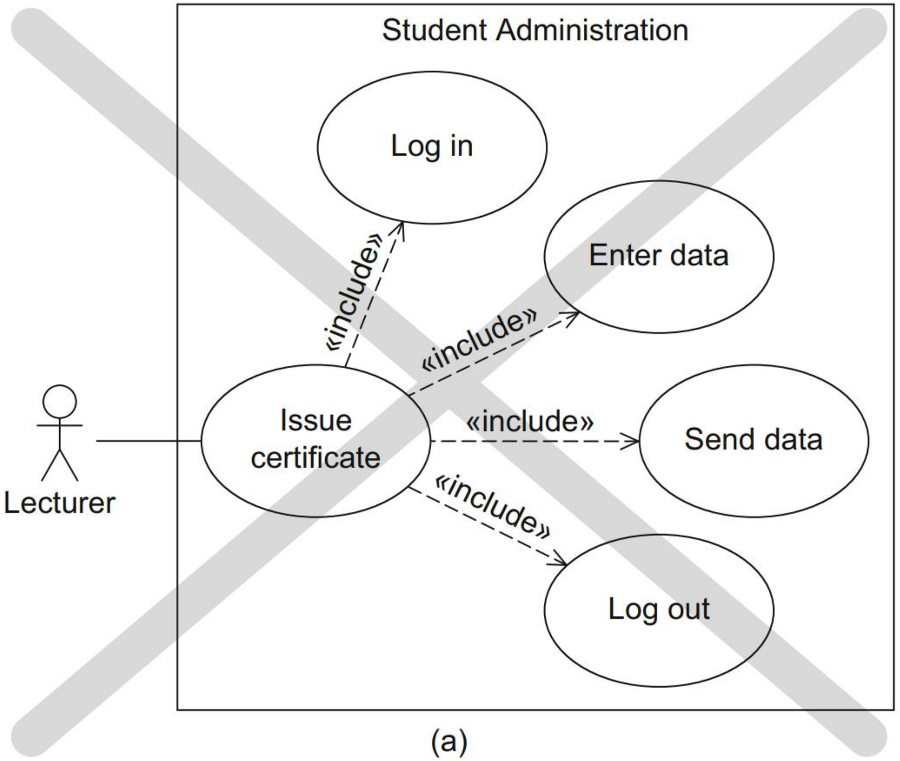
Common Problems

2. Setting wrong system boundary

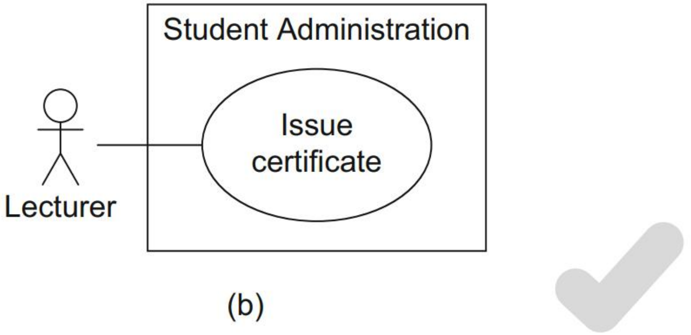


Common Problems

3. Functional decomposition



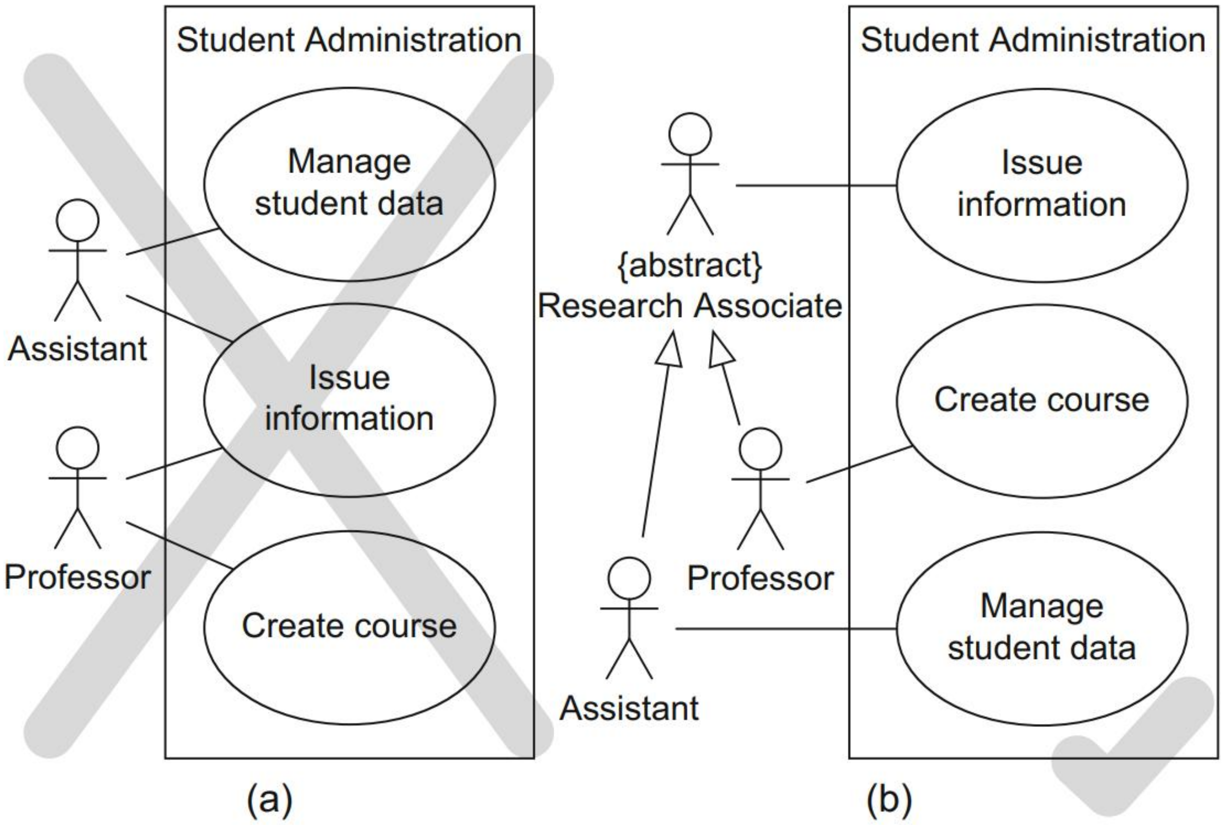
(a)



(b)

Common Problems

4. Incorrect association



Example

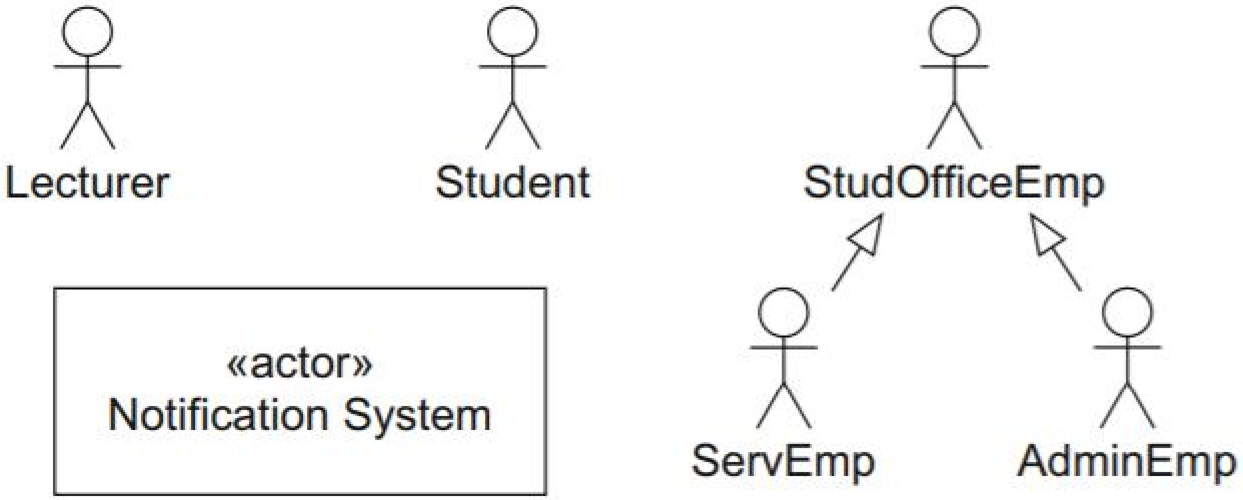
- Many important administrative activities of a university are processed by the student office. Students can register for studies (matriculation), enroll, and withdraw from studies here. Matriculation involves enrolling, that is, registering for studies.
- Students receive their certificates from the 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.

Example

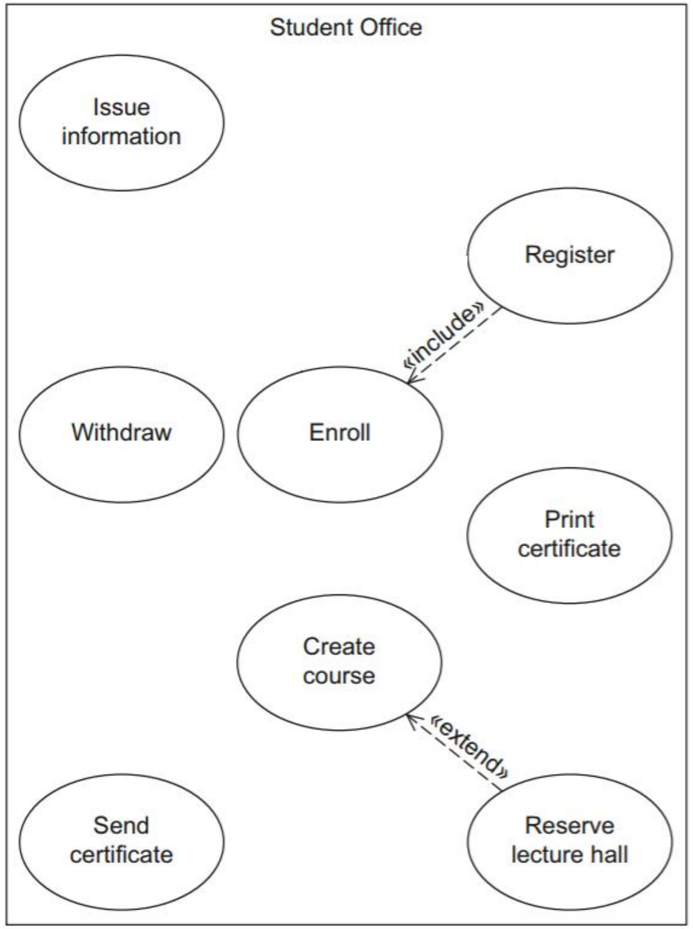
- Many important administrative activities of a university are processed by the student office. Students can register for studies (matriculation), enroll, and withdraw from studies here. Matriculation involves enrolling, that is, registering for studies.
- Students receive their certificates from the 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.

Example: Actors

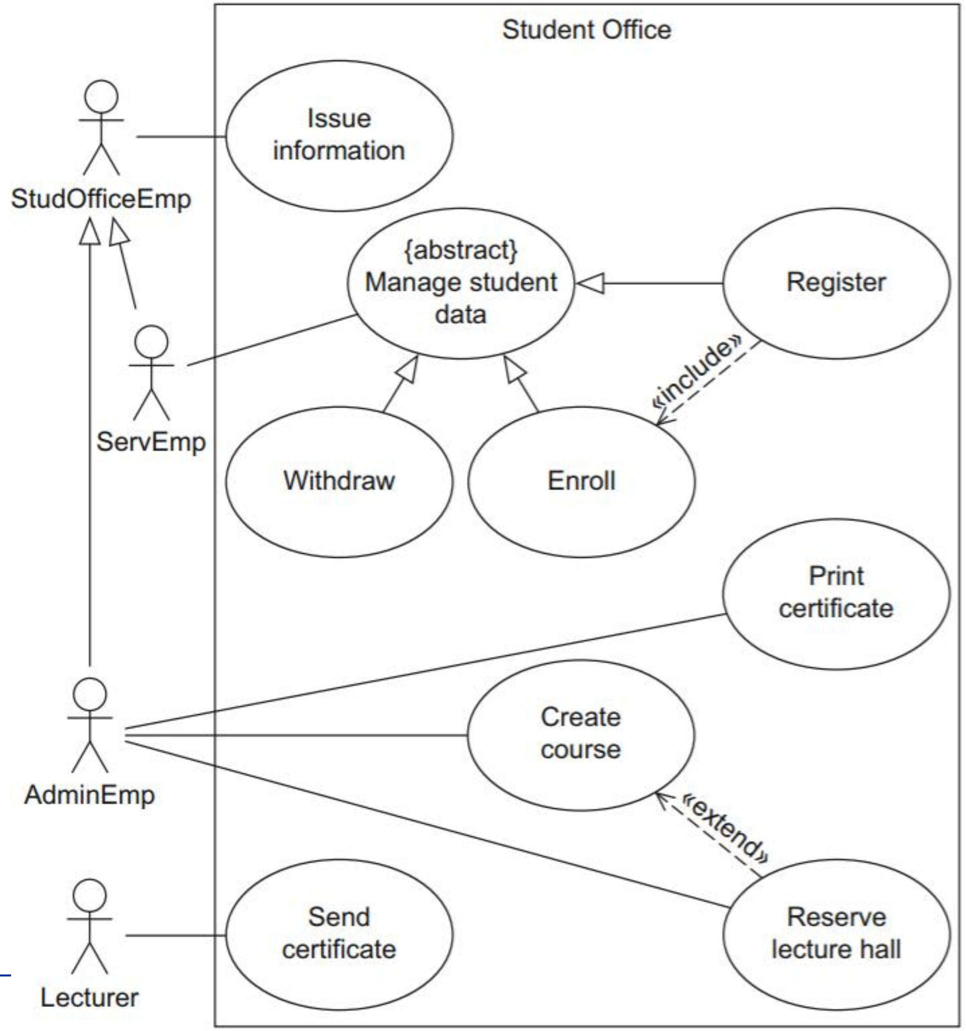
- Students and Notification system are not part of this use case



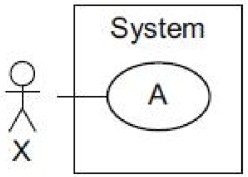
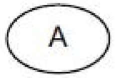
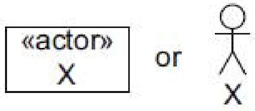
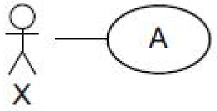
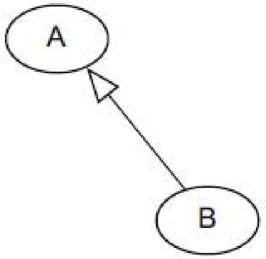
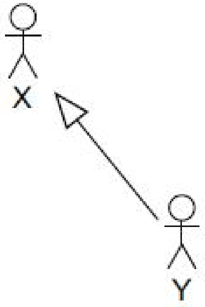
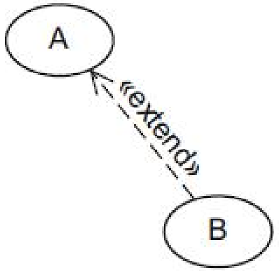
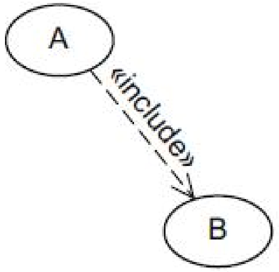
Example: Use Cases



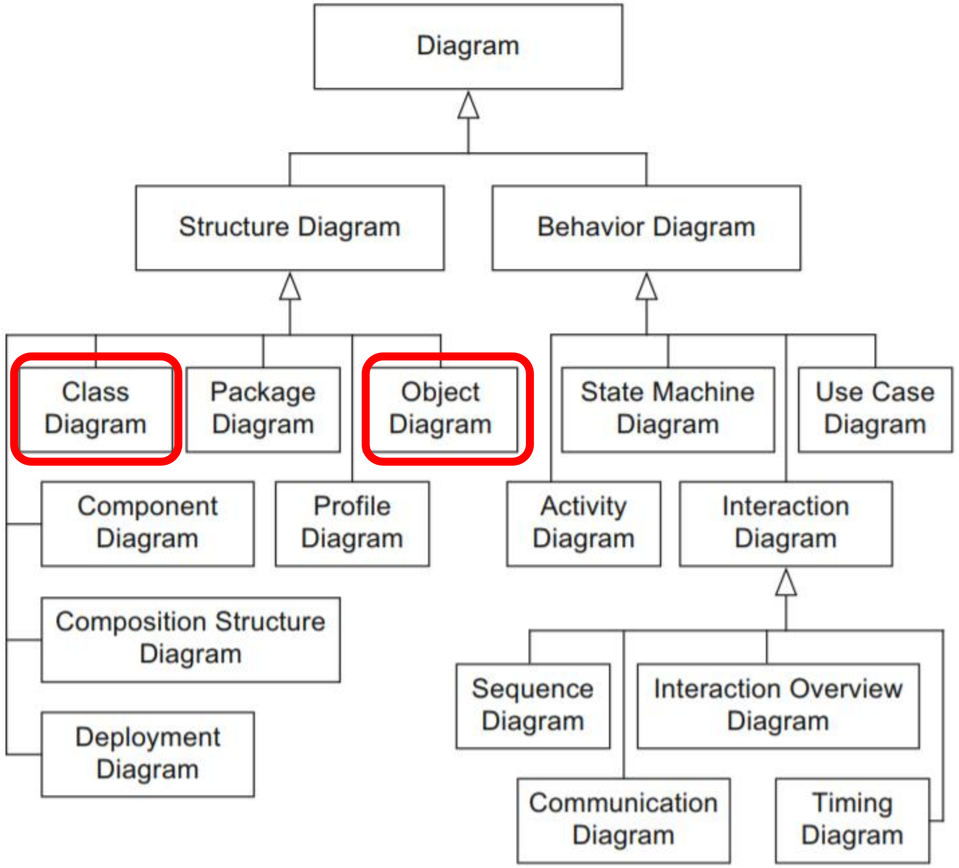
Example


Student Office
StudOfficeEmp
Issue information
{abstract} Manage student data
Register
ServEmp
«include»
Withdraw
Enroll
Print certificate
Create course
AdminEmp
«extend»
Lecturer
Send certificate
Reserve lecture hall

Summary: Use Case Diagram

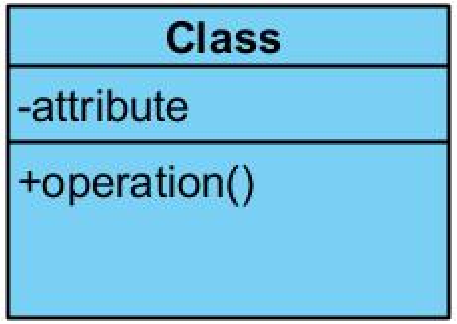
Name	*Notation*	*Description*
System		Boundaries between the system and the users of the system
Use case		Unit of functionality of the system
Actor		Role of the users of the system
Association		X participates in the execution of A
Generalization (use case)		B inherits all properties and the entire behavior of A
Generalization (actor)		Y inherits from X; Y participates in all use cases in which X participates
Extend relationship		B extends A: optional incorporation of use case B into use case A
Include relationship		A includes B: required incorporation of use case B into use case A

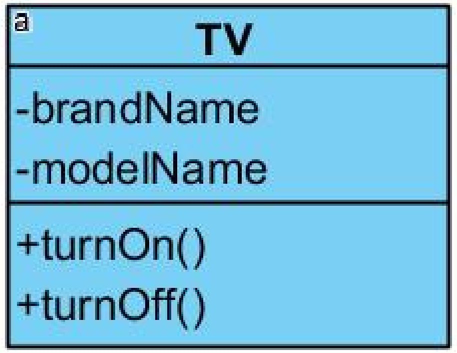
UML Diagrams

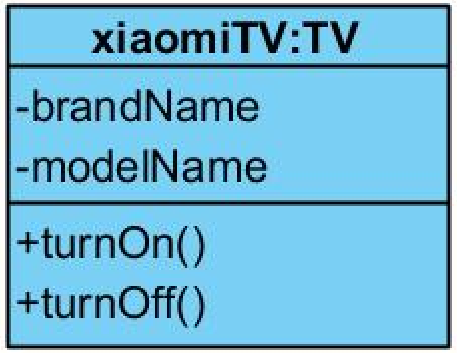


Class and object

- Class: The basic component of object-oriented approaches
- Each class has a list of attributes and operations
- Objects are *instances* of classes
- Visibility:
  - + global: accessible to all
  - - private: accessible within the obj
  - # protected: only accessible by its sub-classes







Attributes

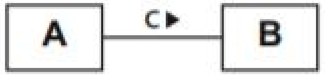
- Name
  - Noun clause, lowercase first letter, then uppercase for latter words
    - i.e. gradStudent, firstName
- Data Type
  - i.e. String
- Multiplicity: how many value it can contain
  - [min .. max]: i.e. [0 .. 1]
- Example: address: String [1..*]= “Huanke Rd 199”
- Which attributes to include depends on the stage of development
  - The closer to implementation, the more detailed the models are

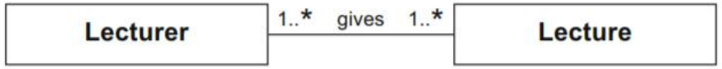
Operations

- Name
  - Verb clause: i.e. getGrade()
- Parameters
  - Direction: in, out, inout
  - Name
  - Data type
- Return value
  - Only need a data type
- Example
  - getName(out fn: String, out in: String): void
  - updateLastName(in newName: String): boolean

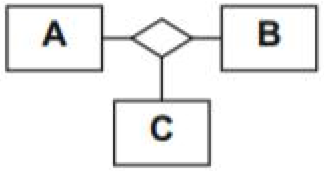
Association

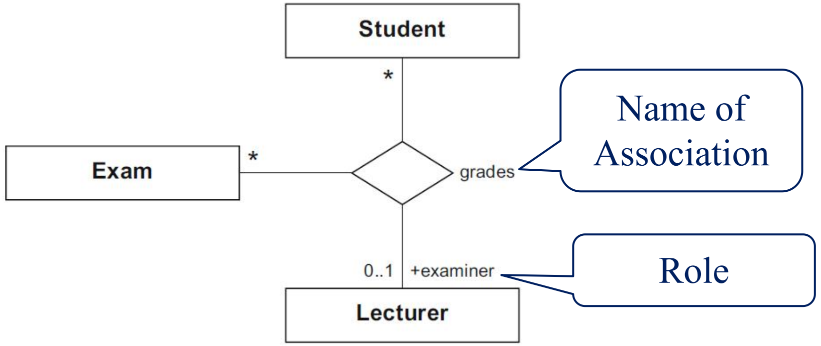
- Possible relationships between instances of the classes.
- Can access attributes and operations with corresponding visibility
- Binary association





- N-nary association

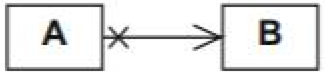




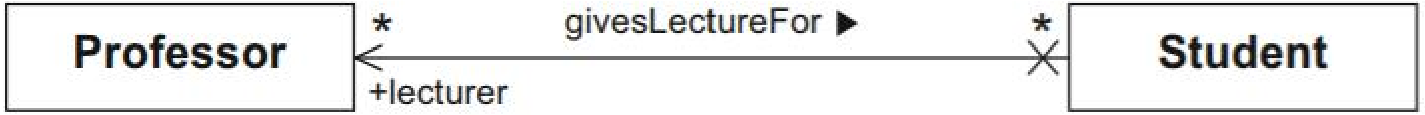
- Multiplicity (Cardinality)
  - The number of objects that may be associated with exactly one object of the opposite side

Navigability

- By default the information sharing is bi-directional
- Non-navigability

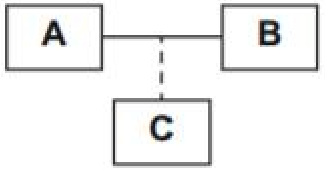


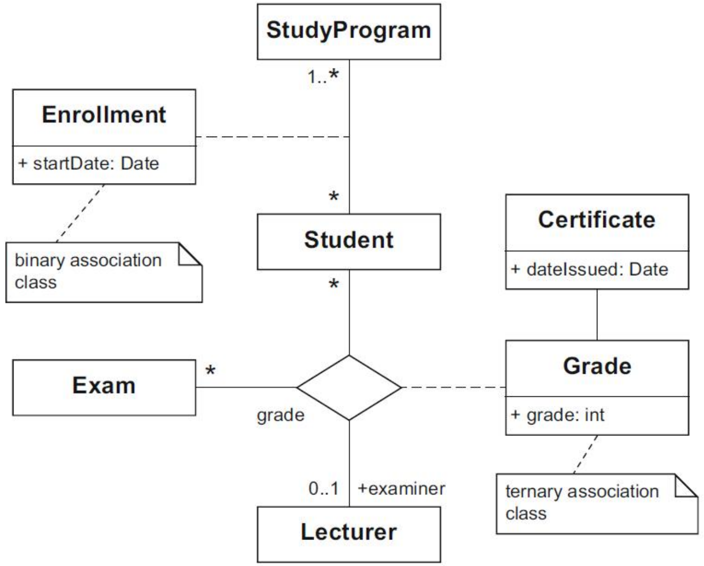
  - A can access visible information of B
  - B cannot access information of A

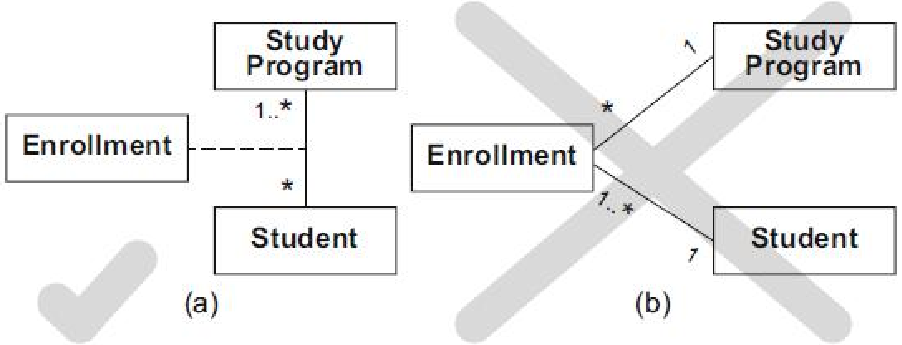


Association Class

- Has the property of both a class and an association
- Cannot be simply replaced by a “normal” class
- In (b), a student can enroll a study program multiple times



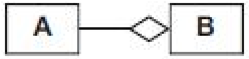


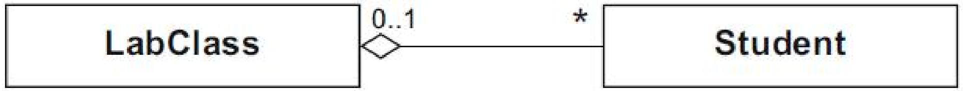


Aggregation

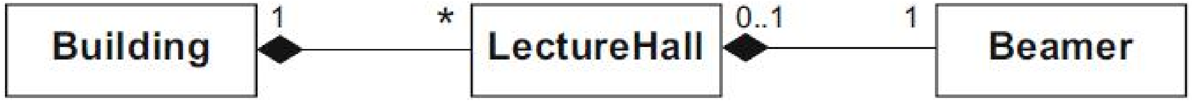
- A special form association: A is part of B

- Shared aggregations



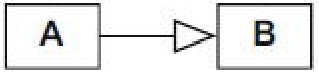


  - A can also be part of something else
  - When B is gone, A can still exist
- Compositions
  - A specific part can only be contained in at most one composite object at one specific point in time.
  - A much stronger bond (normally physical)

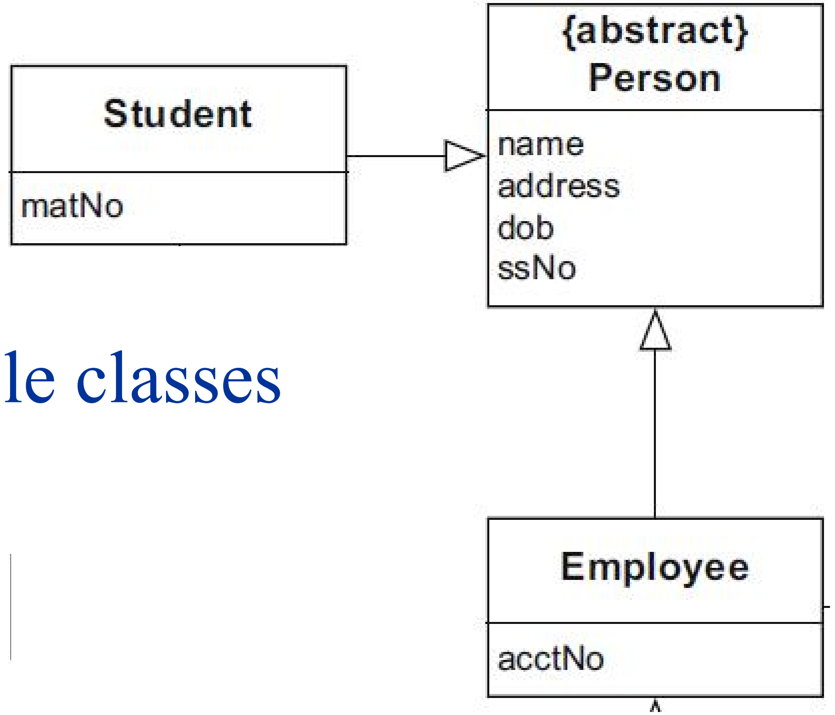


Generalization/Inheritance

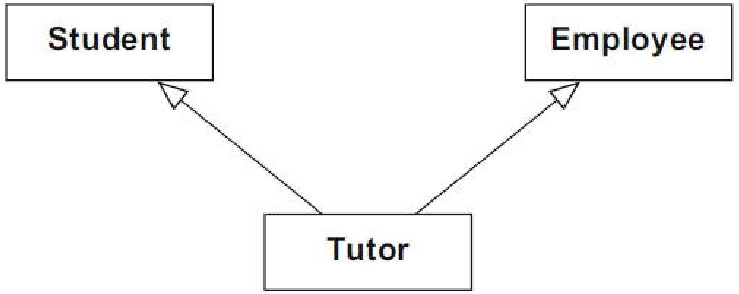
- Highlight common attributes and methods of objects and classes



- Abstract class
  - No instances



- A class can inherit from multiple classes



An Example

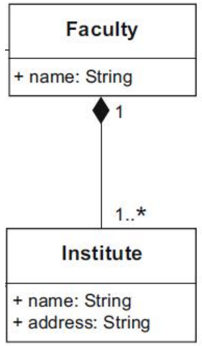
- A university consists of multiple faculties which are composed of various institutes. Each faculty and each institute has a name. An address is known for each institute.
- Each faculty is led by a dean, who is an employee of the university.
- The total number of employees is known. Employees have a social security number, a name, and an e-mail address. There is a distinction between research and administrative personnel.
- Research associates are assigned to at least one institute. The field of study of each research associate is known. Furthermore, research associates can be involved in projects for a certain number of hours, and the name, starting date, and end date of the projects are known. Some research associates teach courses. They are called lecturers.
- Courses have a unique number (ID), a name, and a weekly duration in hours.

An Example

- A university consists of multiple faculties which are composed of various institutes. Each faculty and each institute has a name. An address is known for each institute.
- Each faculty is led by a dean, who is an employee of the university.
- The total number of employees is known. Employees have a social security number, a name, and an e-mail address. There is a distinction between research and administrative personnel.
- Research associates are assigned to at least one institute. The field of study of each research associate is known. Furthermore, research associates can be involved in projects for a certain number of hours, and the name, starting date, and end date of the projects are known. Some research associates teach courses. They are called lecturers.
- Courses have a unique number (ID), a name, and a weekly duration in hours.

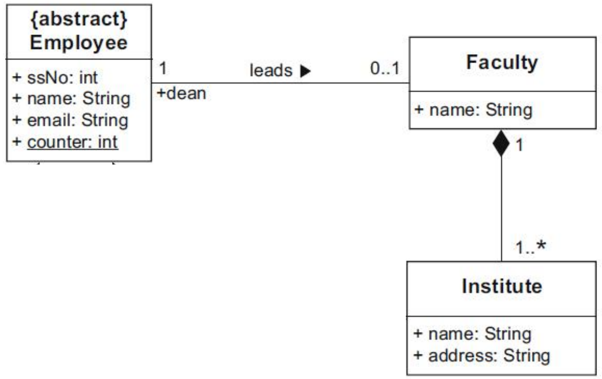
An Example

- A university consists of multiple faculties which are composed of various institutes. Each faculty and each institute has a name. An address is known for each institute.



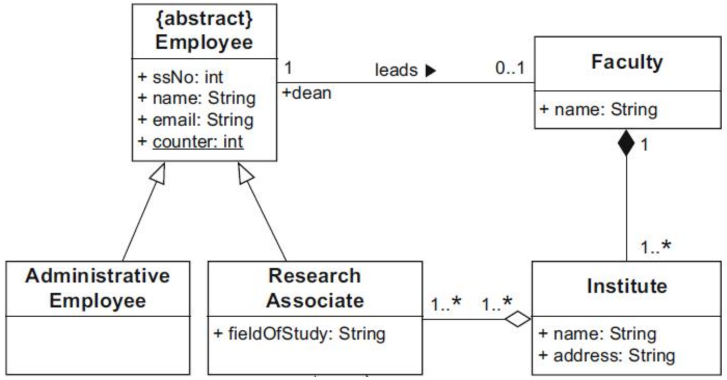
An Example

- Each faculty is led by a dean, who is an employee of the university.



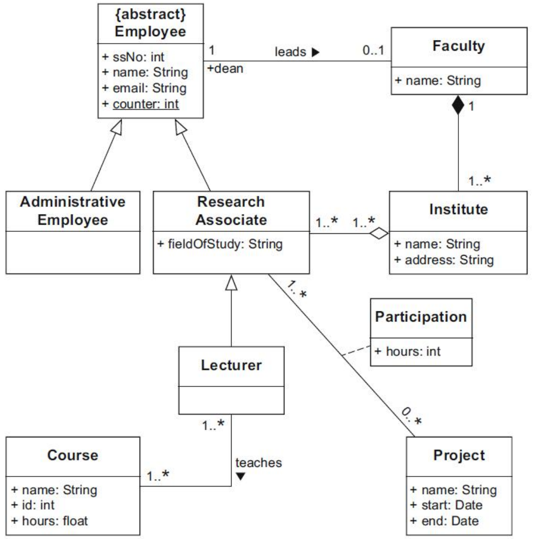
An Example

- The total number of employees is known. Employees have a social security number, a name, and an e-mail address. There is a distinction between research and administrative personnel.
- Research associates are assigned to at least one institute.

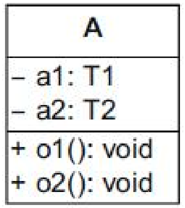
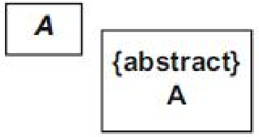
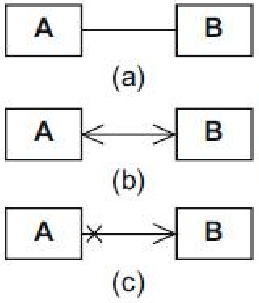
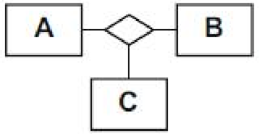
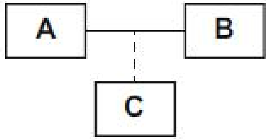
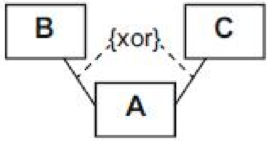
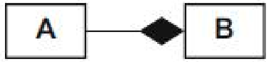
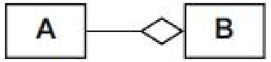
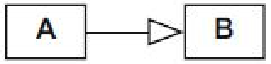
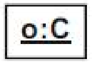
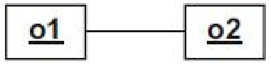


An Example

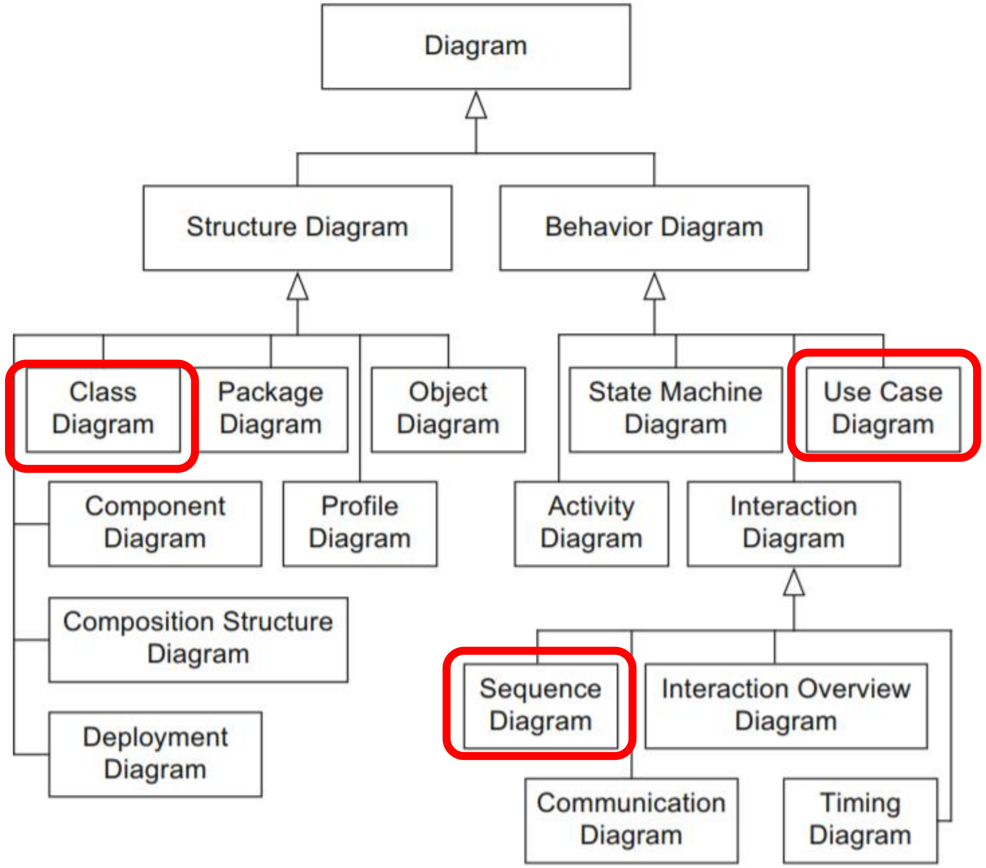
- The field of study of each research associate is known. Furthermore, research associates can be involved in projects for a certain number of hours, and the name, starting date, and end date of the projects are known. Some research associates teach courses. They are called lecturers.
- Courses have a unique number (ID), a name, and a weekly duration in hours.



Summary: Class Diagram

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 (a), navigable in both directions (b), not navigable in one direction (c)
N-ary association		Relationship between *N* (in this case 3) classes
Association class		More detailed description of an association
xor relationship		An object of A is in a relationship with an object of B or with an object of C but not with both
Strong aggregation = composition		Existence-dependent parts-whole relationship (A is part of B; if B is deleted, related instances of A are also deleted)
Shared aggregation		Parts-whole relationship (A is part of B; if B is deleted, related instances of A need not be deleted)
Generalization		Inheritance relationship (A inherits from B)
Object		Instance of a class
Link		Relationship between objects

UML Diagrams

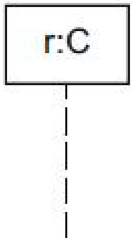


Sequence Diagram

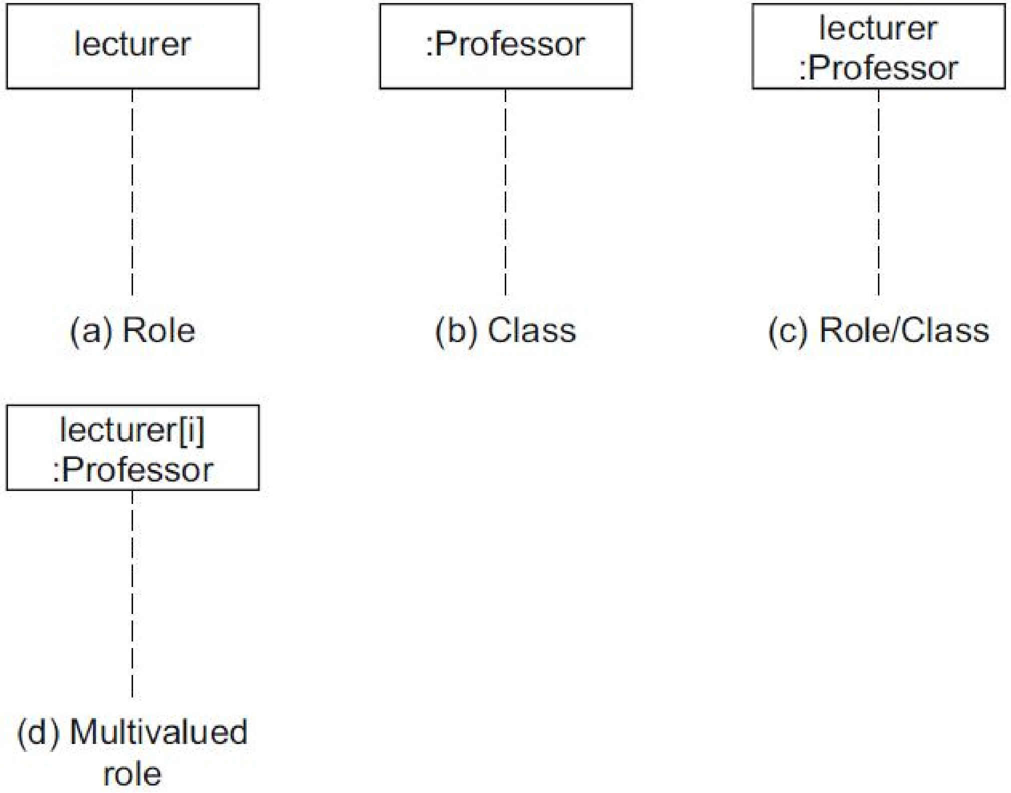
- Message among interaction partners
- Can be constructed with different granularity at different design stages
  - Interaction between the system and its environment
  - Interaction among system parts
  - Interaction between design objects

Interaction Partners

- Lifeline
  - r: role
  - C: class



- Use roles instead of objects
  - Each object can play different roles



(a) Role

(b) Class

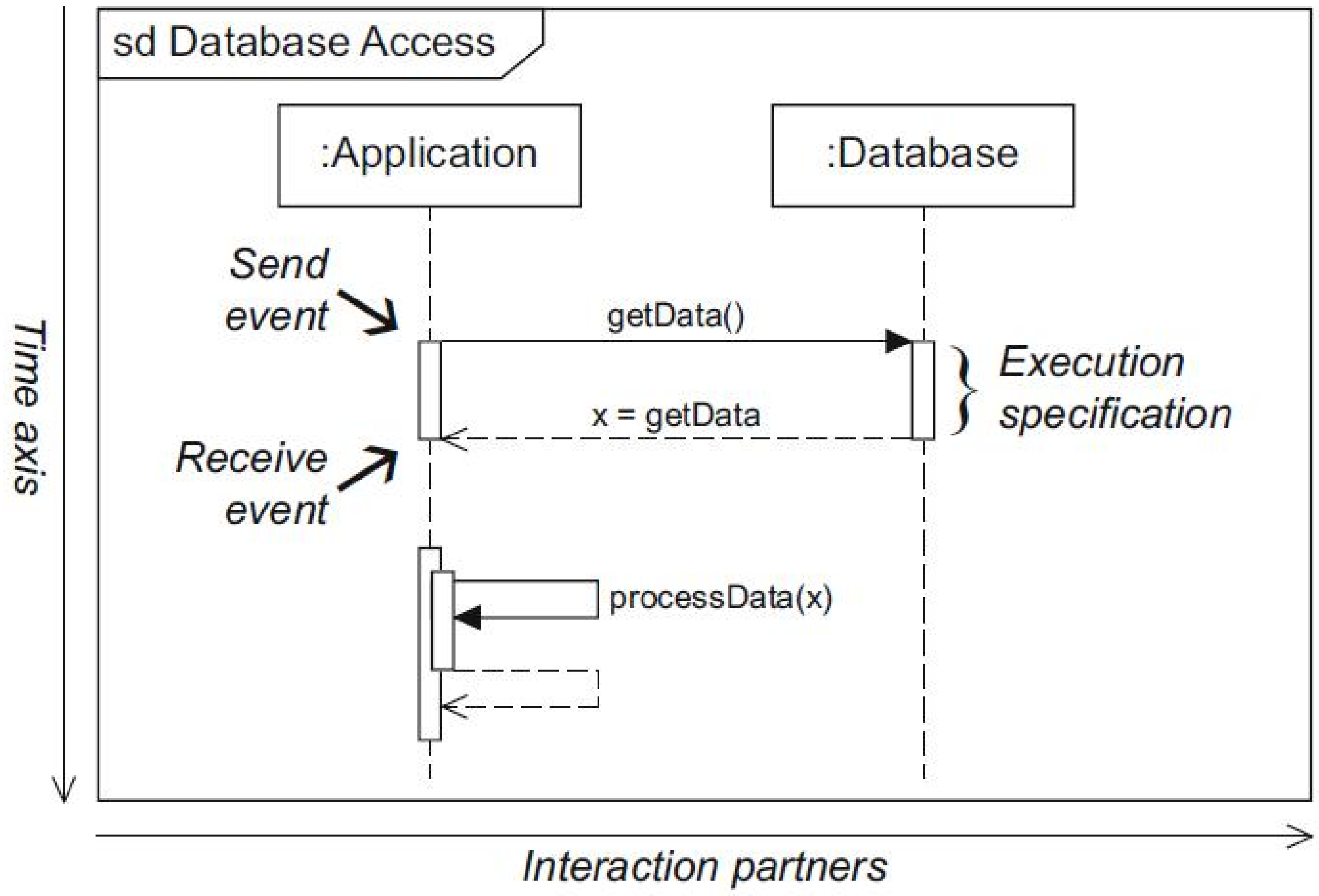
(c) Role/Class

(d) Multivalued role

Message Exchange

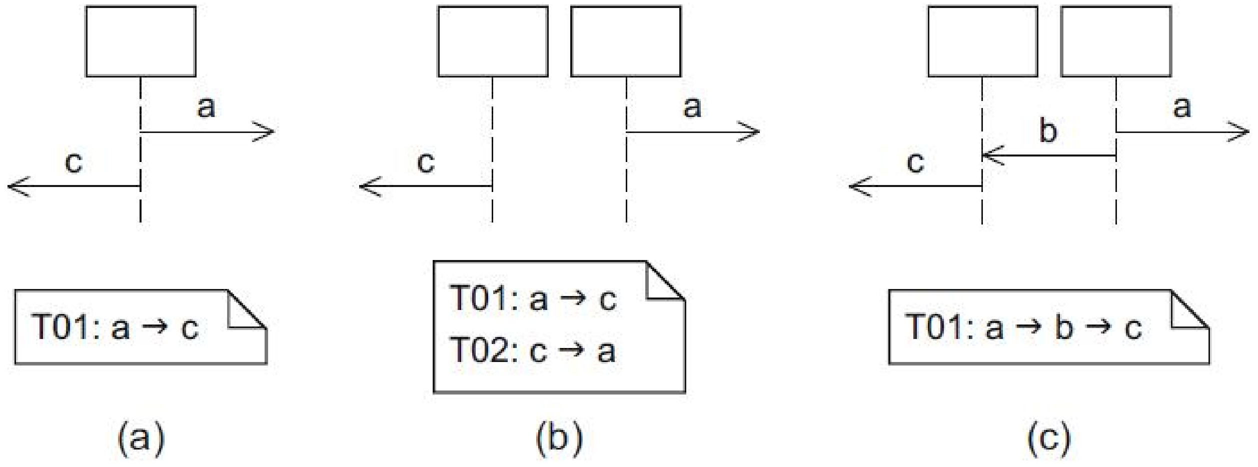
- Two dimensions
  - Time
  - Interaction partners

- Execution specification
  - Self message



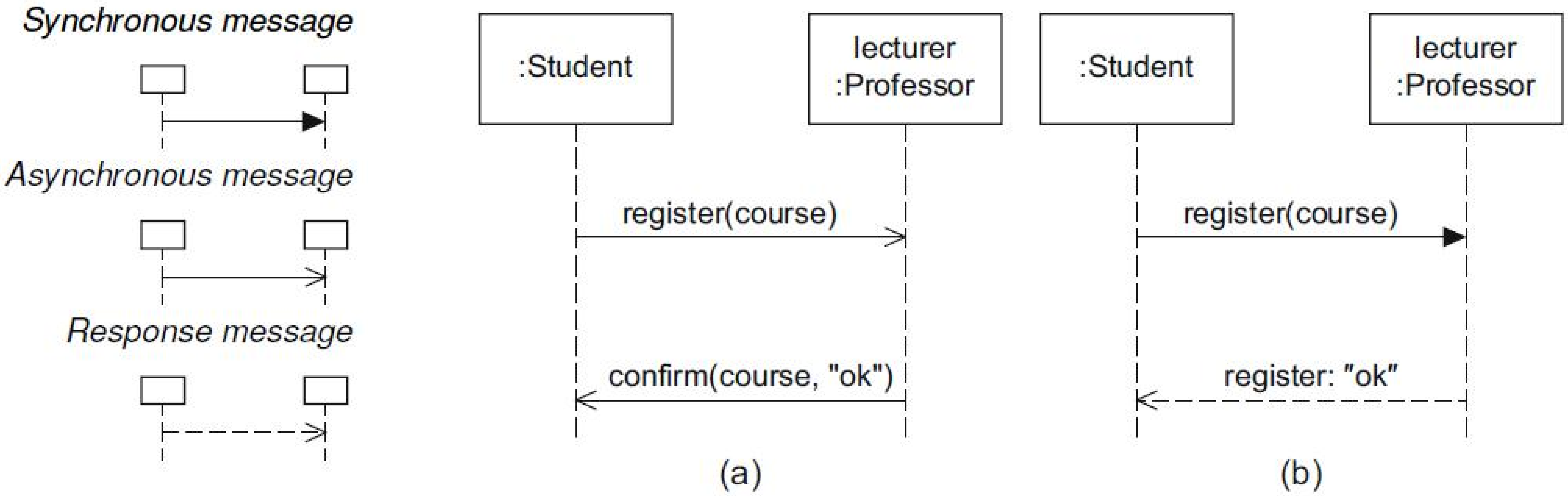
Message Exchange: Order

- Message order is chronical if messages on the same lifeline
  - It’s a transitive relationship



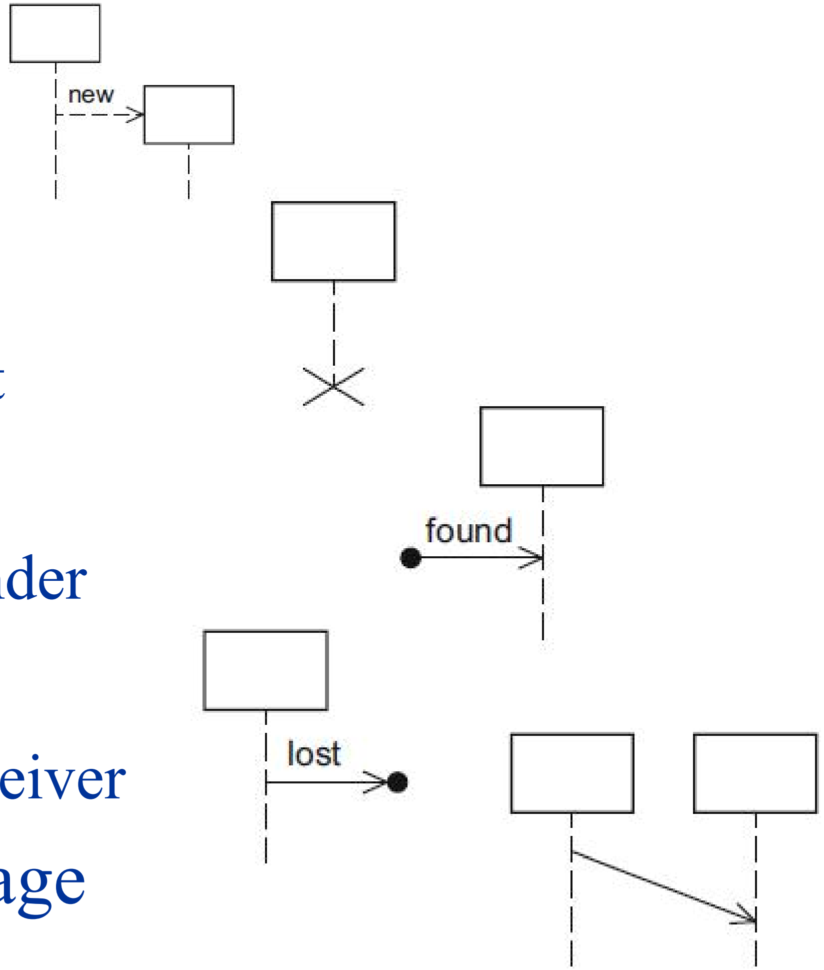
Message Exchange: Types

- (a) Register a course via email
- (b) Register a course in person



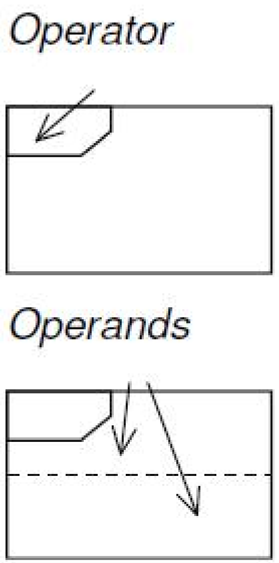
Special Messages

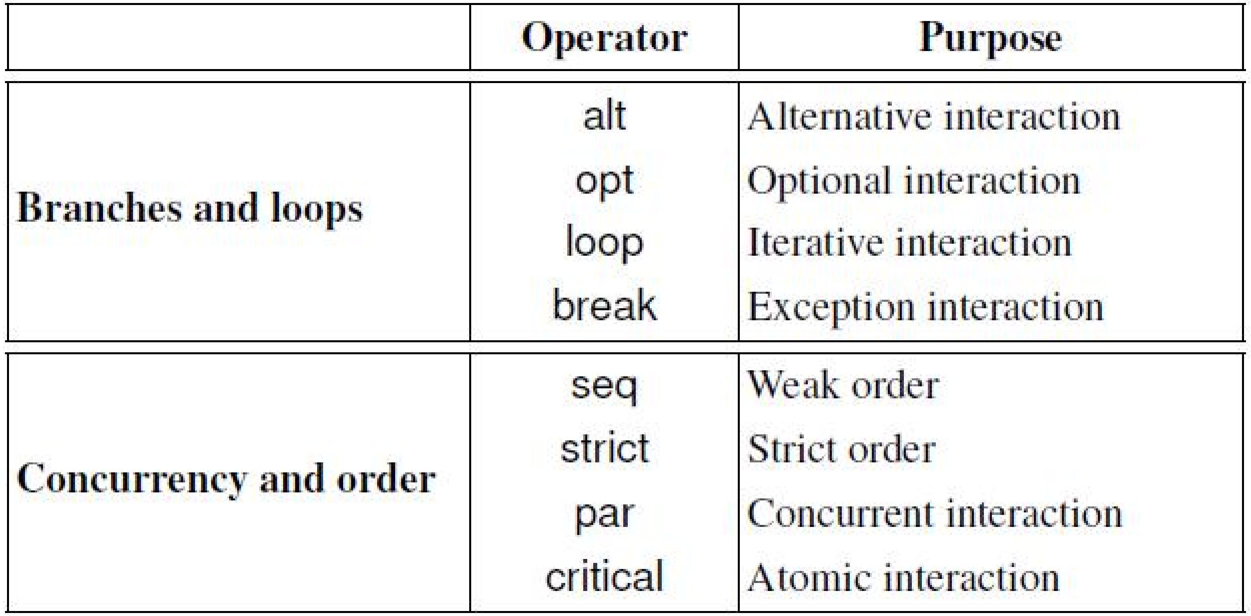
- Create message
  - Creating new object
- Destruction event
  - Destruction of an object
- Found message
  - Unknown/irrelevant sender
- Lost message
  - Unknown/irrelevant receiver
- Time-consuming message



Combined Fragments

- Each operand has a guard

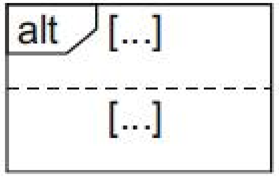




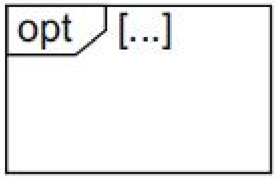
	Operator	Purpose
Branches and loops	alt	Alternative interaction
	opt	Optional interaction
	loop	Iterative interaction
	break	Exception interaction
Concurrency and order	seq	Weak order
	strict	Strict order
	par	Concurrent interaction
	critical	Atomic interaction

Branches and loops

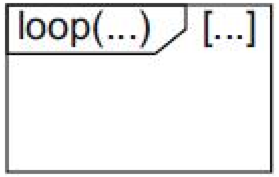
- Alternative interactions
  - If-else



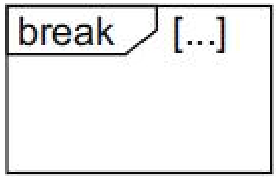
- Optional interactions
  - “if” without an “else”

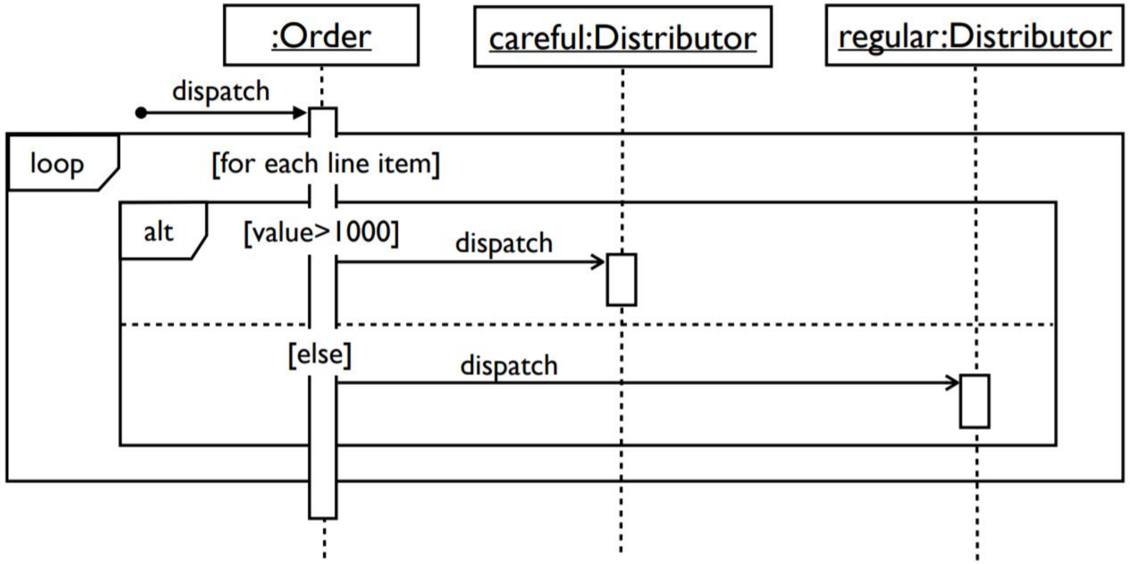


- Iterative interactions
  - For loop



- Exception interactions
  - Omit the remaining



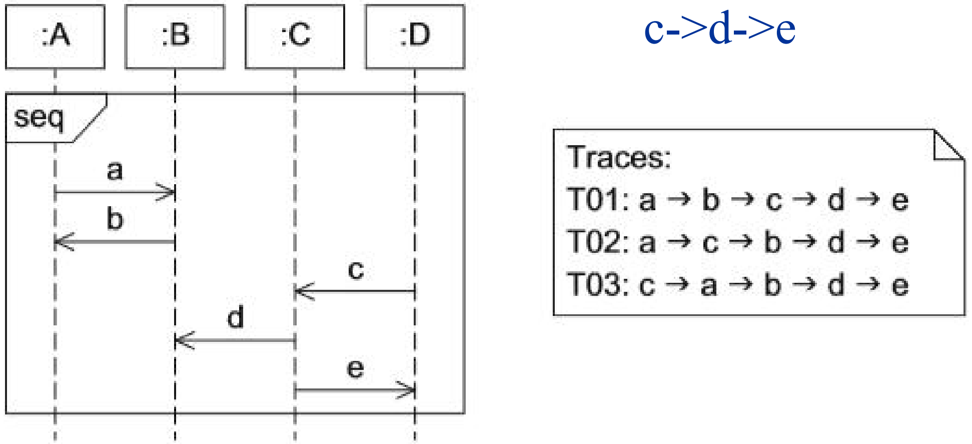


Concurrency and Order

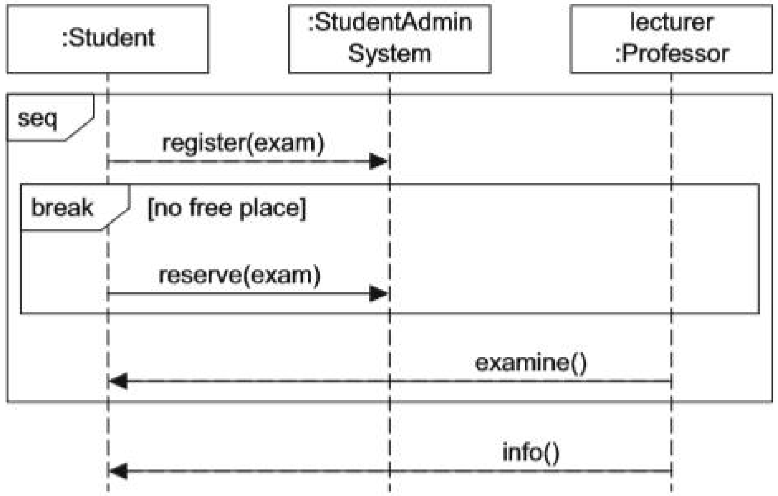
- Seq fragment

  - Weak order
  - The ordering of events within each of the operands is maintained in the result.
  - Events on different lifelines from different operands may come in any order.
  - Events on the same life line from different operands are ordered such that an event of the first operand comes before that of the second operand.

a->b->d
c->d->e

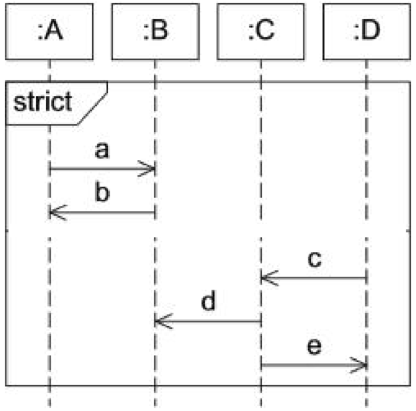


Traces:
T01: a → b → c → d → e
T02: a → c → b → d → e
T03: c → a → b → d → e



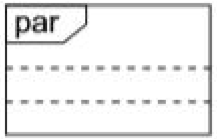
Concurrency and Order (cont.)

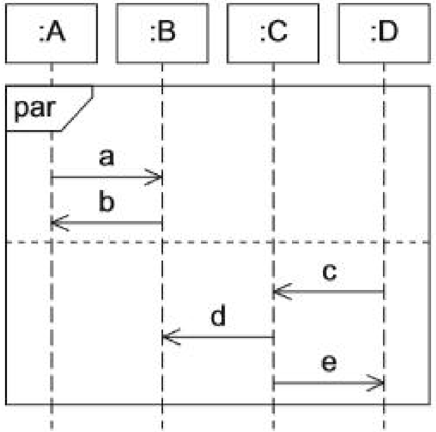
- Strict fragment
  - Strong & strict order



Traces:
T01: a → b → c → d → e

- Par fragment
  - Order within the operands are respected
  - The order of operands does not matter

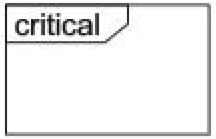


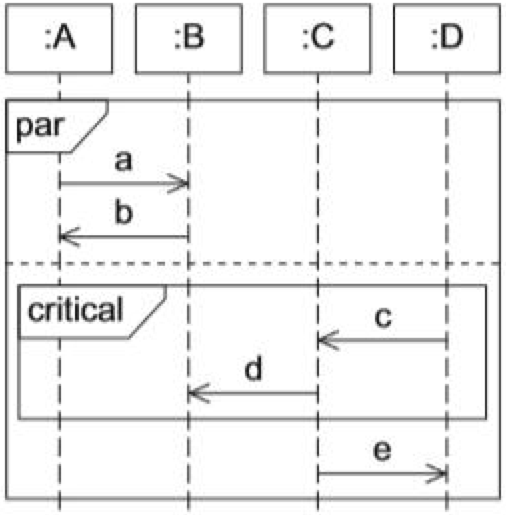


Traces:
T01: a → b → c → d → e
T02: a → c → b → d → e
T03: a → c → d → b → e
T04: a → c → d → e → b
T05: c → a → b → d → e
T06: c → a → d → b → e
T07: c → a → d → e → b
T08: c → d → a → b → e
T09: c → d → a → e → b
T10: c → d → e → a → b

Concurrency and Order (cont.)

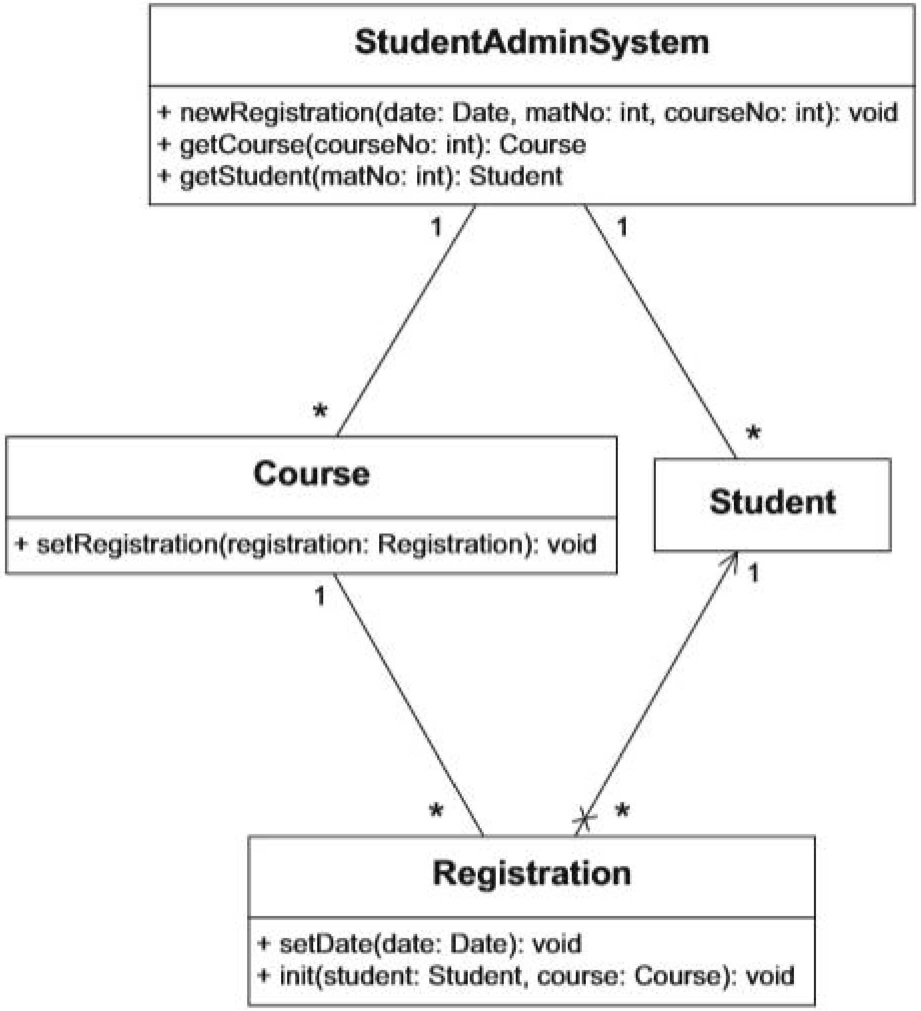
- Critical fragment
  - Atomic interaction
  - No other messages can happen during the execution

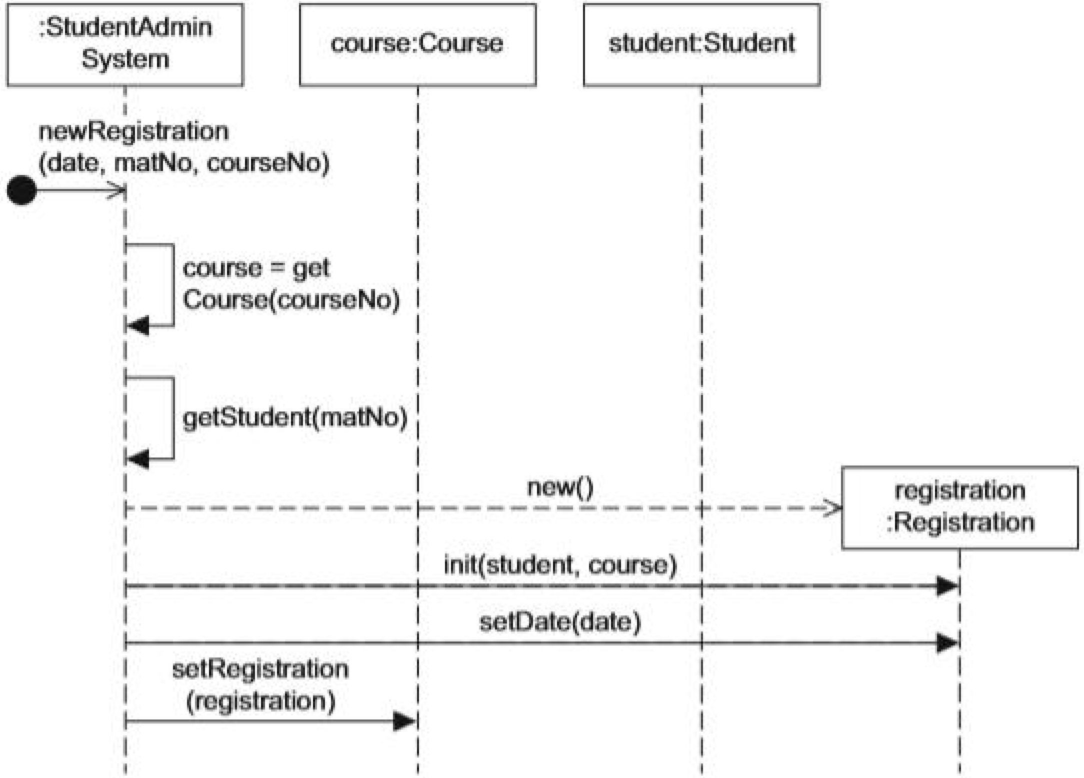


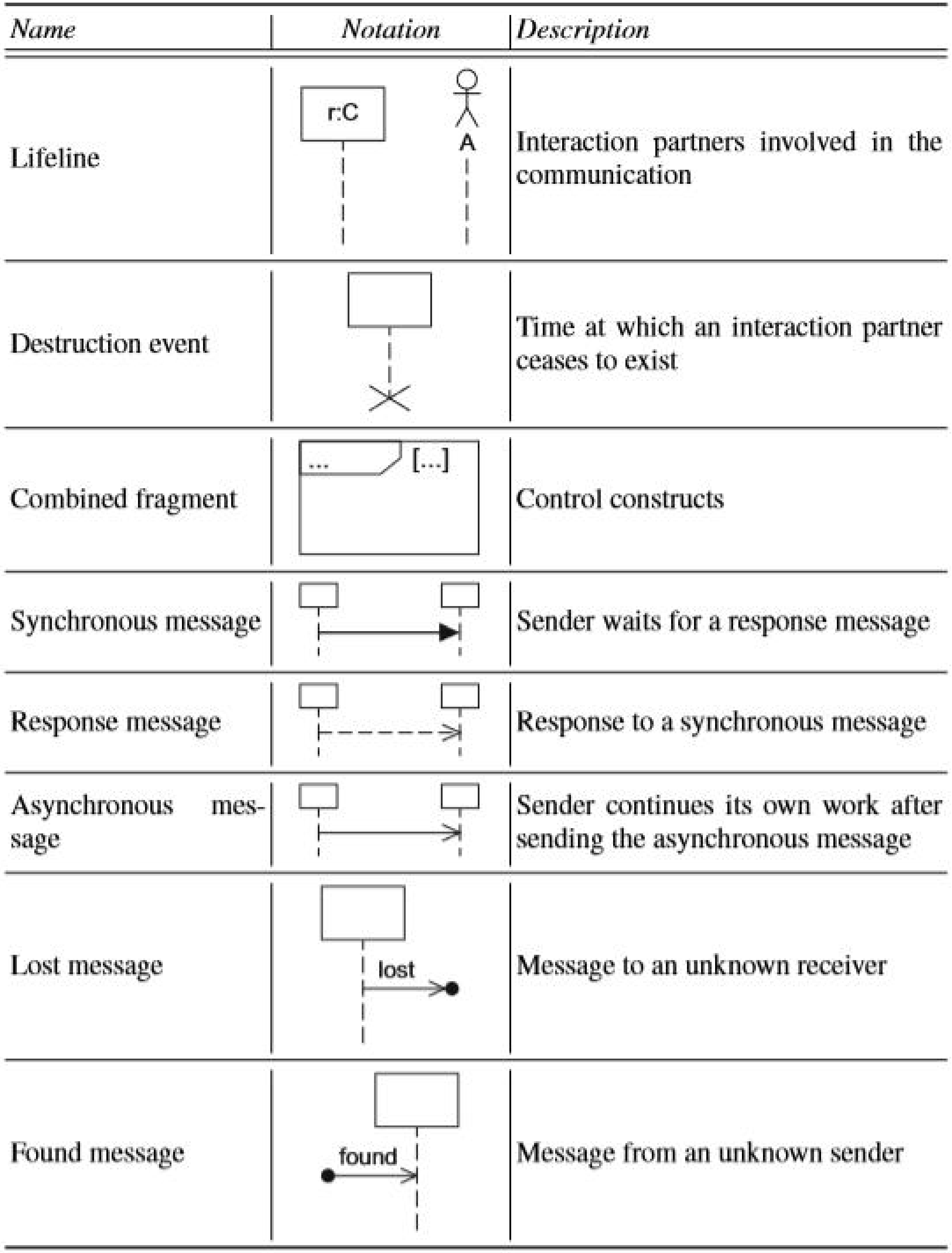


Traces:
T01: a → b → c → d → e
T02: a → c → d → b → e
T03: a → c → d → e → b
T04: c → d → a → b → e
T05: c → d → a → e → b
T06: c → d → e → a → b

Example


StudentAdminSystem
+ newRegistration(date: Date, matNo: int, courseNo: int): void
+ getCourse(courseNo: int): Course
+ getStudent(matNo: int): Student
1
1
*
*
Course
+ setRegistration(registration: Registration): void
Student
1
1
*
*
Registration
+ setDate(date: Date): void
+ init(student: Student, course: Course): void

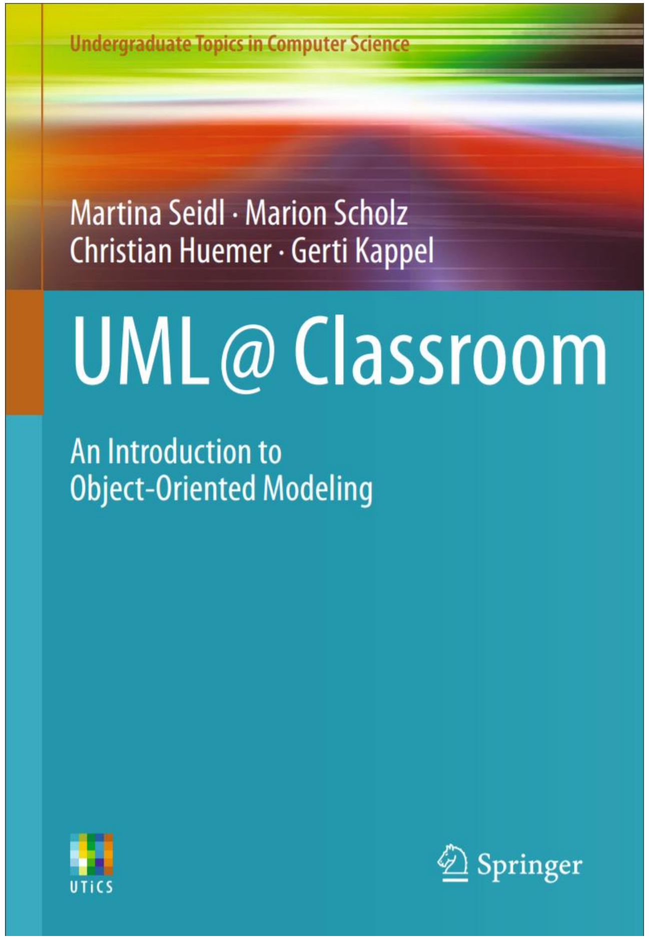

:StudentAdmin System
course:Course
student:Student
newRegistration (date, matNo, courseNo)
course = get Course(courseNo)
getStudent(matNo)
new()
registration :Registration
init(student, course)
setDate(date)
setRegistration (registration)



Name	*Notation*	*Description*
Lifeline	r:C A	Interaction partners involved in the communication
Destruction event		Time at which an interaction partner ceases to exist
Combined fragment	... [...]	Control constructs
Synchronous message		Sender waits for a response message
Response message		Response to a synchronous message
Asynchronous message		Sender continues its own work after sending the asynchronous message
Lost message	lost	Message to an unknown receiver
Found message	found	Message from an unknown sender

Reference for UML

- Freely available online
- Search from our library website



Undergraduate Topics in Computer Science

Martina Seidl · Marion Scholz
Christian Huemer · Gerti Kappel

UML @ Classroom

An Introduction to
Object-Oriented Modeling

UTiCS

Springer

UML Drawing Tools

- Microsoft Visio can draw basic UML diagrams
  - Available from the library
- Visual Paradigm (Community edition)
  - https://www.visual-paradigm.com/download/community.jsp
- IBM Rational Rose
  - Cracked version online (not recommended)