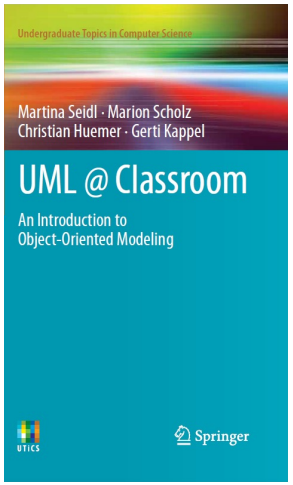
Business Informatics Group

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

Introduction

https://youtu.be/pCK6prSq8aw

Introduction

- Modeling inter-object behavior which is the 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. For example: an oral exam.
  - Message exchange between humans and a software system. For example: between a lecturer and the student administration system when the lecturer publishes exam results.
  - Communication protocols. For example: HTTP.
  - Sequence of method calls in a program. For example: fire alarm and the resulting communication processes.

Interaction Diagrams

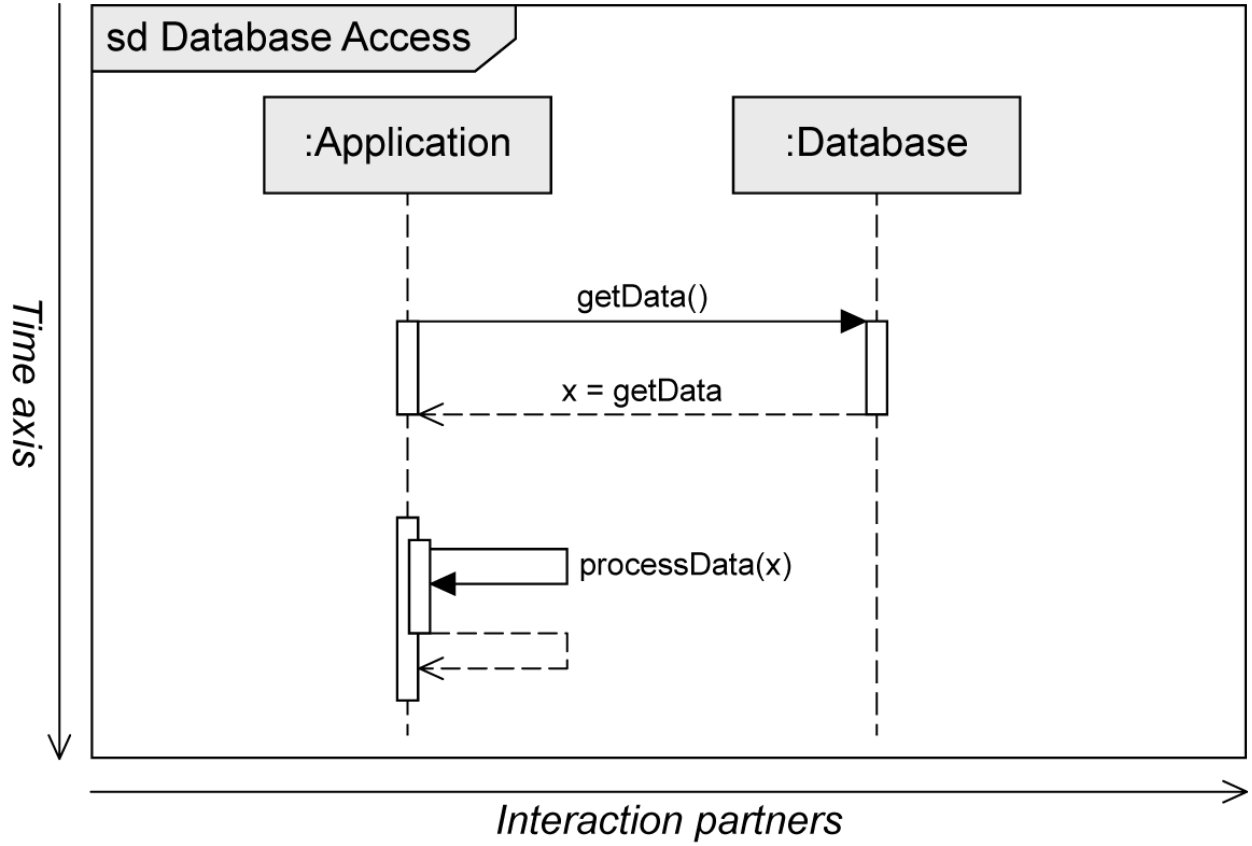
- An interaction describes the **interplay** between *multiple interaction partners* and comprises a sequence of **messages**.
- The sending or receipt of a message can be triggered by the occurrence of certain events.
  - For example: the receipt of another message, and can take place at specified times, for example, at 05:00.
- **Predefined constraints** specify any necessary *preconditions* that must be met for successful interactions.
  - For example, the lecturer must be logged into the system before entering the students' grades.
- In UML, you use interaction diagrams to:
  - Used to specify interactions
  - Modeling concrete scenarios meaning that the message exchange takes place within a specific context to fulfill a specific task.
  - Describing communication sequences at different levels of detail
- Interactions usually only describe a specific part of a situation. There are often other valid execution paths that the interaction diagram does not cover.

Interaction Diagrams

- 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)

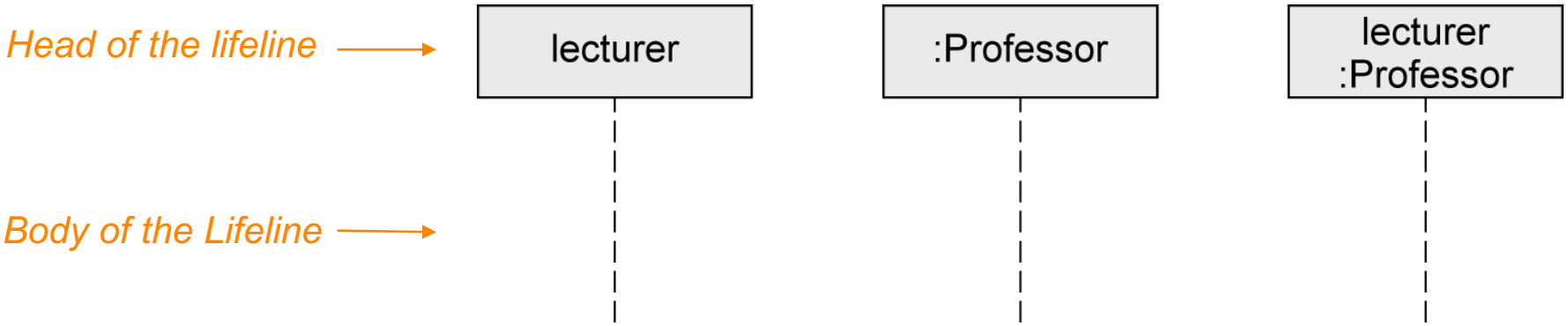
Sequence Diagram

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



Interaction Partners

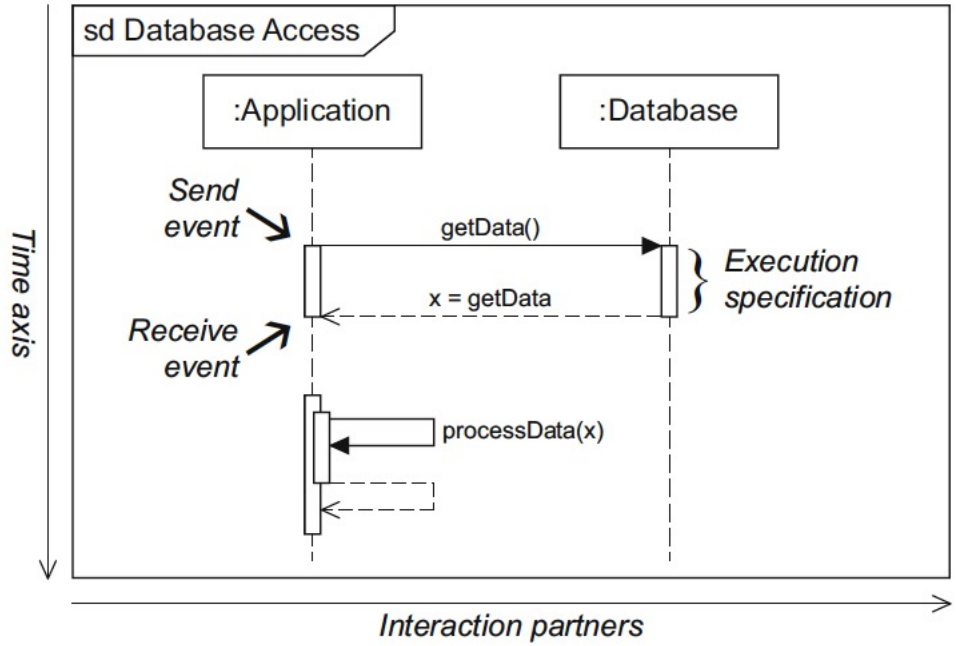
- Interaction partners are depicted as **lifelines**
- Head of the lifeline
  - Rectangle that contains the expression **roleName:Class**
  - Roles are a more general concept than objects
  - Object can take on different roles over its lifetime
- Body of the lifeline
  - Vertical, usually dashed line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

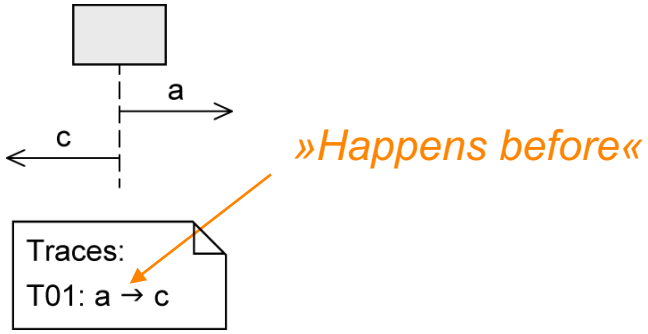
- Execution specification
  - Continuous bar
  - Used to visualize when an interaction partner executes some behavior



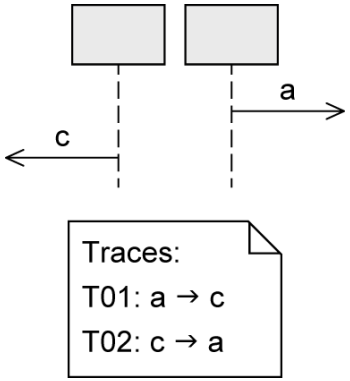
Exchanging Messages (2/2)

Order of messages:

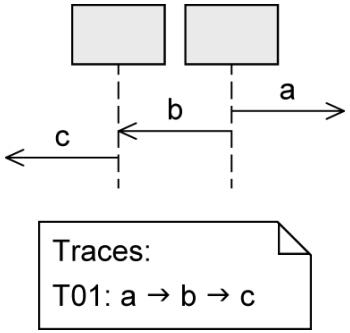
… on one lifeline



… on different lifelines

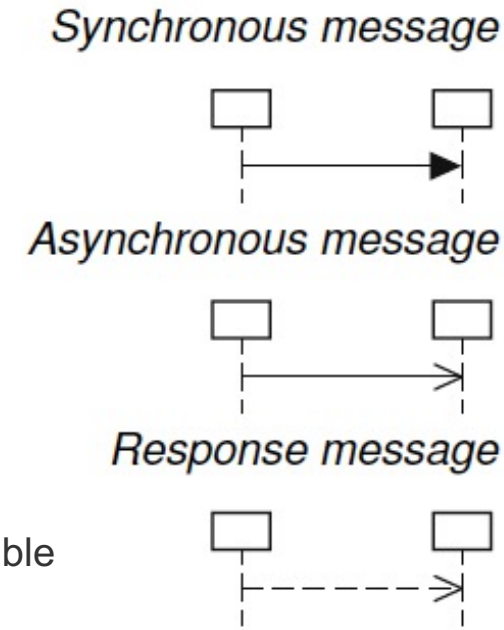


… on different lifelines which exchange messages

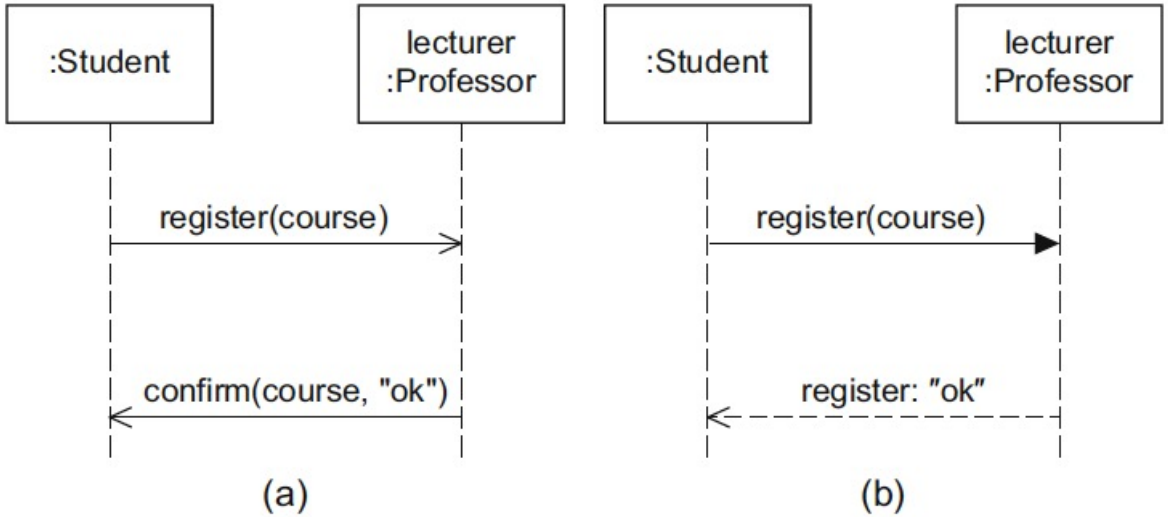


Messages (1/3)

- 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 (1/3)

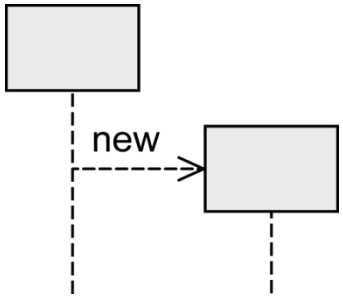


In both cases, a student is communicating with a professor in order to register for a course.

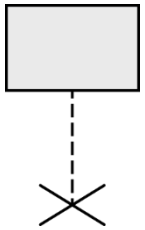
- In case (a), the registration is via e-mail, that is, **asynchronous**. The student does not explicitly wait for the receipt of the confirmation message.
- In case (b), the student registers with the professor personally and the communication is therefore **synchronous**. The student waits until receiving a response message.

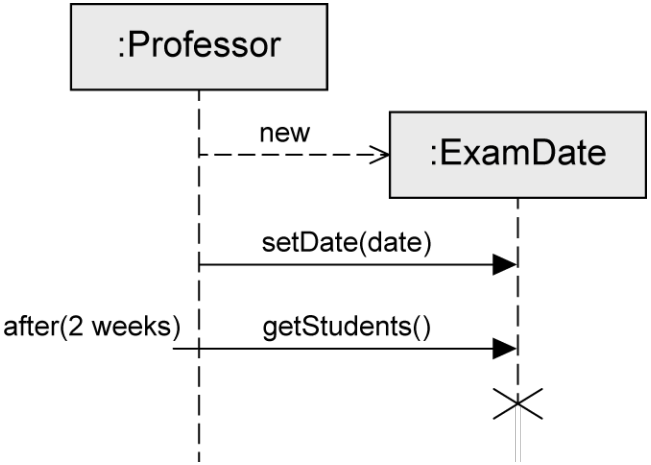
Messages (2/3)

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



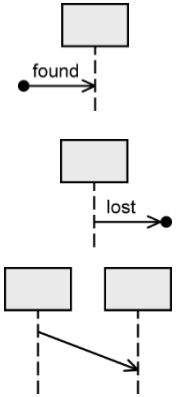
- Object Destruction
  - Object is deleted
  - Large cross (×) at the end of the lifeline

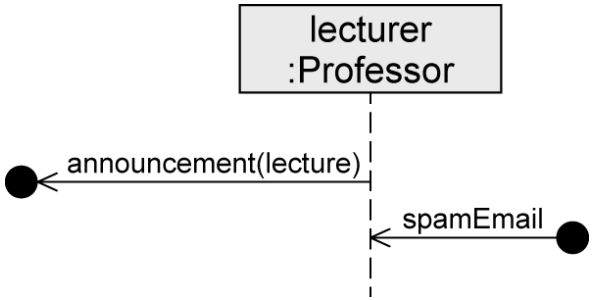


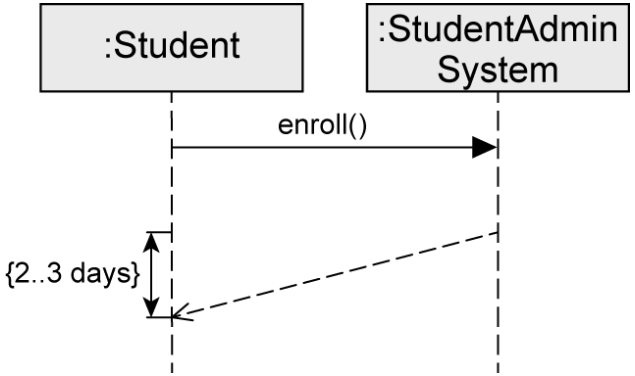


Messages (3/3)

- 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

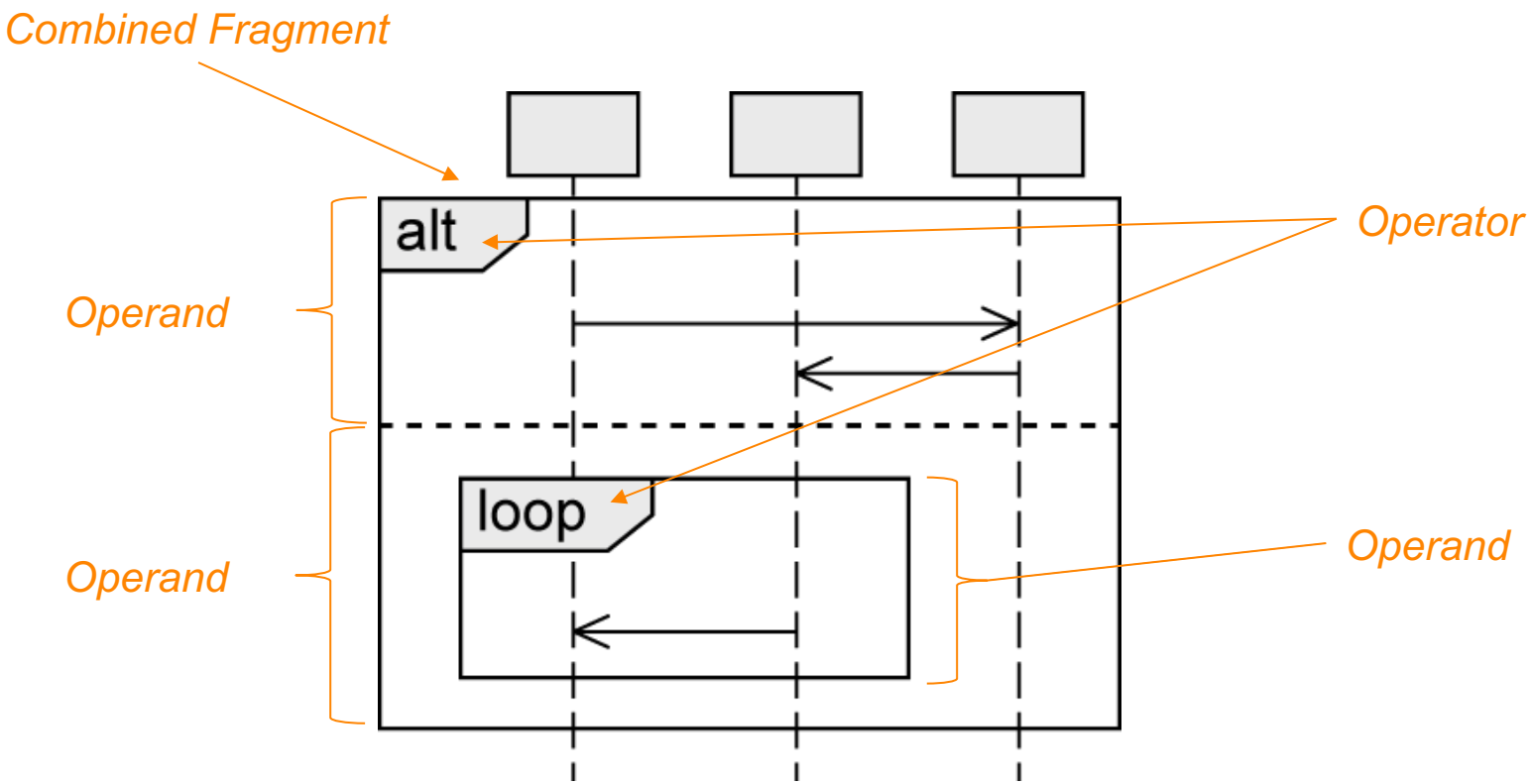






Combined Fragments

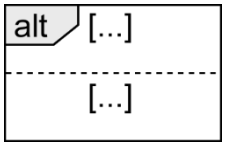
- Model various control structures
- This enables you to describe a number of possible execution paths compactly and precisely
- 12 predefined types of operators



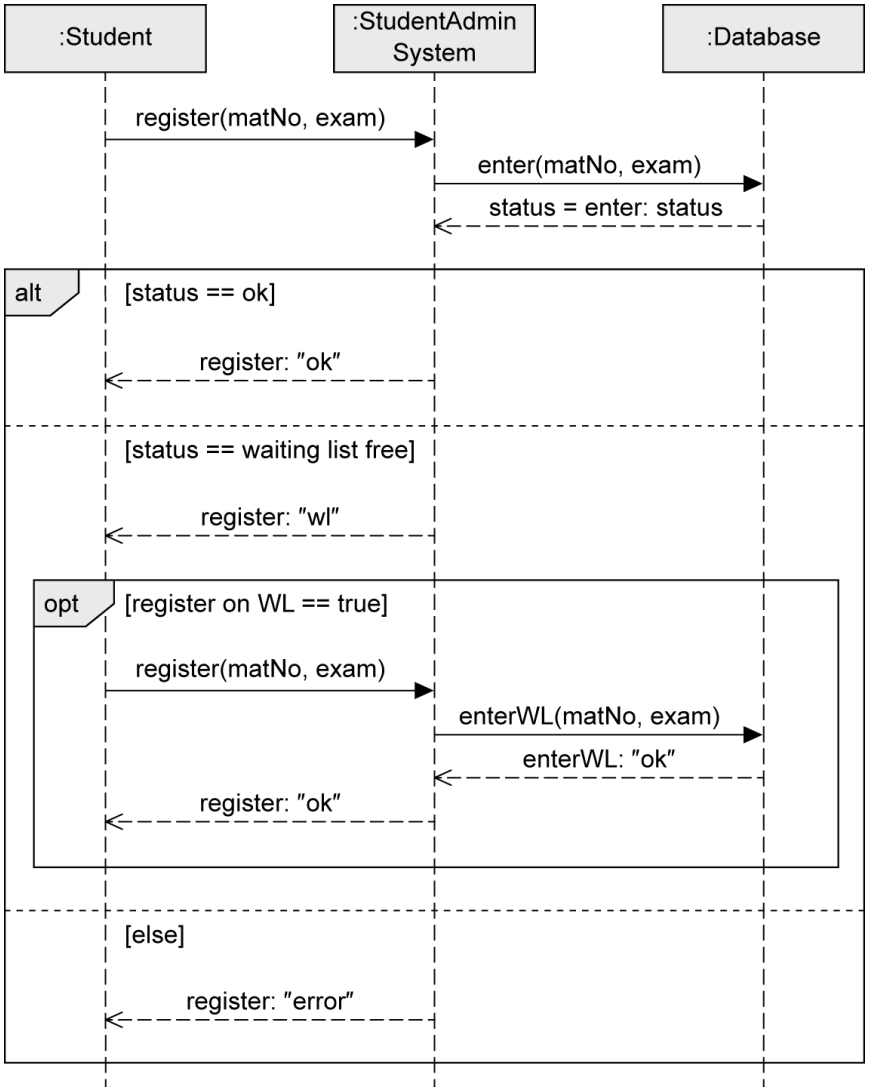
Types of Combined Fragments

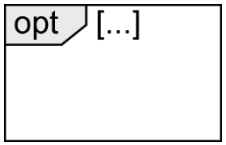
	Operator	Purpose
Branches and loops	`alt`	Alternative interaction
	`opt`	Optional interaction
	`loop`	Repeated interaction
	`break`	Exception interaction
Concurrency and order	`seq`	Weak order
	`strict`	Strict order
	`par`	Concurrent interaction
	`critical`	Atomic interaction
Filters and assertions	`ignore`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

`alt` Fragment



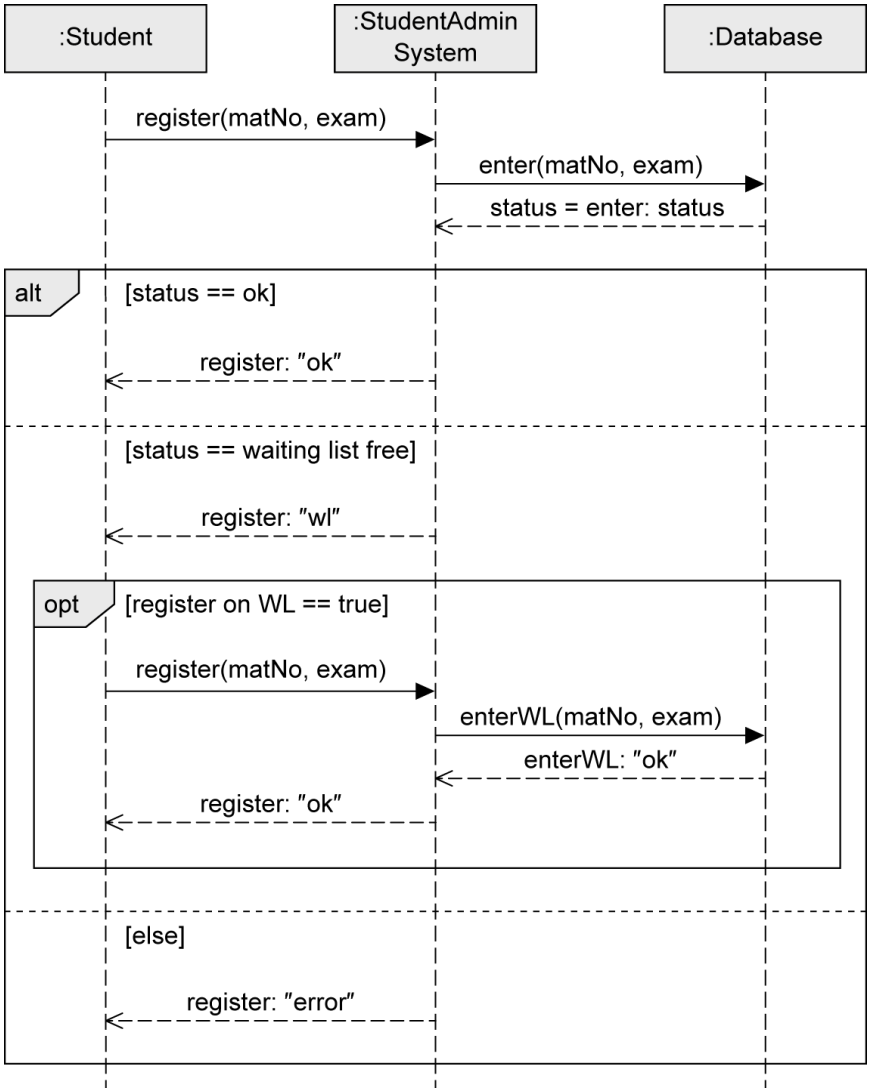
- To model alternative sequences
- Similar to switch statement in Java
- Guards are used to select the one path to be executed
- Guards
  - Modeled in square brackets
  - default: `true`
  - predefined: `[else]`
- Multiple operands
- Guards must be disjoint to avoid indeterministic behavior

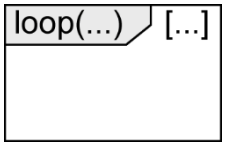




`opt` Fragment

- 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





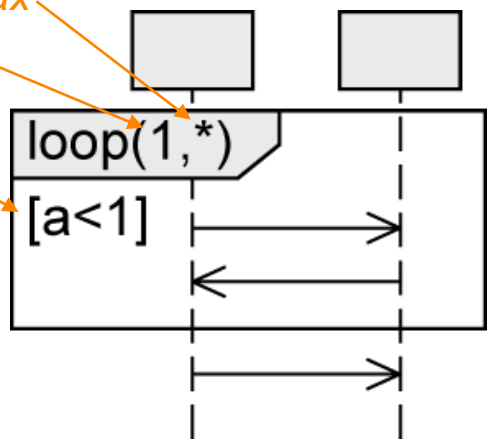
`loop` Fragment

- 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

Max

Min

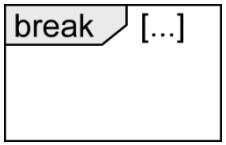
Guard



`loop` *is executed at least once, then as long as* ***`a<1`*** *is true*

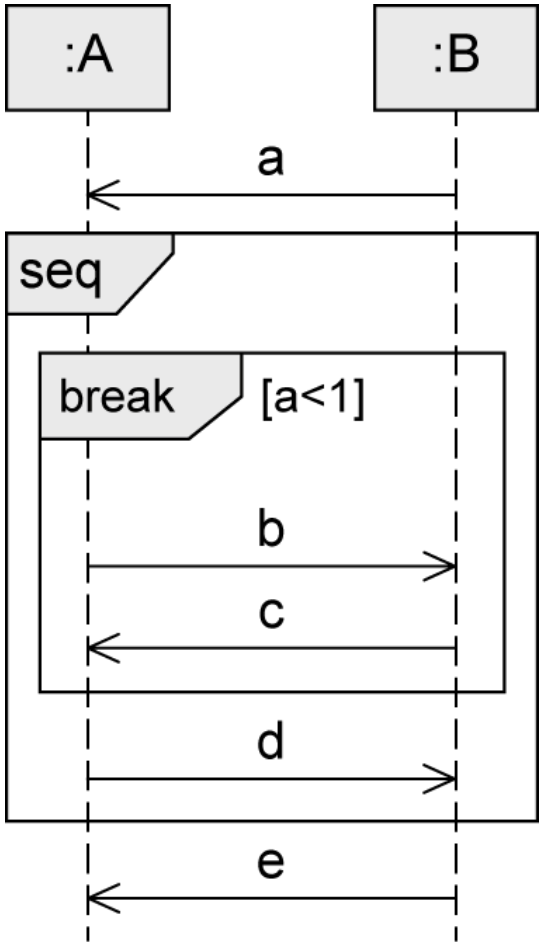
Notation alternatives:

`loop(3,8)` = `loop(3..8)`
`loop(8,8)` = `loop (8)`
`loop` = `loop (*)` = `loop(0,*)`

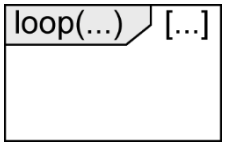


`break` Fragment

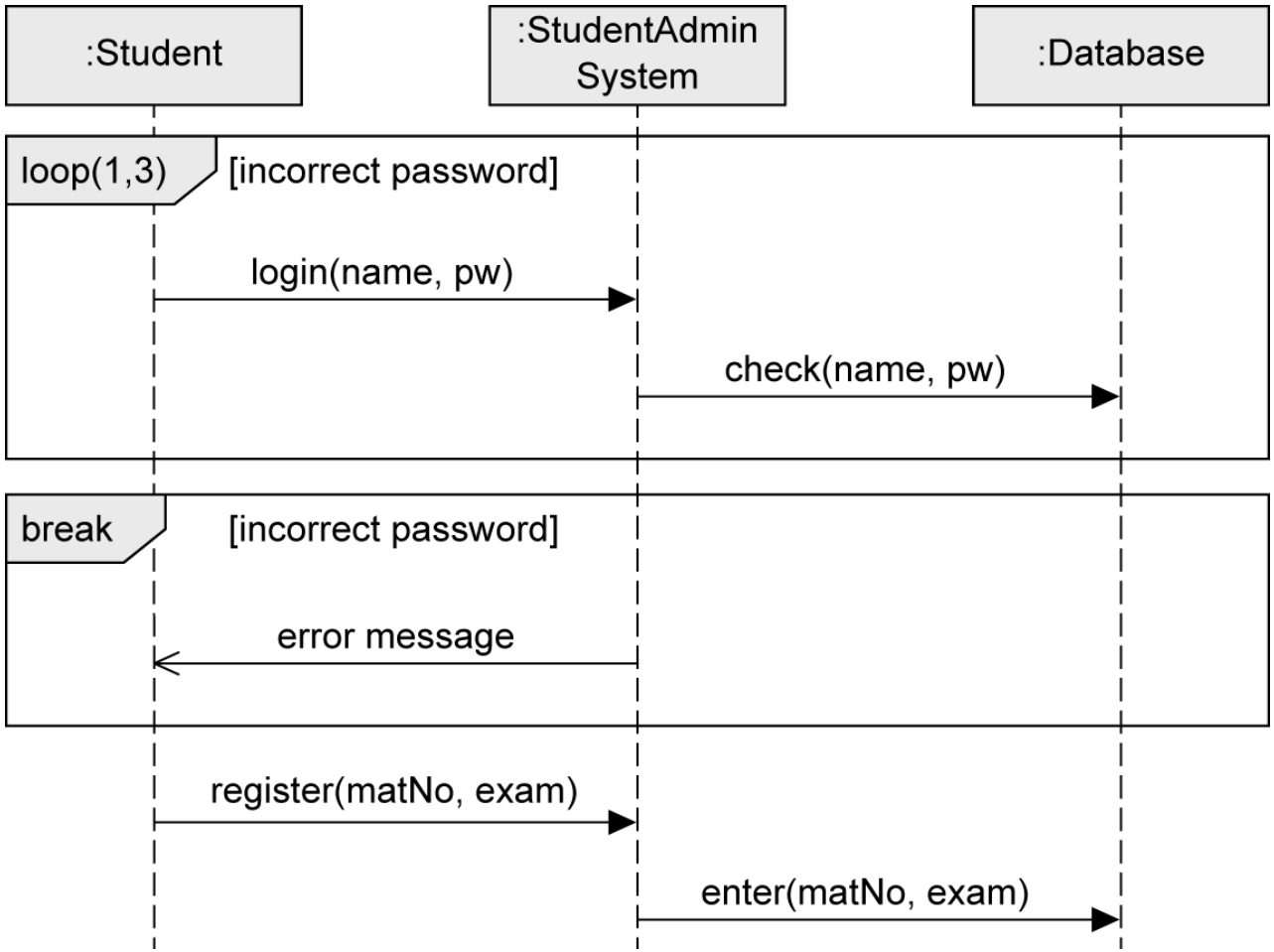
- Simple form of exception handling
- Exactly one operand with a guard
- If the guard is true:
  - Interactions within this operand are executed
  - Remaining operations of the surrounding fragment are omitted
  - Interaction continues in the next higher level fragment
  - Different behavior than **`opt`** fragment



Not executed if break is executed

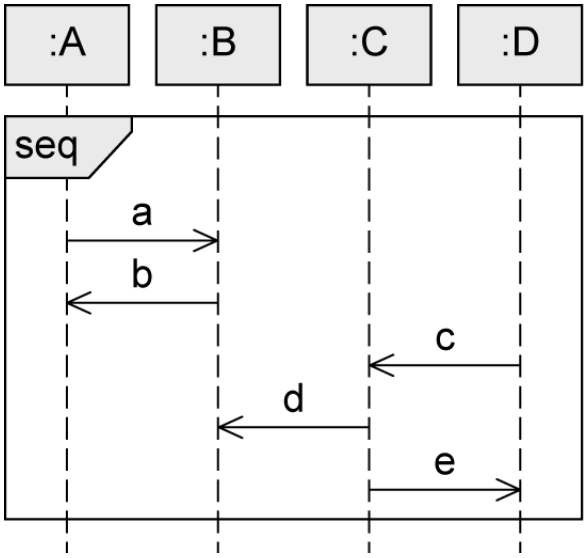


`loop` and `break` Fragment - Example



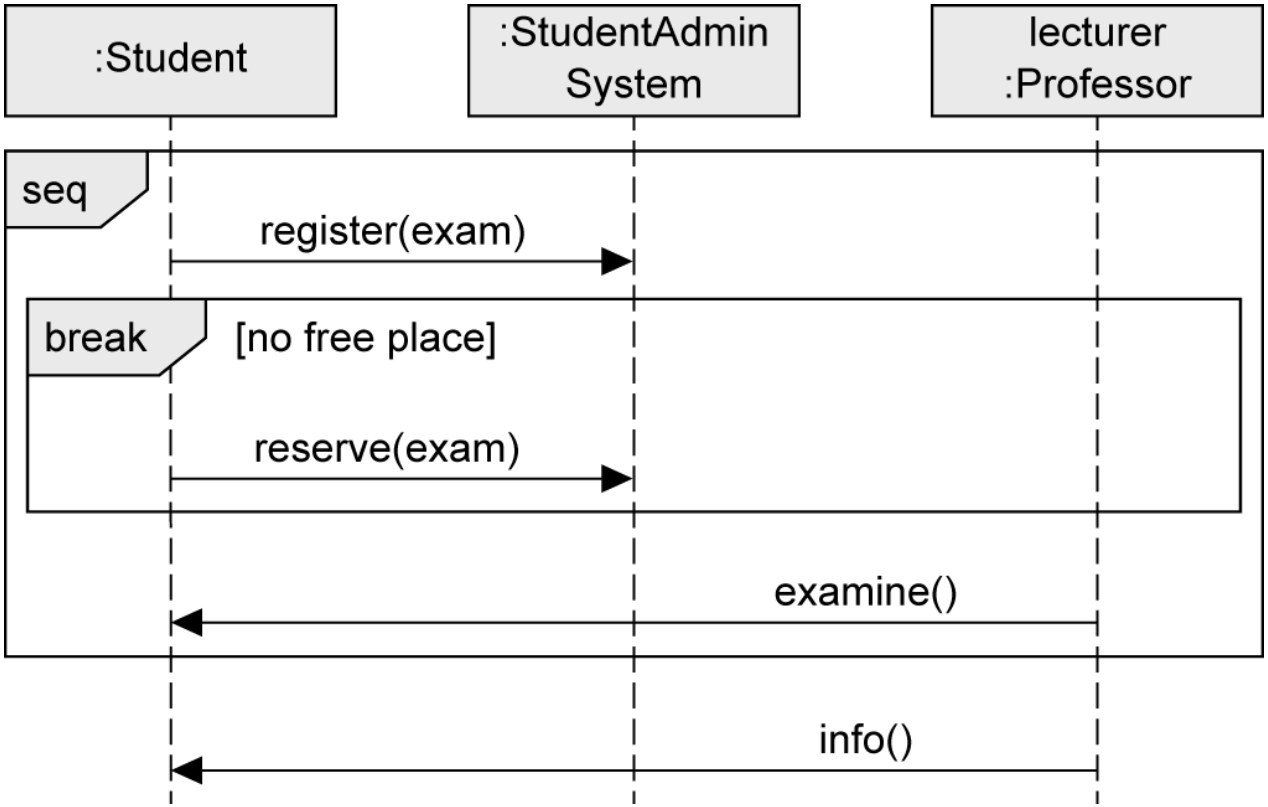
seq Fragment

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



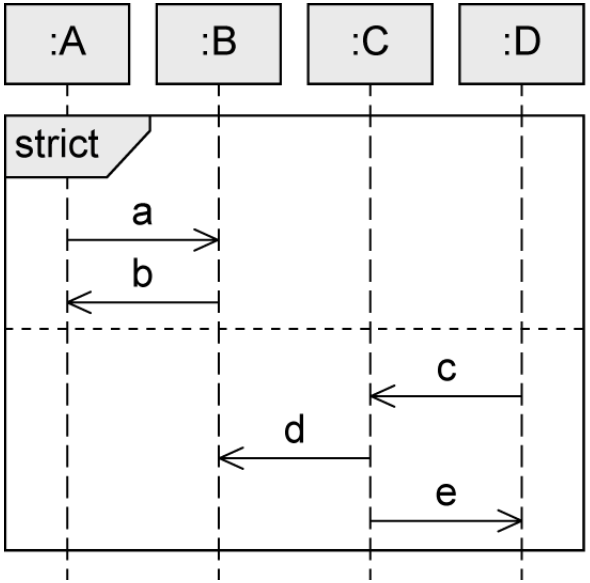
Traces:
T01: a → b → c → d → e
T02: a → c → b → d → e
T03: c → a → b → d → e

seq Fragment – Example



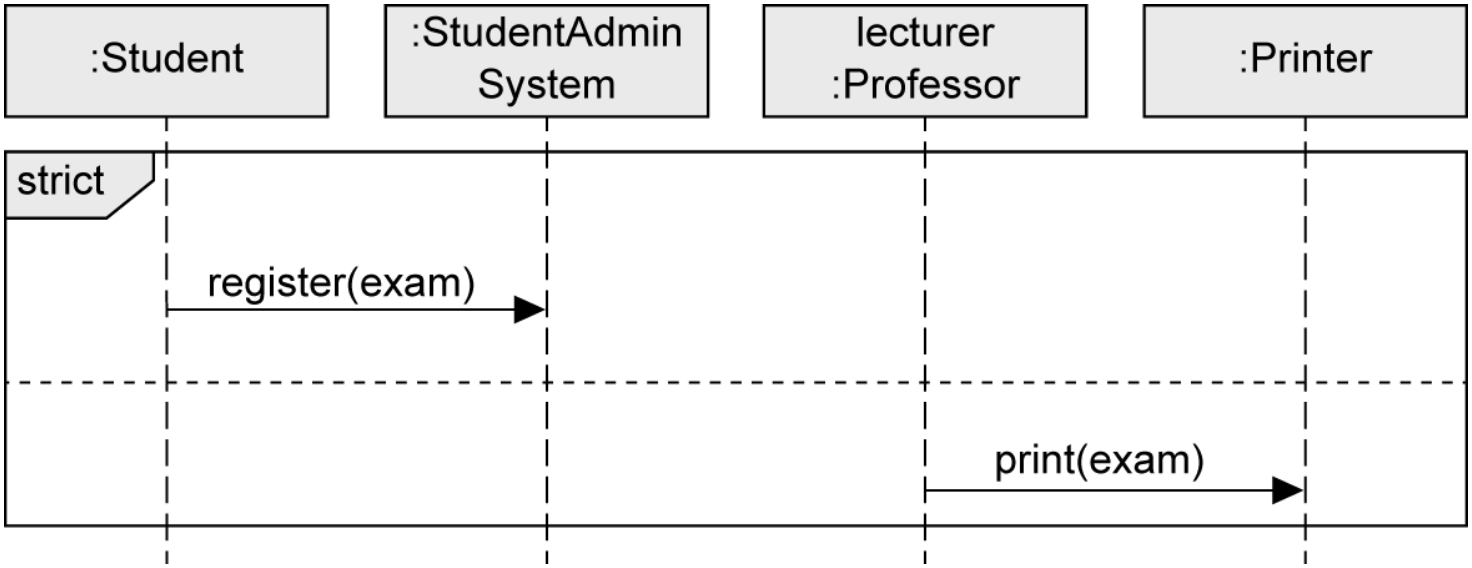
`strict` Fragment

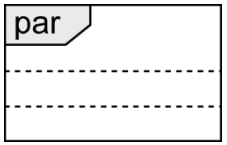
- Sequential interaction with order
- Order of event occurrences on different lifelines between different operands is significant
  - Messages in an operand that is higher up on the vertical axis are always exchanged before the messages in an operand that is lower down on the vertical axis



Traces:
T01: a → b → c → d → e

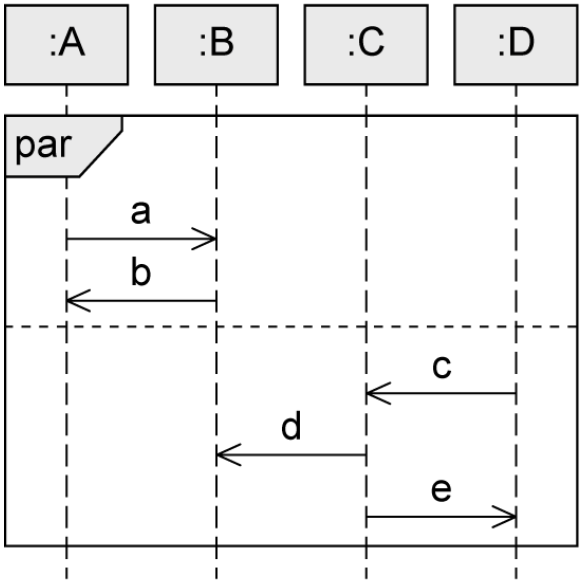
`strict` Fragment - Example





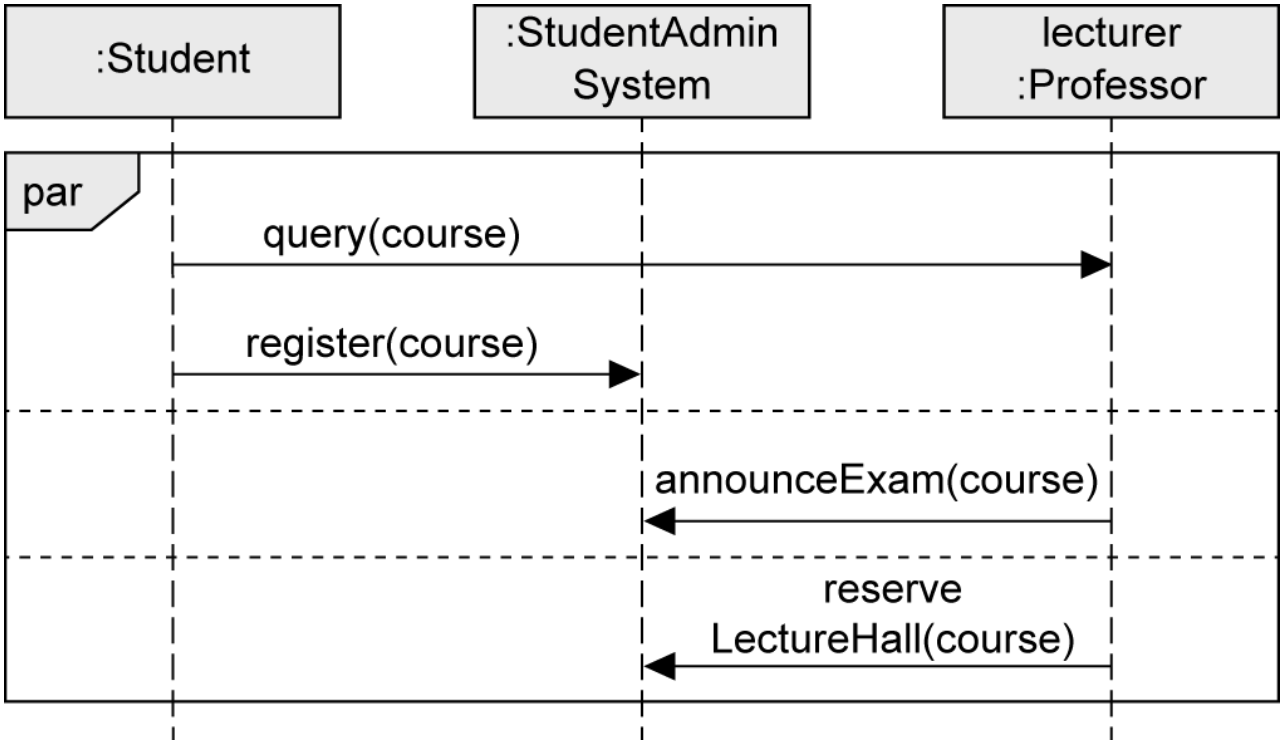
`par` Fragment

- To set aside chronological order between messages in different operands
- Execution paths of different operands can be interleaved
- Restrictions of each operand need to be respected
- Order of the different operands is irrelevant
- Concurrency, no true parallelism



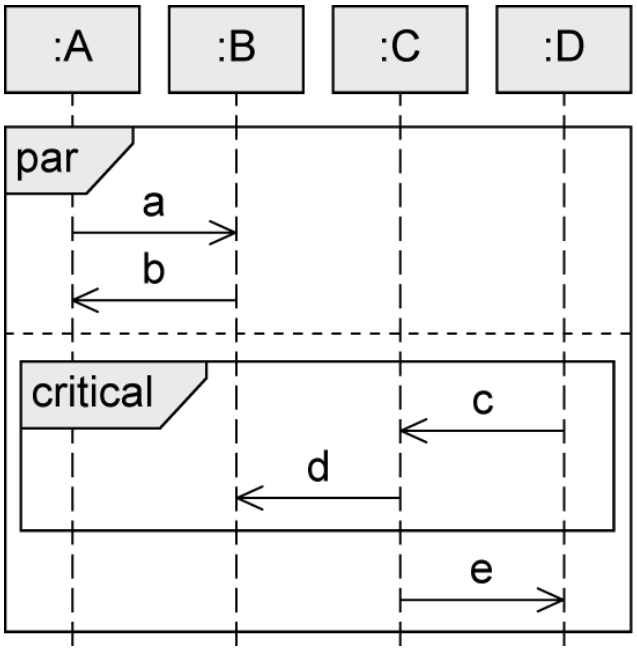
Traces:
T01: a → b → c → d → e
T02: a → c → b → d → e
T03: a → c → d → b → e
T04: a → c → d → e → b
T05: c → a → b → d → e
T06: c → a → d → b → e
T07: c → a → d → e → b
T08: c → d → a → b → e
T09: c → d → a → e → b
T10: c → d → e → a → b

`par` Fragment - Example



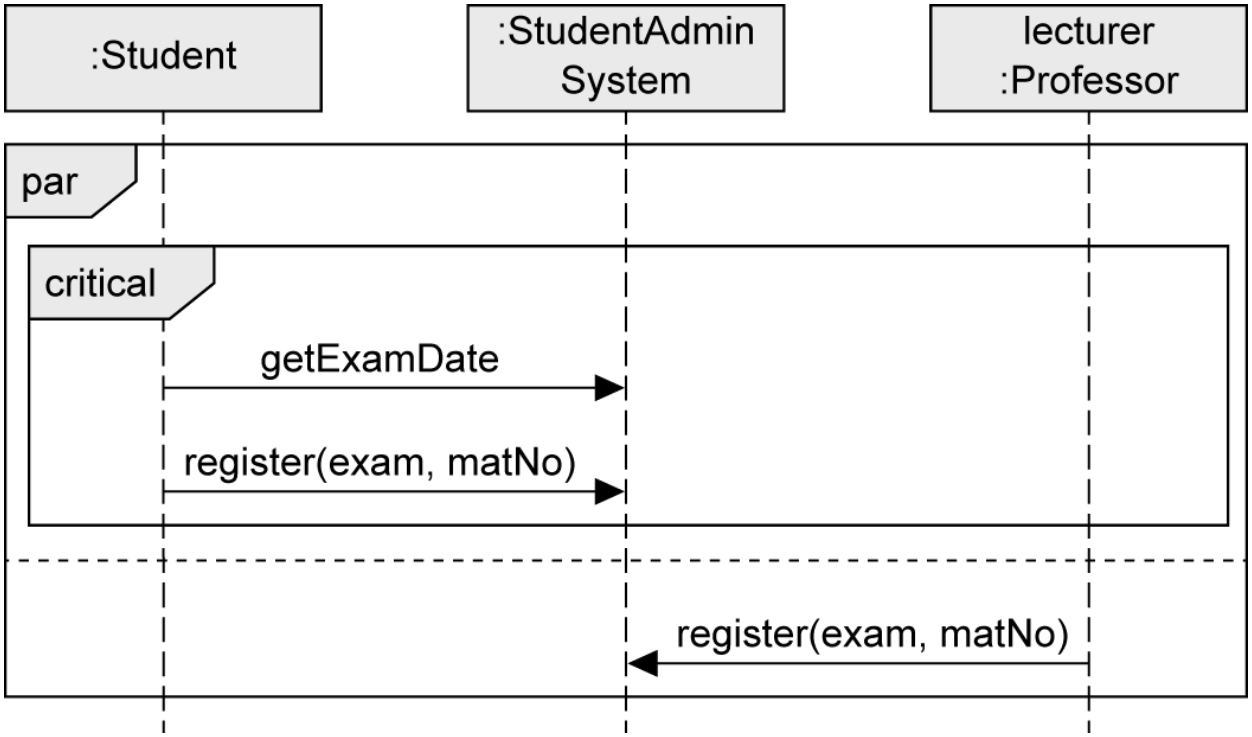
`critical` Fragment

- Atomic area in the interaction (one operand)
- To make sure that certain parts of an interaction are not interrupted by unexpected events
- Order within **`critical`**: default order **`seq`**

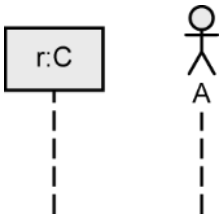
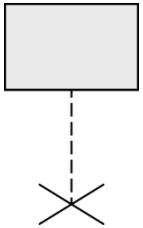
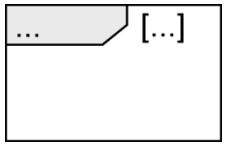


Traces:
T01: a → b → c → d → e
T02: a → c → d → b → e
T03: a → c → d → e → b
T04: c → d → a → b → e
T05: c → d → a → e → b
T06: c → d → e → a → b

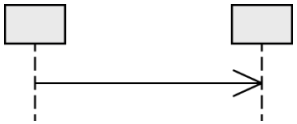
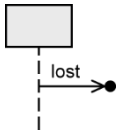
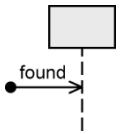
`critical` Fragment - Example



Notation Elements (1/2)

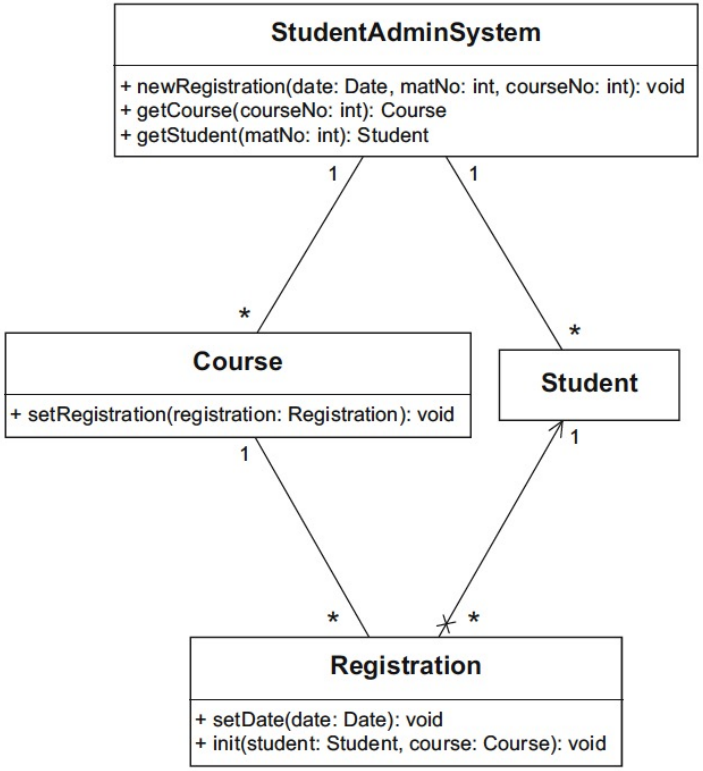
Name	Notation	Description
Lifeline		Interaction partners involved in the communication
Destruction event		Time at which an interaction partner ceases to exist
Combined fragment		Control constructs

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

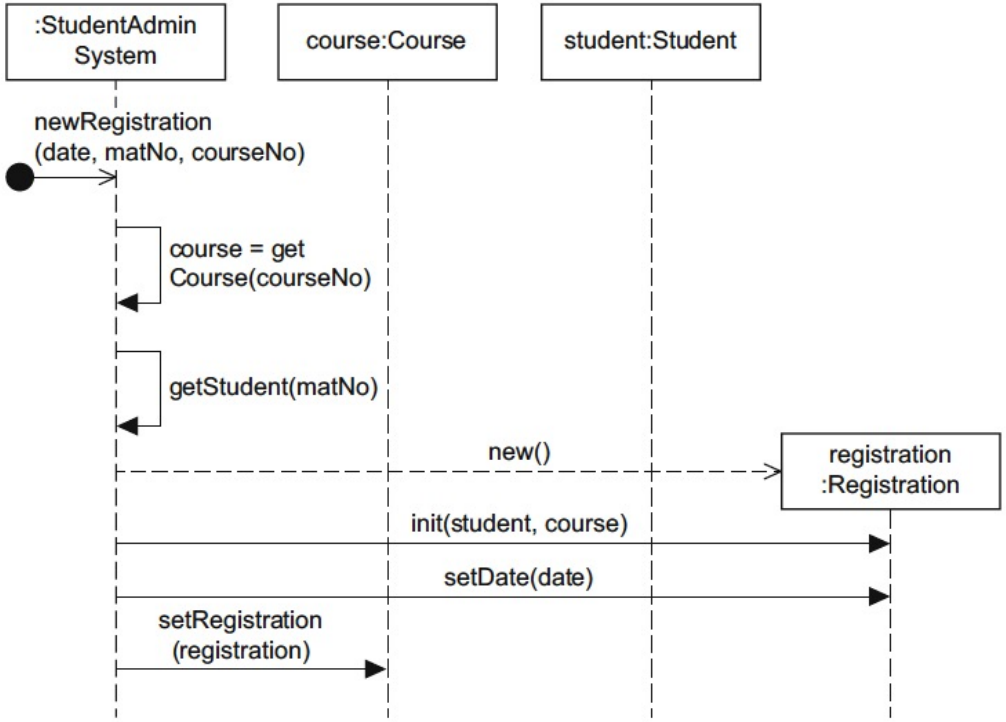
The Connection between a Class Diagram and a Sequence Diagram

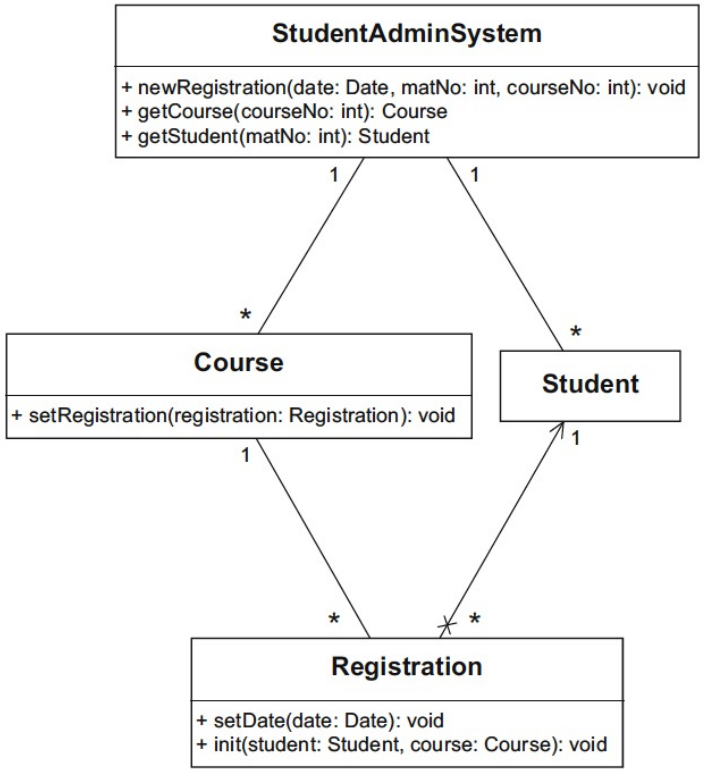
- We have repeatedly stated that the different UML diagrams should not be considered independently of one another; they merely offer different views of a certain content.
- For example, the class diagram models a part of a university system that also includes the student administration system.



The Connection between a Class Diagram and a Sequence Diagram

- We want to depict the communication that is required to create a new registration of a certain student for a certain course.



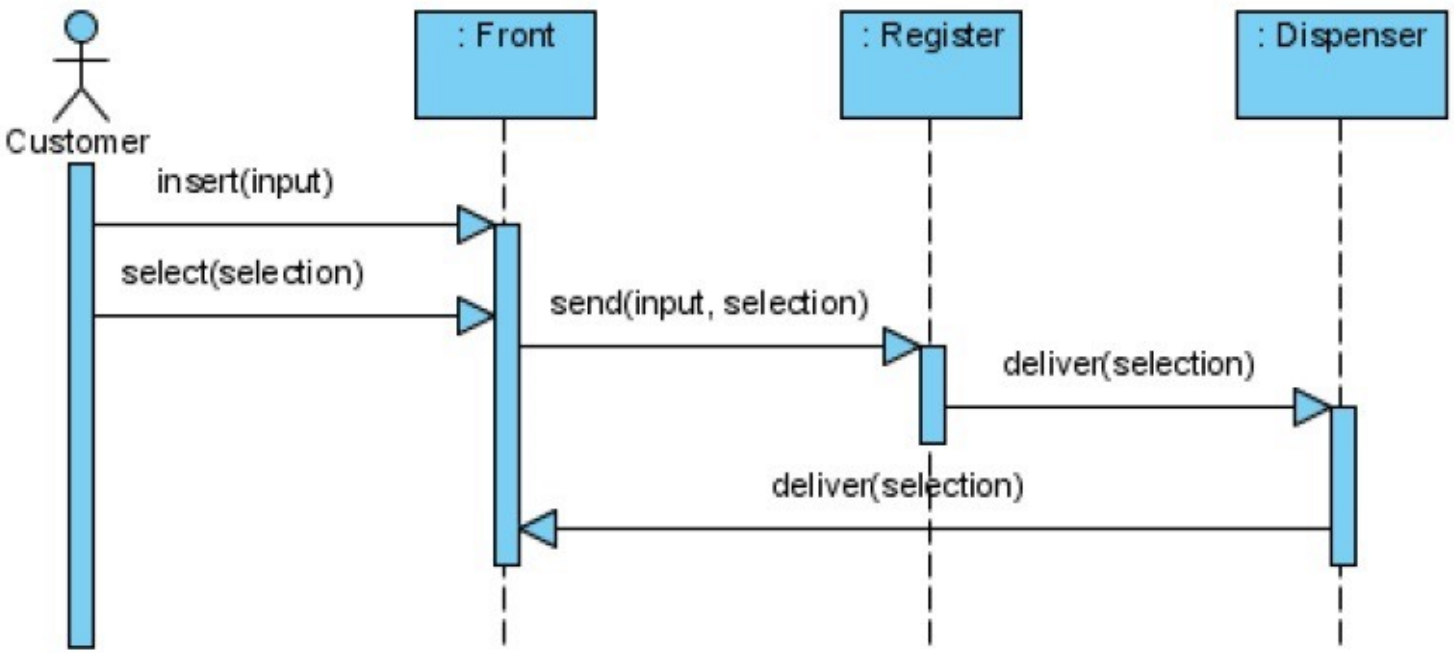


Example

- In a self-service, e.g., money (e.g., ATM), machine, three objects do the work we're concerned with:
  - **The front**: the interface the self-service machine presents to the customer
  - **The money register**: part of the machine where money is collected
  - **The dispenser**: which delivers the selected product to the customer
- The instance sequence diagram may be sketched by using this sequences:
  - The customer inserts money in the money slot in front money collector.
  - The customer makes a selection on the front UI
  - The money travels to the register
  - The register checks to see whether the correct money is in the money collector/dispenser
  - The register updates its cash reserve
  - The register notifies the dispenser which delivers the product (e.g., receipt) to the front of the machine

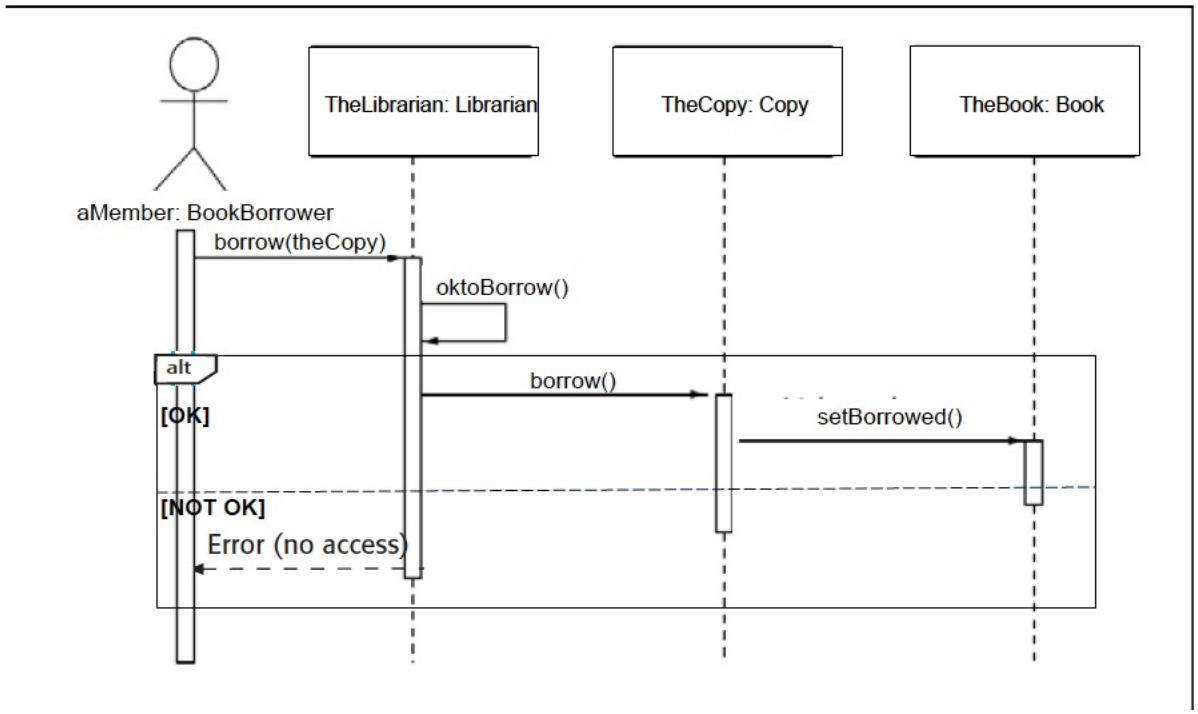
Example

- The customer **inserts** money in the money slot in front money collector.
- The customer **makes a selection** on the front UI
- The money **travels** to the register
- The register checks to see whether the correct money is in the money collector/dispenser
- The register updates its cash reserve
- The register notifies the dispenser which **delivers** the product (e.g., receipt) to the front of the machine

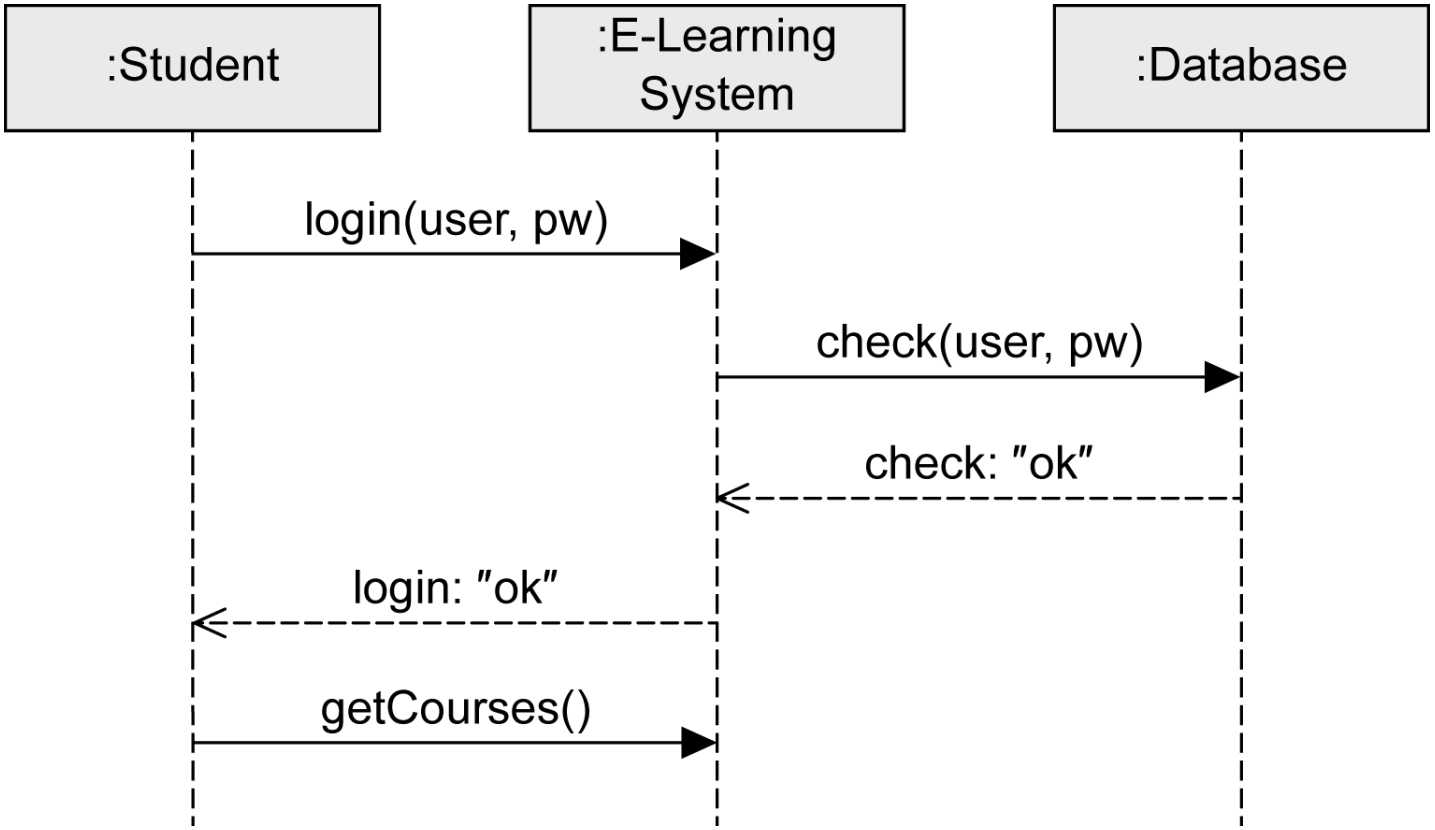


Example

- Library system, three objects do the work we're concerned with
  - **Book Borrower**: that will borrow the book
  - **Copy**: copy of a book
  - **Librarian/Library Staff**: which authorizes and register the borrowing of the borrowed copy.



Example


:Student
:E-Learning System
:Database
login(user, pw)
check(user, pw)
check: "ok"
login: "ok"
getCourses()