

# Sequence Diagram

**Lecture # 25, 26**  
**3, 4 Nov**

Rubab Jaffar  
[rubab.jaffar@nu.edu.pk](mailto:rubab.jaffar@nu.edu.pk)

## Software Design and Analysis CS-3004



# Today's Outline

- Interaction Diagrams
  - Sequence Diagram
  - Collaboration Diagram
- Sequence Diagram
- Sequence Diagram Notations
- Sequence Diagram Example
- Sequence Diagram Advanced Notations
- Sequence Diagram Example

# Interaction Diagrams

- A type of behavior diagram
- A series of diagrams describing the *dynamic behavior* of an object-oriented system.
  - A set of messages exchanged among a set of objects within a context to accomplish a purpose.
- Often used to model the way a use case is realized through a sequence of messages between objects.
- The purpose of Interaction diagrams is to:
  - Model interactions between objects
  - Assist in understanding how a system (a use case) actually works
  - Verify that a use case description can be supported by the existing classes
  - Identify responsibilities/operations and assign them to classes

# Interaction Diagrams (Cont.)

- Two types of interaction diagrams
  - Collaboration Diagrams
    - Emphasizes structural relations between objects
  - Sequence Diagram
    - The subject of today

# What is Sequence Diagram?

- A sequence diagram shows how objects interact in a specific situation. Sequence diagrams provide an **approximation of time** and the general sequence of these interactions by reading the diagram from top to bottom.
- They are also called event diagrams.
- Illustrates how objects interact with each other.
- Emphasizes time ordering of messages.
- Can model simple sequential flow, branching, iteration, recursion and concurrency.

# Sequence Diagrams

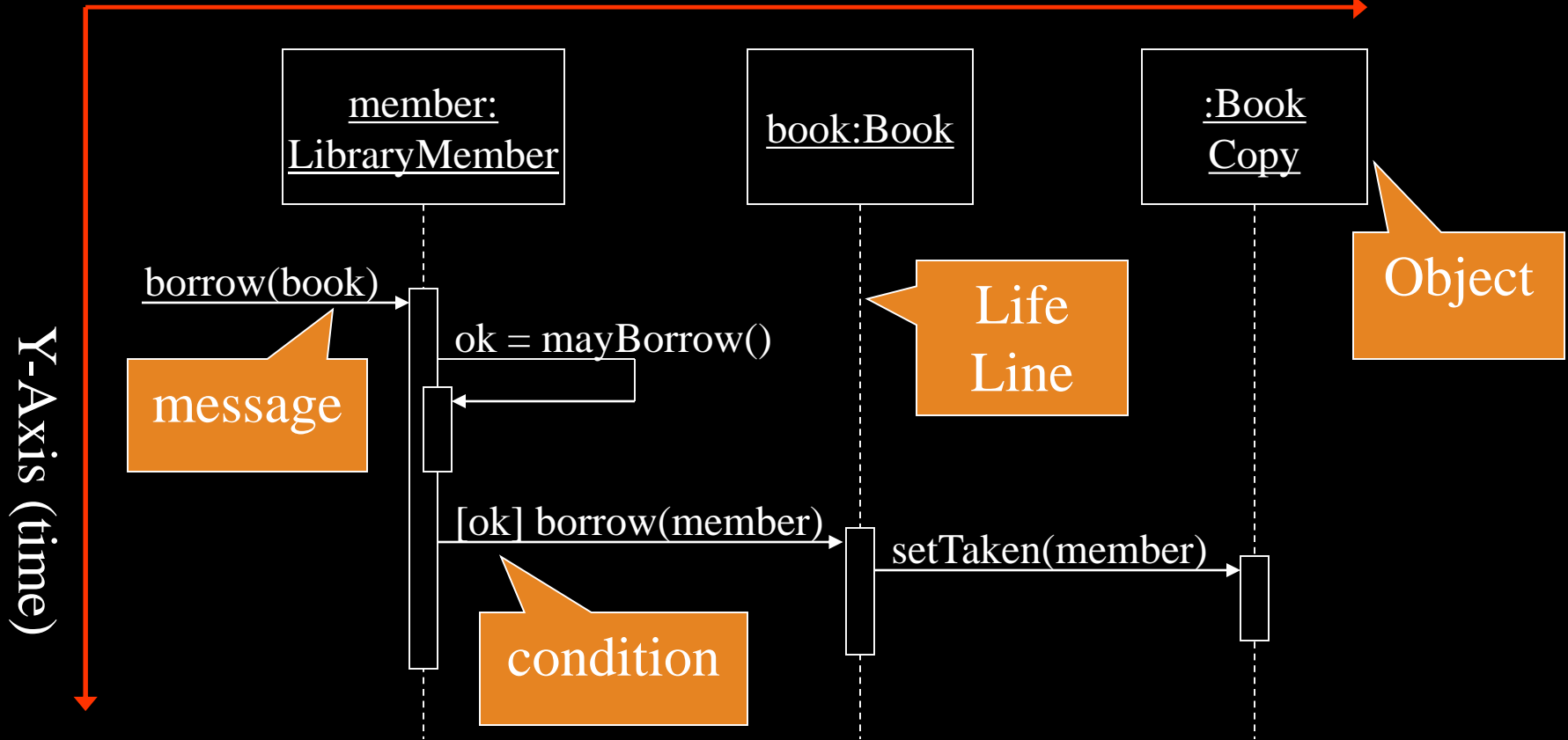
- They focus on **message sequences**, that is, how messages are sent and received between a number of objects.
- Sequence diagrams have two axes:
  - the vertical axis shows time and
  - the horizontal axis shows a set of objects.
- A sequence diagram also reveals the **interaction for a specific scenario**—a specific interaction between the objects that happens at some point in time during the system's execution (for example, when a specific function is used).

# Sequence Diagrams

- A sequence diagram is enclosed by a **rectangular frame** with the name of the diagram shown in a pentagon in the upper-left corner prefixed by *sd*.
- On the **horizontal axis** are the objects involved in the sequence. Each is represented by an object rectangle with the object and/or class name underlined.
- The rectangle along with the vertical dashed line, called the object's lifeline, indicates the object's execution during the sequence (that is, messages sent or received and the activation of the object).
- Communication between the objects is represented as horizontal message lines between the objects' lifelines.
- To read the sequence diagram, start at the top of the diagram and read down to view the exchange of messages taking place as time passes.

# A Sequence Diagram

X-Axis (objects)

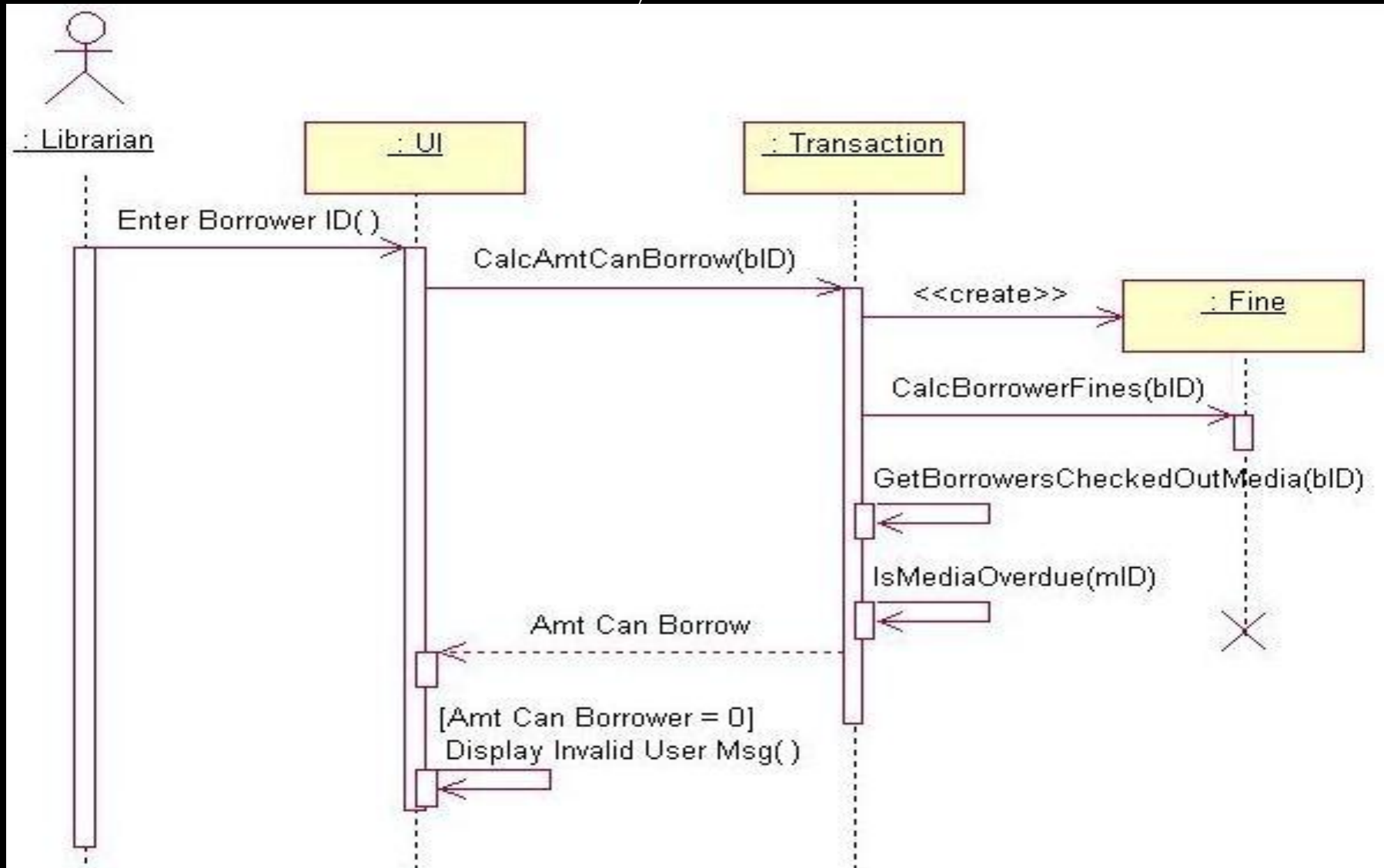




# Generic and Instance Form

- Sequence diagrams can be used in two forms:
  - the generic form and
  - the instance form.
- The **instance form** describes a specific scenario in detail; it documents one possible interaction. The instance form does not have any conditions, branches, or loops; it shows the interaction for just the chosen scenario.(e.g. Successful opening of an account)
- The **generic form** describes all possible alternatives in a scenario; therefore branches, conditions, and loops may be included .(e.g. Opening an account)

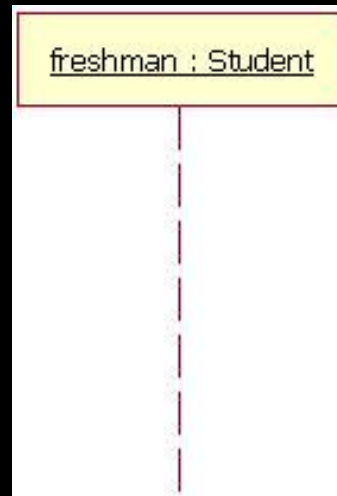
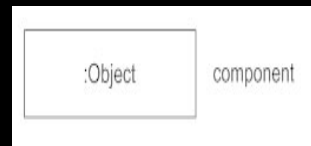
This particular sequence, shown in Figure , documents the interaction that occurs between business objects when determining how many items a borrower can check out of the library.



# Basic Notations

# Class Roles or Participants

Class roles describe the way an object will behave in context. Use the UML object symbol to illustrate class roles, but don't list object attributes.

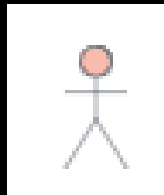


An example of the Student class whose instance name is freshman

Instance Name : Class Name

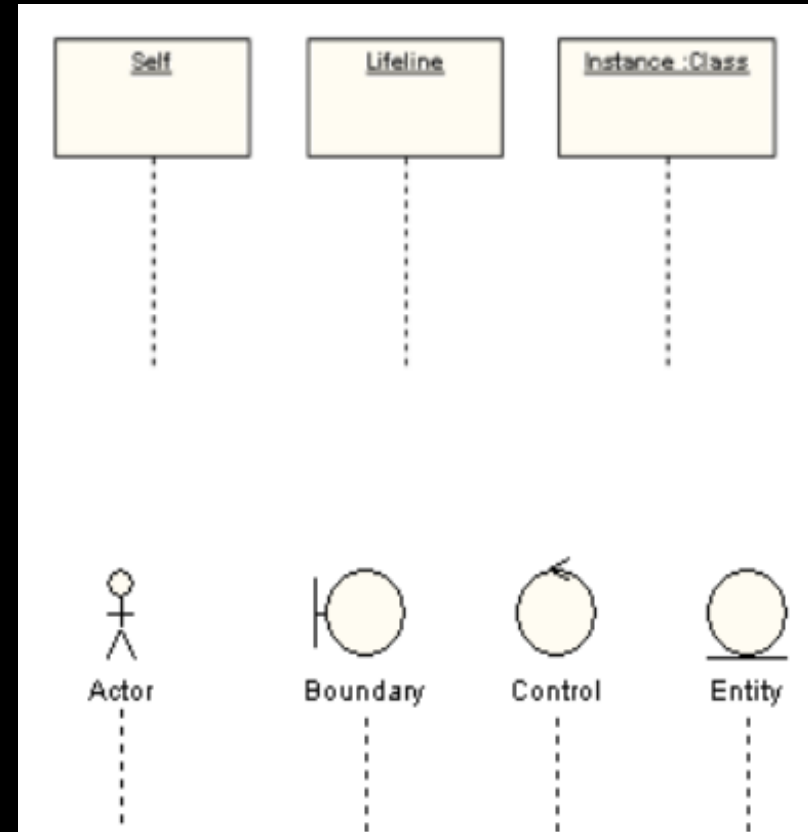
# Actor Symbol

- Represented by a stick figure, actors are entities that are both interactive with and external to the system.



# Lifelines

- An activated object is either executing its own code or is waiting for the return of another object to which it has sent a message.
- The lifeline represents the existence of an object at a particular time; it is drawn as an object icon with a dashed line extending down to the point at which the object stops existing.
- Lifelines indicate the object's presence over time.



# Activation or Execution Occurrence

- Activation boxes represent the time an object needs to complete a task. When an object is busy executing a process or waiting for a reply message, use a thin gray rectangle placed vertically on its lifeline.



# Message

- A message is a communication between objects that conveys information with the expectation that action will be taken.
- Messages can be signals, operation invocations, or something similar (for example, remote procedure calls)
- In the sequence diagram, communication between the objects can be shown with distinct message types.



# Messages (Cont.)



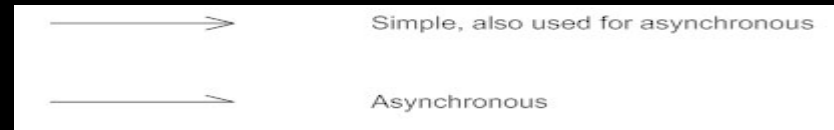
- A message is represented by an arrow between the life lines of two objects.
  - The time required by the receiver object to process the message is denoted by an *activation-box*.
- A message is labeled at minimum with the message name.
  - Arguments and control information (conditions, iteration) may be included.

# Types of Messages

- **Synchronous Message**



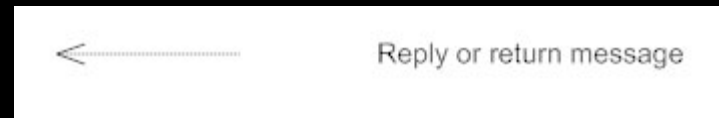
- A synchronous message indicates wait semantics.
- A synchronous message requires a response before the interaction can continue. It's usually drawn using a line with a solid arrowhead pointing from one object to another.



- **Asynchronous message**

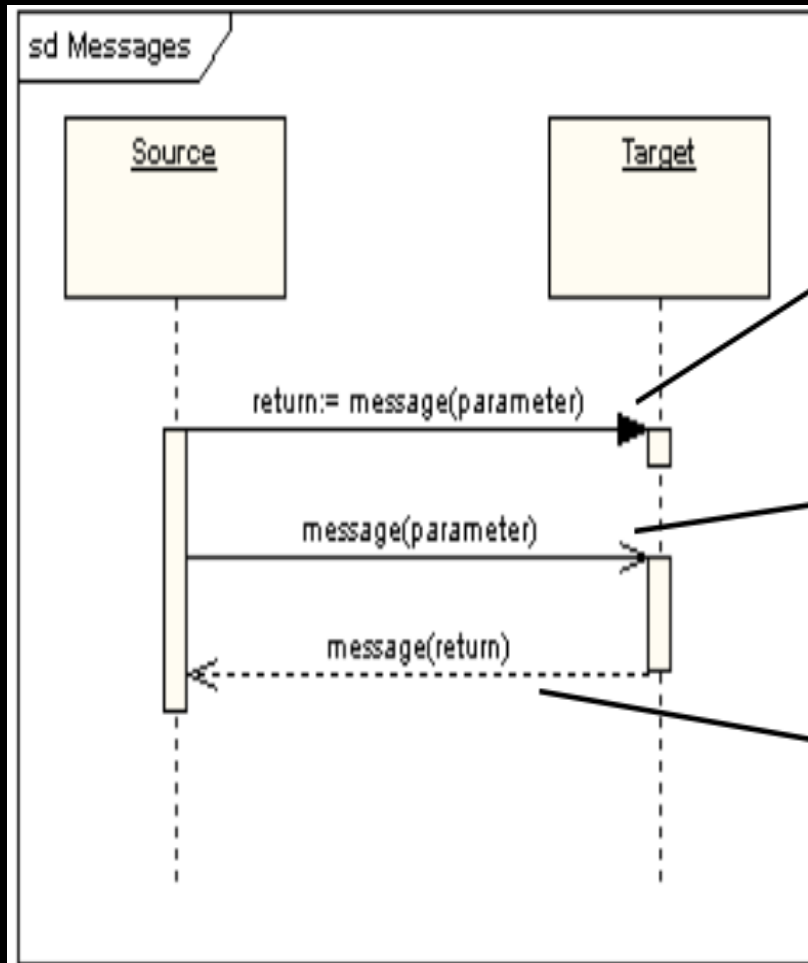
- An asynchronous message reveals that the sending object does not wait, but continues to execute immediately after having sent the message (any result is typically sent back as an asynchronous message as well).

- **Reply or Return Message**



- A reply message is drawn with a dotted line and an open arrowhead pointing back to the original lifeline.

# Example



**synchronous** message line  
denoted by the solid arrowhead

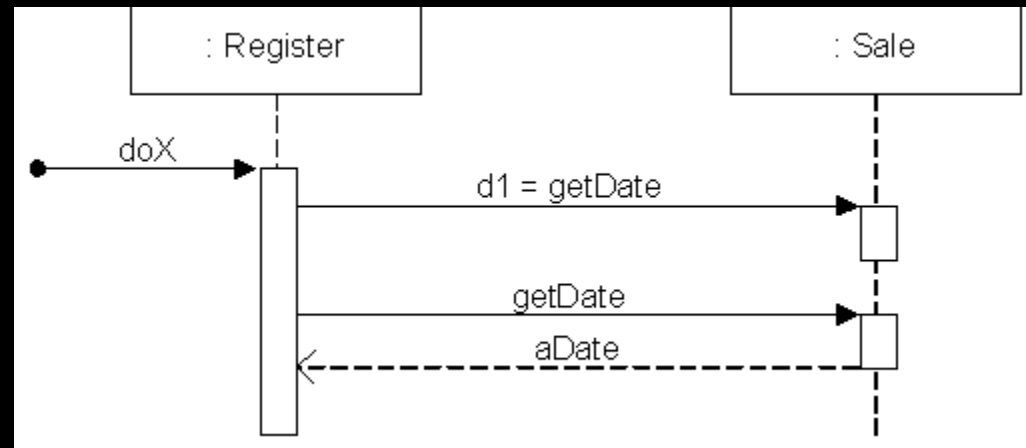
**asynchronous** message line  
denoted by line arrowhead

**return** message line  
denoted by dashed line

# Reply or Return

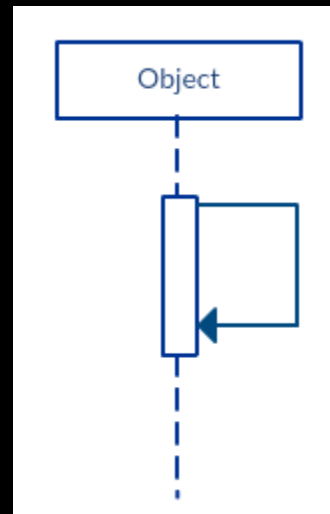
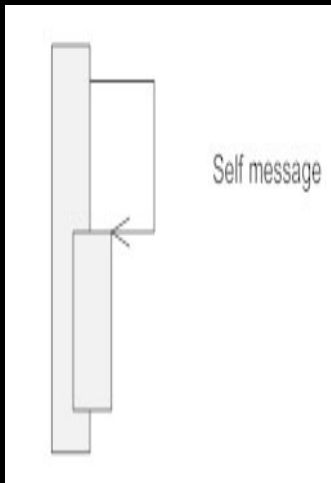
**There are two ways to show return result from a message:**

- Using the message syntax `returnVar=message(parameter)`
- Using a reply(or return) message line at the end of an activation bar.



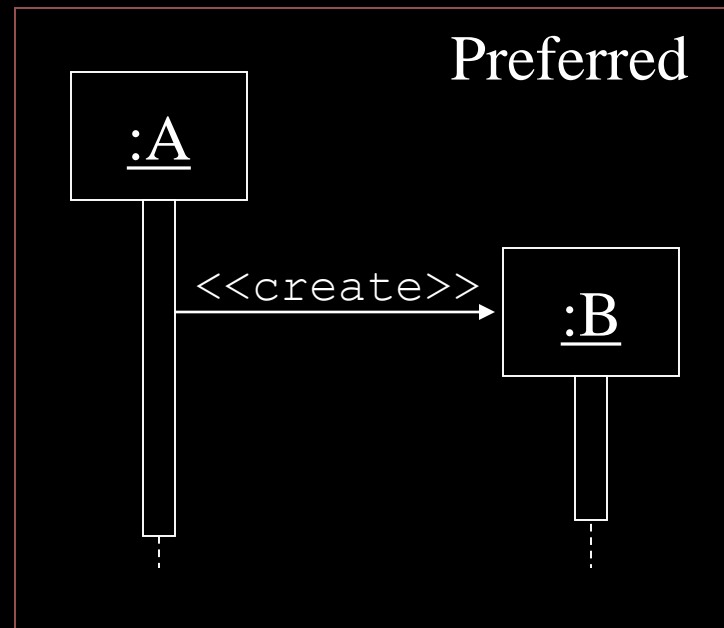
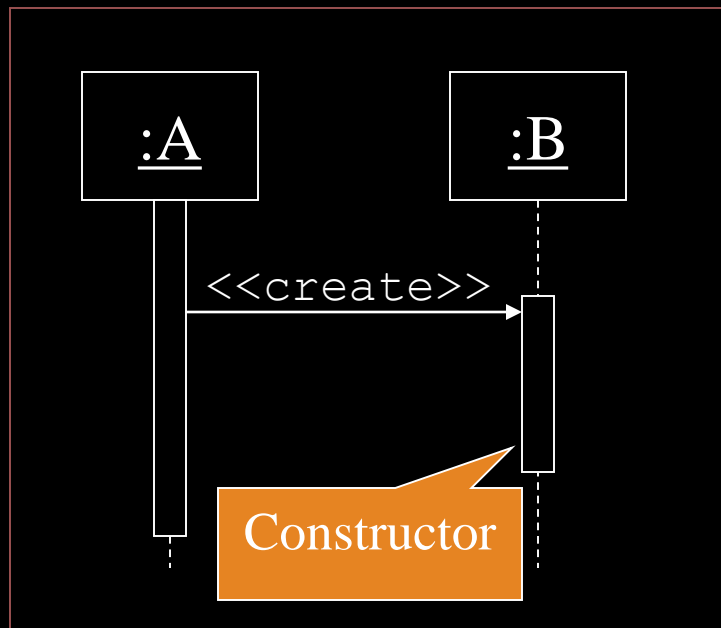
# Types of Messages

- **Self Message/Reflexive message**
- A message an object sends to itself, usually shown as a U shaped arrow pointing back to itself



# Types of Messages

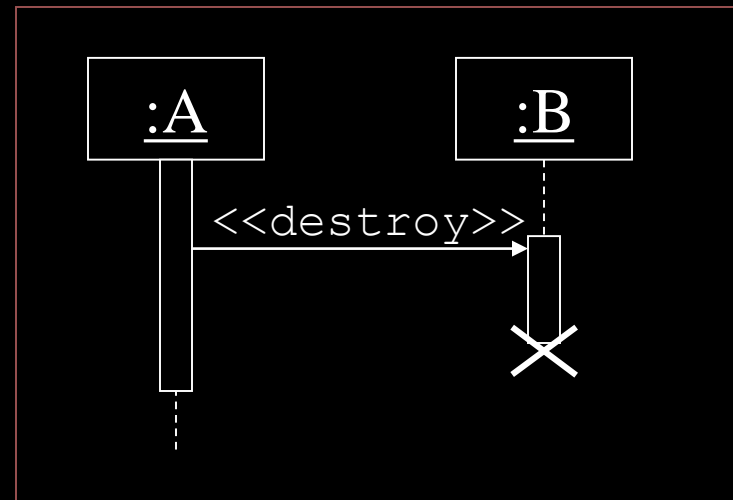
- Create Message
- An object may create another object via a **<<create>>** message.



# Types of Messages

- An object may destroy another object via a **<<destroy>>** message.
  - An object may destroy itself.
  - Avoid modeling object destruction unless memory management is critical.

Objects can be terminated early using an arrow labeled "<< destroy >>" that points to an X. This object is removed from memory. When that object's lifeline ends, you can place an X at the end of its lifeline to denote a destruction occurrence



# Types of Messages

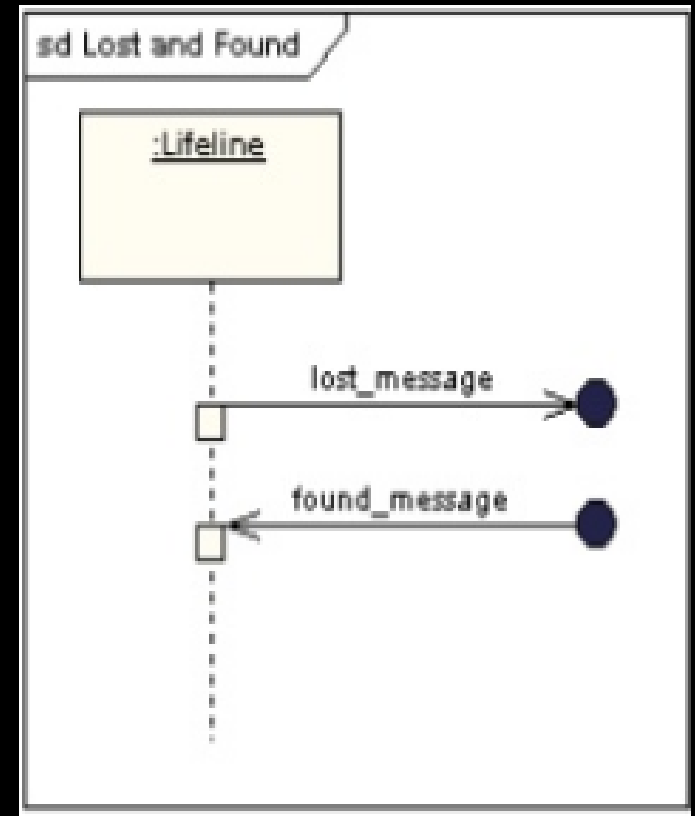
## Found Message



A message sent from an unknown recipient or from a sender not shown on the current diagram, shown by an arrow from an endpoint to a lifeline.

- **Lost Message**

A message sent to an unknown recipient that is not shown on the diagram. It's shown by an arrow going from a lifeline to an endpoint, a filled circle.

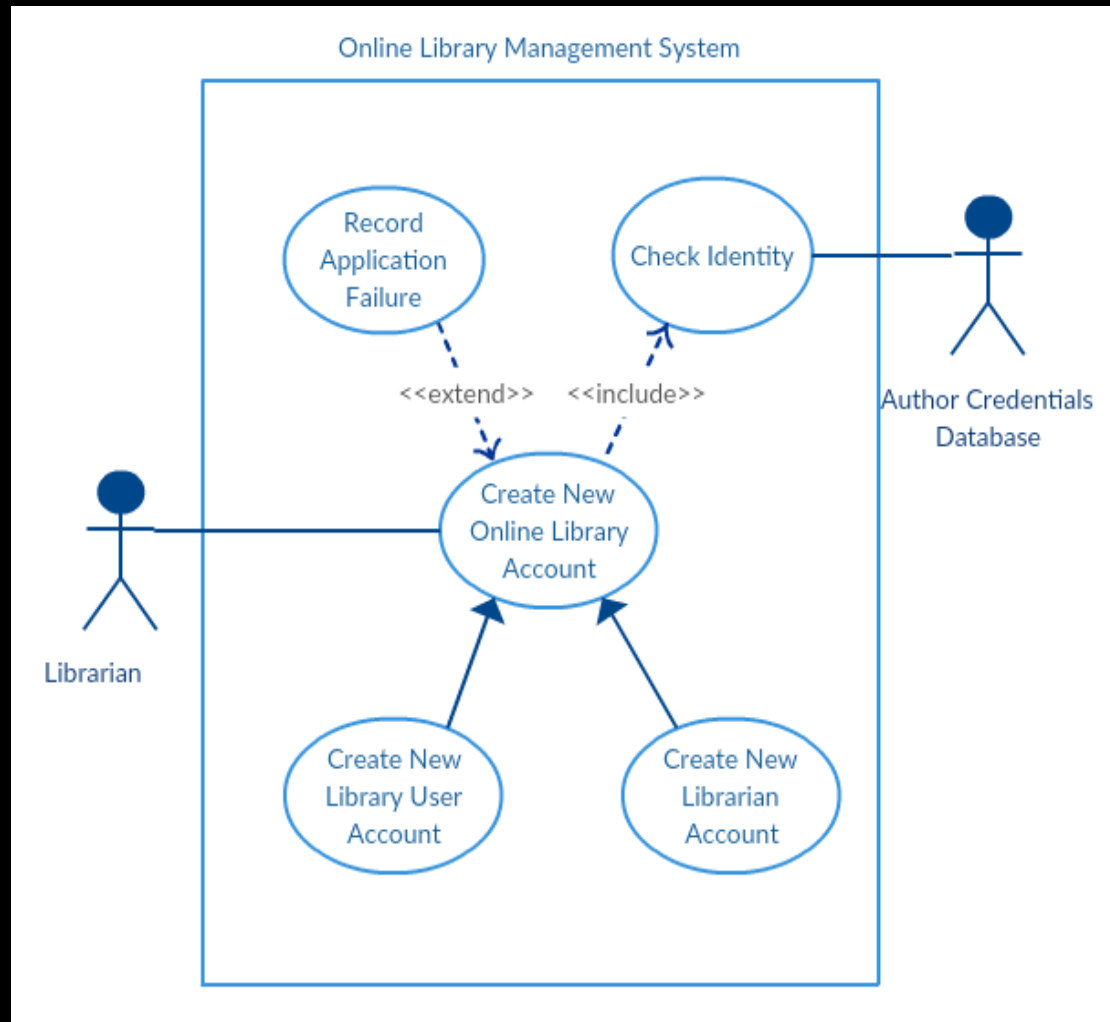




# Example: Balance Lookup Use Case

- Customer asks the bank for his account balance.
- Bank verifies his account number from the account ledger.
- Bank then sends message to checkaccount for getting the balance.
- The bank informs the customer about balance.

# Example: Sequence diagram from use cases



# Use Case Description

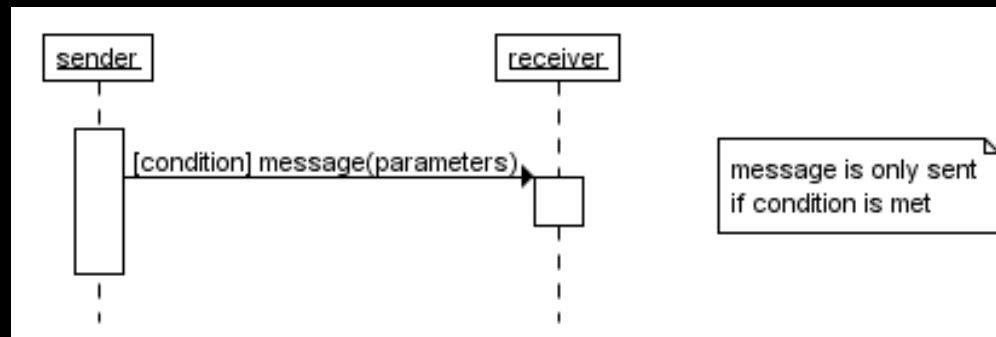
- Here are the steps that occur in the use case named 'Create New Library User Account'.
  - The librarian request the system to create a new online library account
  - The librarian then selects the library user account type
  - The librarian enters the user's details
  - The user's details are checked using the user Credentials Database
  - The new library user account is created
  - A summary of the of the new account's details are then emailed to the user

# Example: 'Create New User Account' -- Sequence diagram.

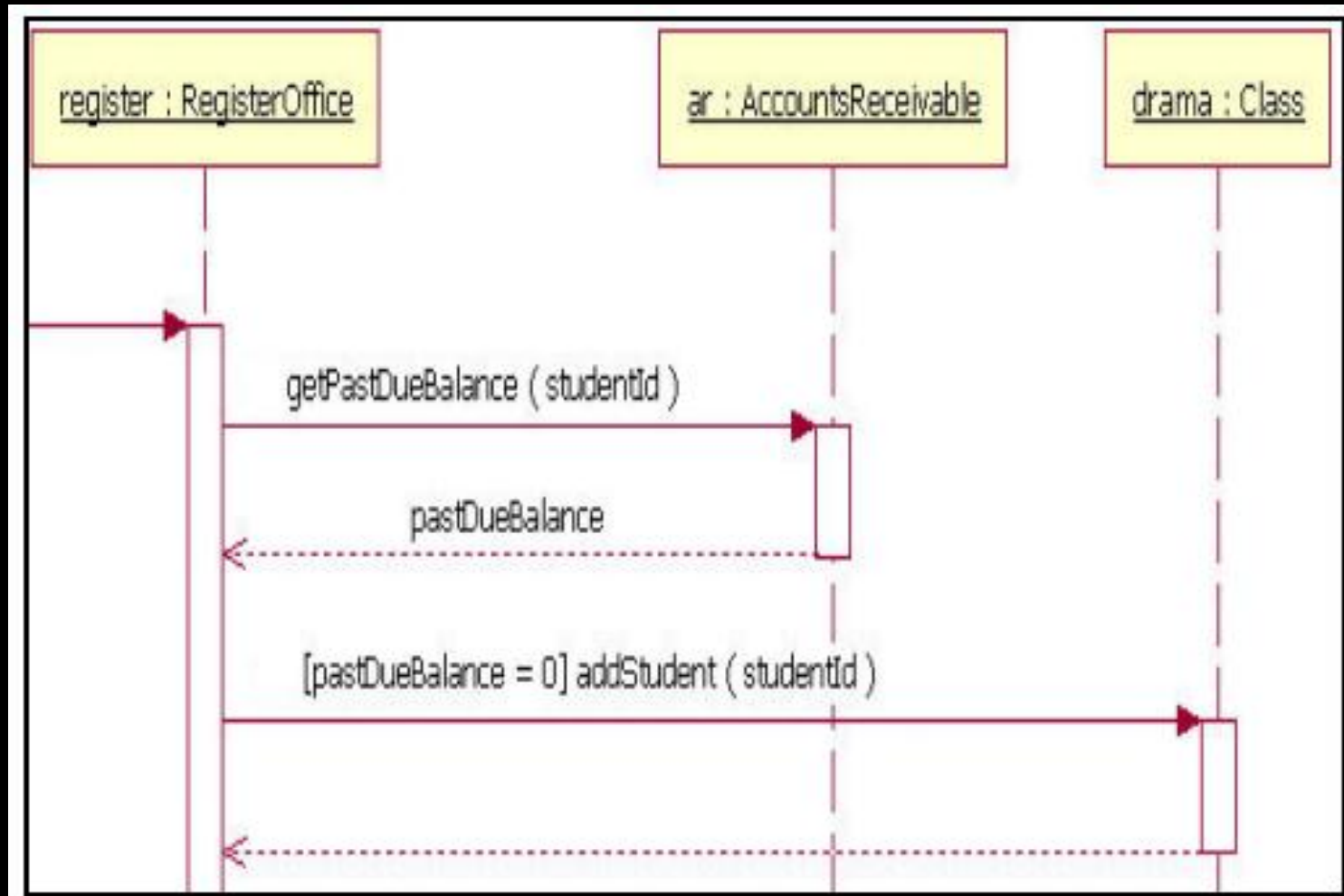
- Before drawing the sequence diagram, it's necessary to identify the objects or actors that would be involved in creating a new user account. These would be;
- Librarian
- Online Library Management system
- User credentials database
- Email system

# Guards

- There will be times when a condition must be met for a message to be sent to the object. i.e Conditional interaction
- There will be certain prerequisite for communication or a message to be sent to the sender.
- These conditions are attributed as “Guard” in sequence diagram, a guard behaves likes “if statements” in the sequence diagram.
- They are used to control the flow of the messages between objects.

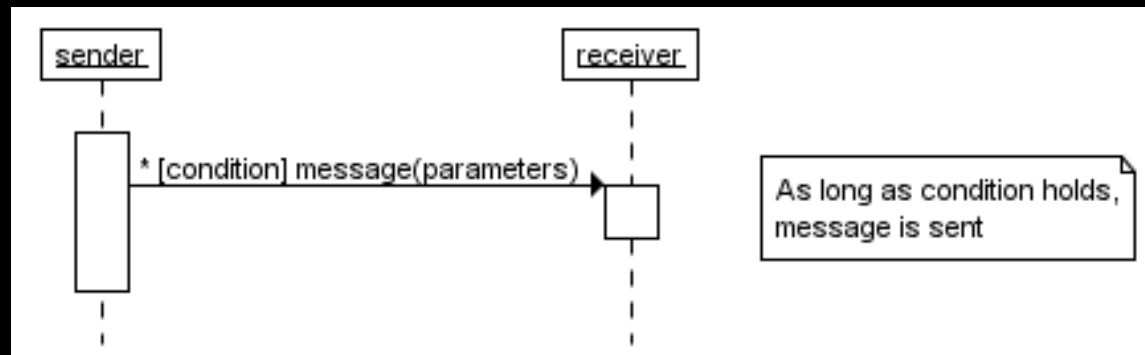


# Guards



# Repeated Interaction

- When a message is prefixed with an asterisk (the '\*'-symbol), it means that the message is sent repeatedly. A guard indicates the condition that determines whether or not the message should be sent (again). As long as the condition holds, the message is repeated.

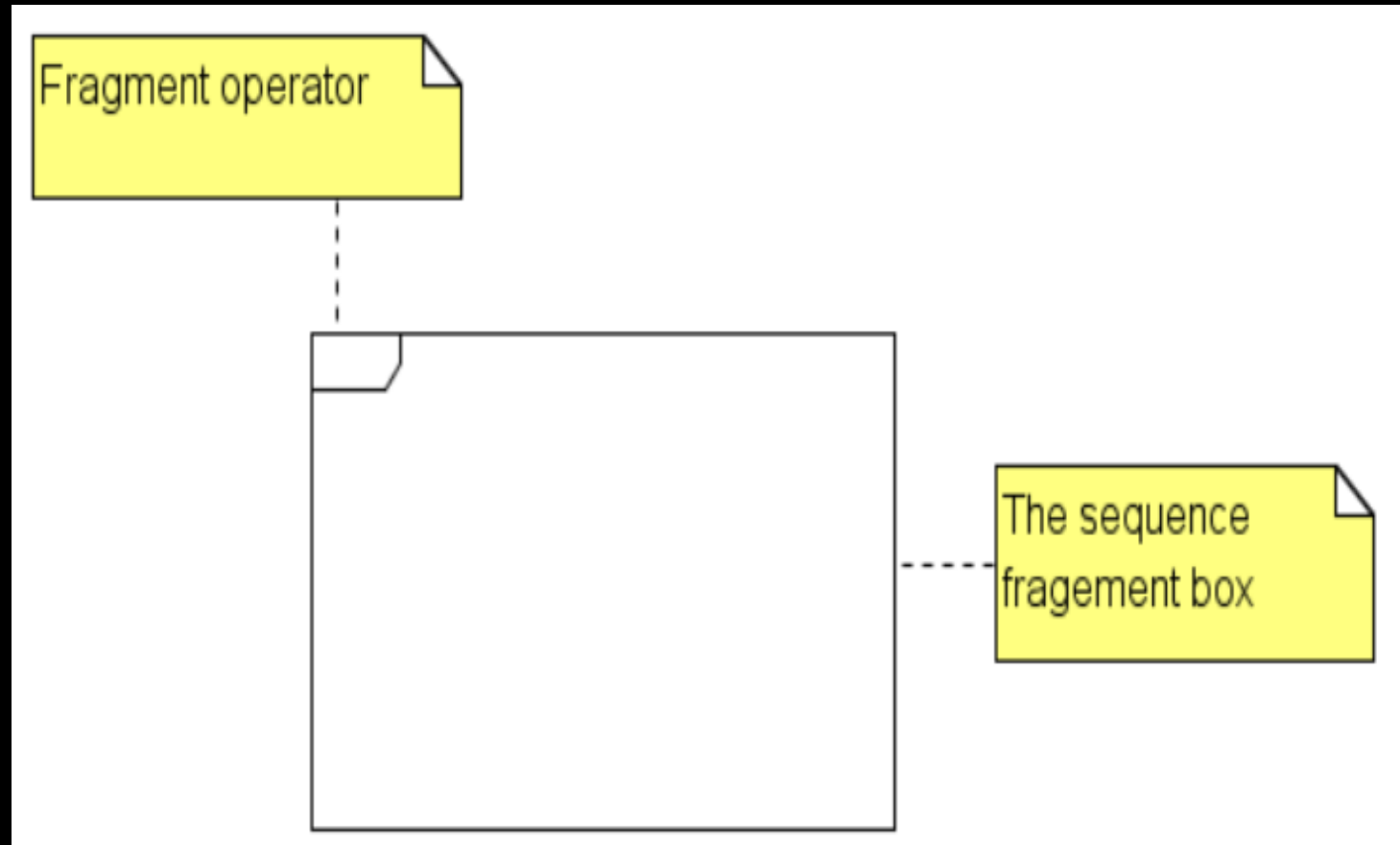


# Fragments

- Sequence diagrams can be broken up into chunks called fragments or combined fragments.
- **Manage complex interactions with sequence fragments**
- It is used to show complex interactions such as alternative flows and loops in a more structured way. On the top left corner of the fragment sits an operator. This – the fragment operator – specifies what sort of a fragment it is.
  - Fragment types: ref, loop, break, alt, opt, parallel
- Sequence fragments make it easier to create and maintain accurate sequence diagrams



# Combined Fragment



# Frame

- If -> (opt) [condition]
- if/else -> (alt) [condition], separated by horizontal dashed line
- loop -> (loop) [condition or items to loop over]

Frame Operator	Description
alt	Alternative fragment for mutual exclusive logic expressed in the guards.
loop	Loop fragment while guard is true. Can also write loop(n) to indicate looping n times.
Opt	Optional fragment that execute if the guard is true.
par	Parallel fragments that execute in parallel.
region	Critical region within which only one thread can run.

# Conditional Behavior- Option

- The option combination fragment is used to model a sequence that, given a certain condition, will occur; otherwise, the sequence does not occur.
- An option is used to model a simple "if then" statement
  - Example, if there are fewer than five donuts on the shelf, then make two dozen more donuts.

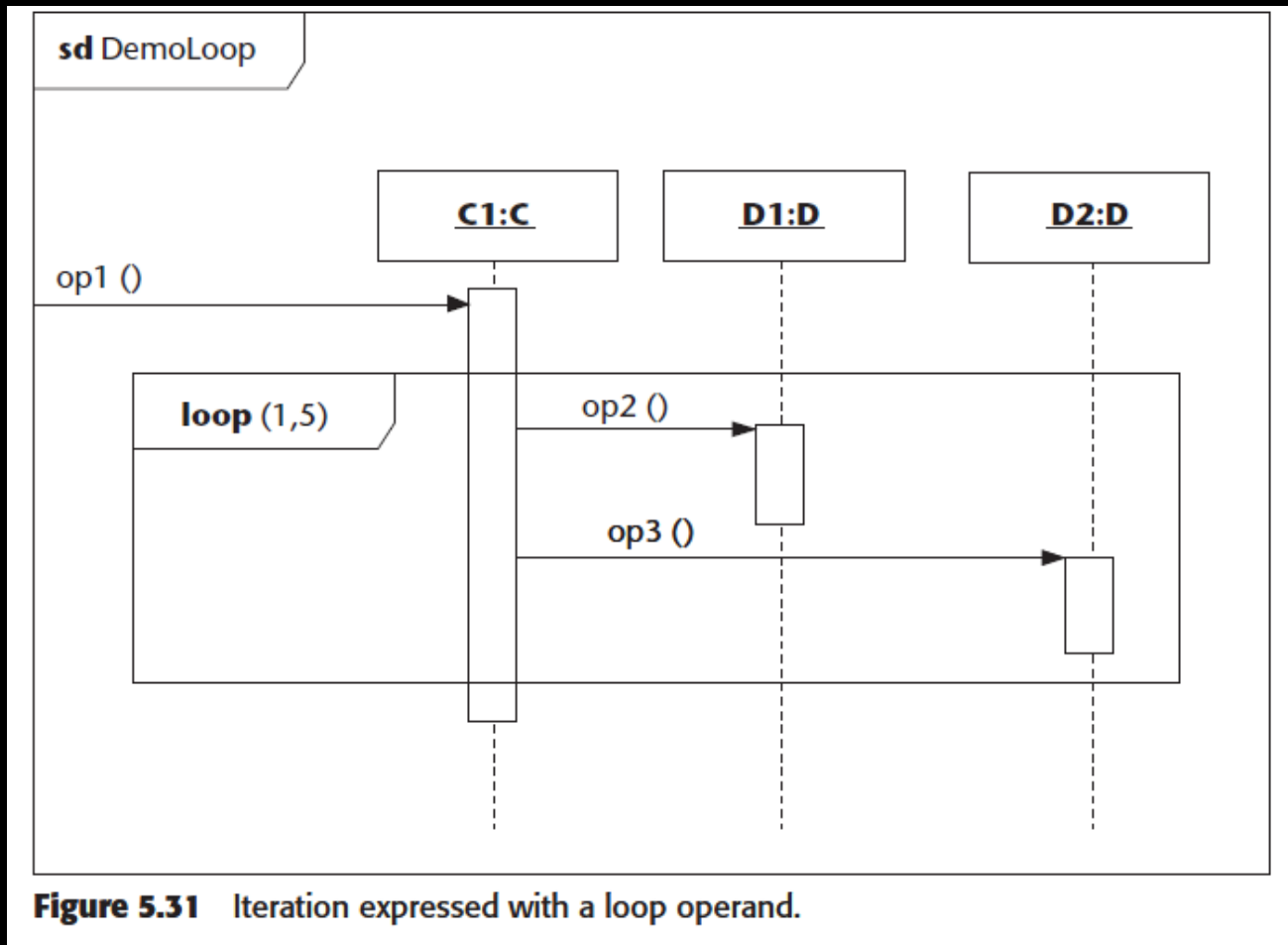
# Conditional Behavior-Alt

- When showing conditional behavior, the interaction operator keyword **alt** is put in the pentagram, the fragment is partitioned horizontally with a dashed line separator, and constraints are shown in square brackets .
- At most one of the alternatives occurs;

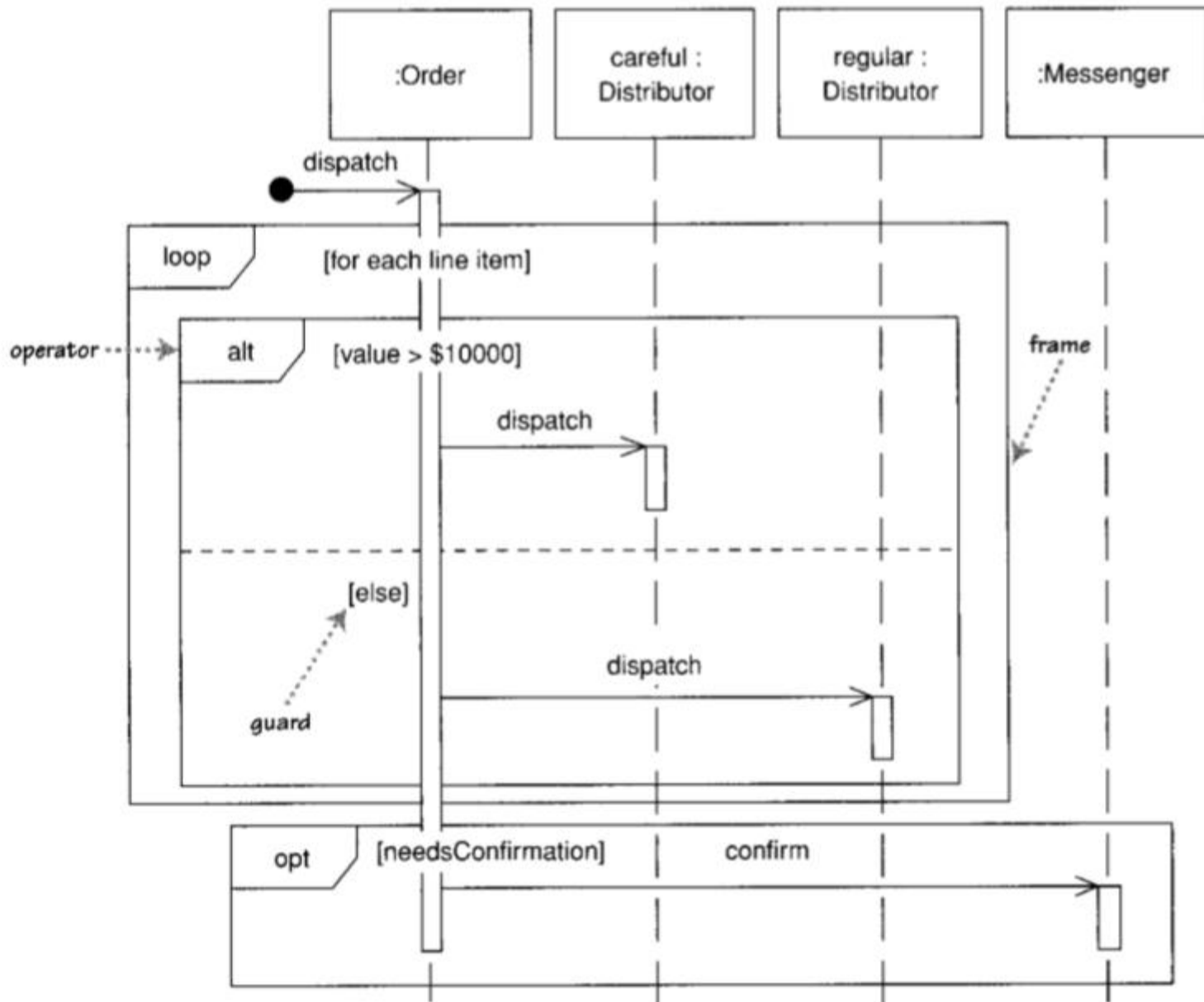
# Loops

- A repetition or loop within a sequence diagram is depicted as a frame.
- In the frame's name box the text "loop" is placed.
- Loops are designated by placing the interaction operator keyword loop in the pentagram. Textual syntax of the loop operand is "loop [ '(' <minint> [,<maxint> ] ')' ]" .
- Inside the frame's content area the loop's guard is placed towards the top left corner, on top of a lifeline.

# A simple example of two operations being repeated five times.

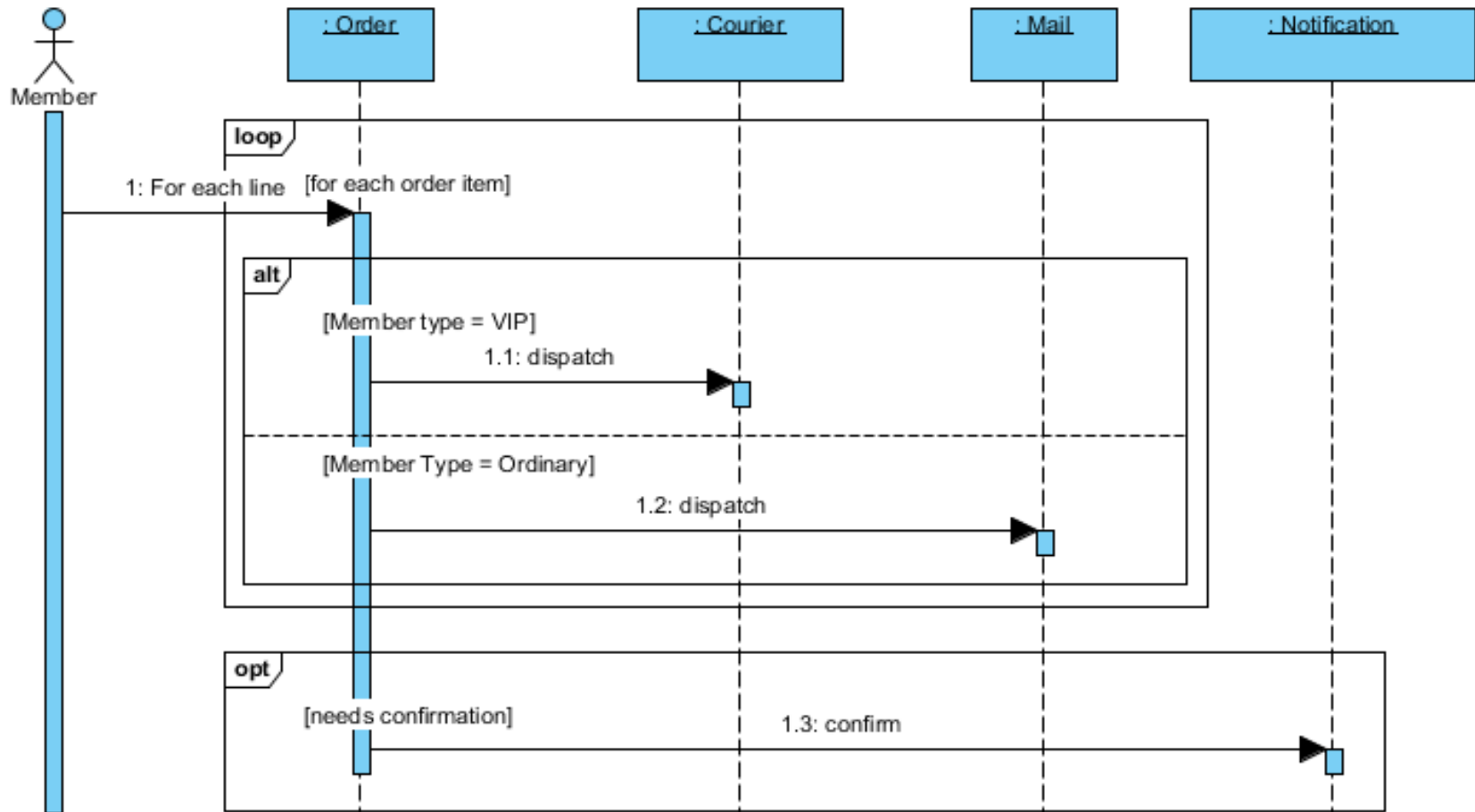


**Figure 5.31** Iteration expressed with a loop operand.



# Indicating selection and loops

sd Place Order Scenario

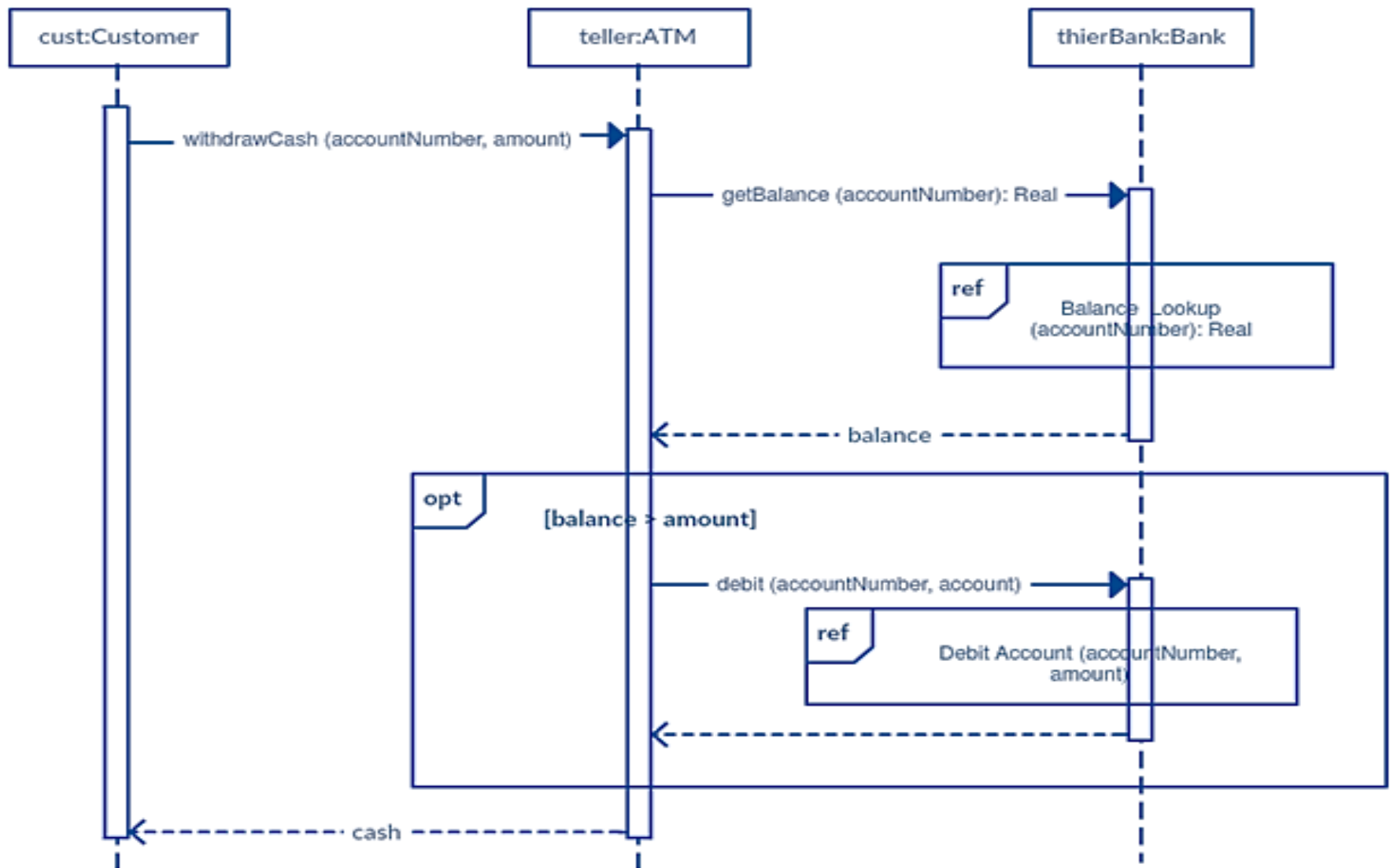




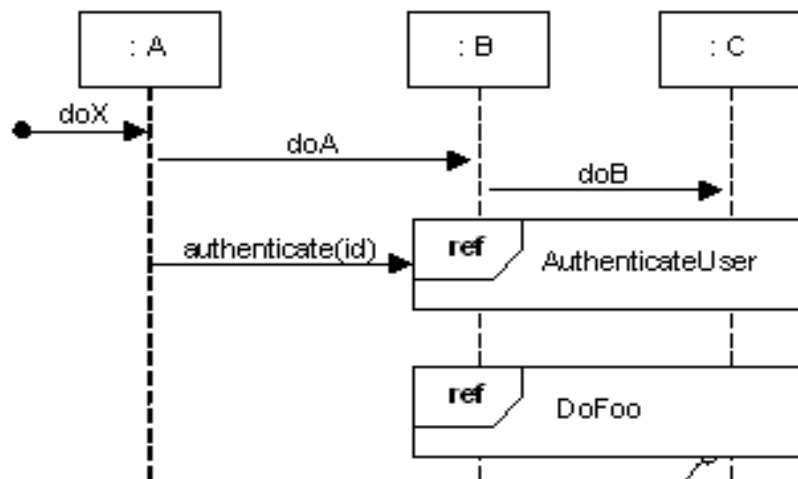
# Reference Fragment

- You can use the **ref** fragment to manage the size of large sequence diagrams. It allows you to reuse part of one sequence diagram in another, or in other words, you can reference part of a diagram in another diagram using the ref fragment.
- To specify the reference fragment, you have to mention 'ref' in the name box of the frame and the name of the sequence diagram that is being referred to inside the frame.

# Reference Fragment- Example



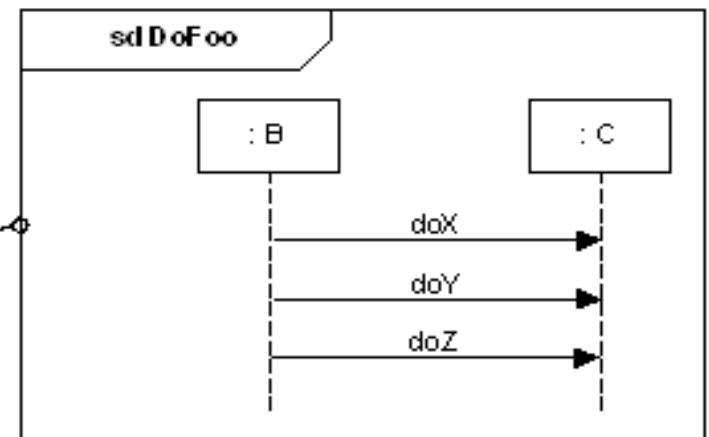
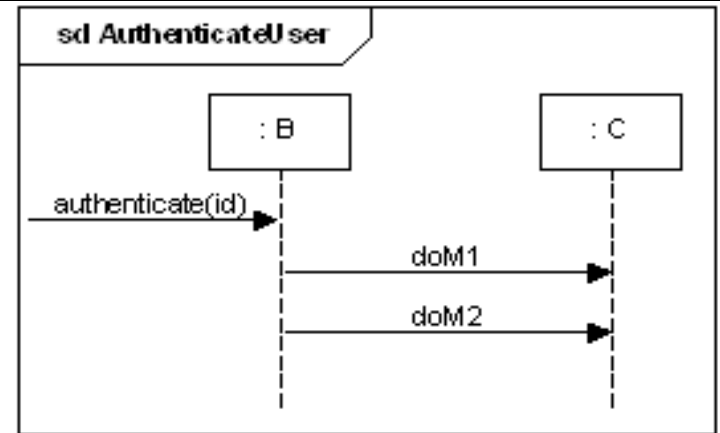
# Reference Fragment-Example



interaction occurrence

note it covers a set of lifelines

note that the sd frame it relates to has the same lifelines: B and C

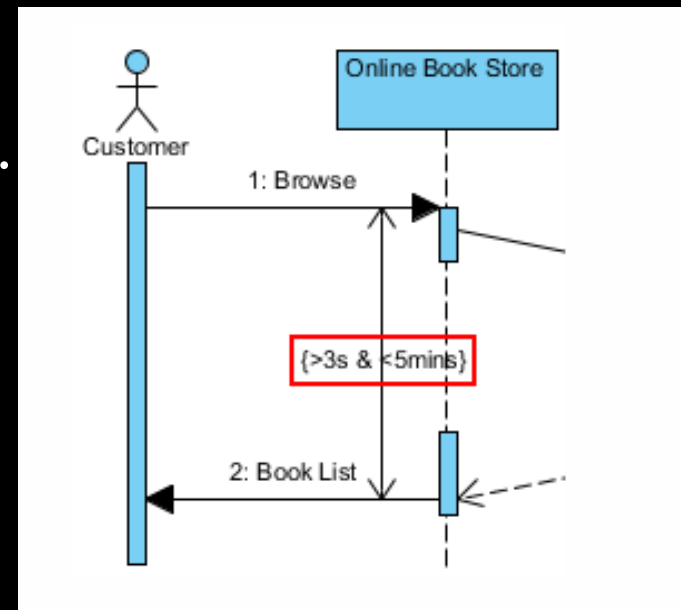


# Common Operators for Interaction Frames

Operator	Meaning
alt	<b>Alternative multiple fragments:</b> only the one whose condition is true will execute.
opt	<b>Optional:</b> the fragment executes only if the supplied condition is true. Equivalent to an alt only with one trace.
par	<b>Parallel:</b> each fragment is run in parallel.
loop	<b>Loop:</b> the fragment may execute multiple times, and the guard indicates the basis of iteration.
region	<b>Critical region:</b> the fragment can have only one thread executing it at once.
neg	<b>Negative:</b> the fragment shows an invalid interaction.
ref	<b>Reference:</b> refers to an interaction defined on another diagram. The frame is drawn to cover the lifelines involved in the interaction. You can define parameters and a return value.
sd	<b>Sequence diagram:</b> used to surround an entire sequence diagram.

# Specifying Timing Requirements

- When modeling a real-time system, or even a time-bound business process, it can be important to consider the length of time it takes to perform actions.
- Assume you need to specify the time limit between Browse message and Book List message, you therefore, have to add duration constraint between them.
- For example, it should take more than 3 seconds but less than 5 minutes. You can enter  $> 3s$  &  $< 5mins$ .



# How to Produce Sequence Diagrams

1. Decide on Context:
  - a) Identify behavior (or use case) to be specified
2. Identify structural elements:
  - a) Model objects (classes)
  - b) Model lifelines
  - c) Model activations
  - d) Model messages
  - e) Model Timing constraints
3. Elaborate as required

# Why not just code it?

- Sequence diagrams can be somewhat close to the code level. So why not just code up that algorithm rather than drawing it as a sequence diagram?
  - A good sequence diagram is still a bit above the level of the real code (not all code is drawn on diagram)
  - Non-coders can do sequence diagrams
  - Easier to do sequence diagrams as a team
  - Can see many objects/classes at a time on same page (visual bandwidth)

# Sequence Diagrams and Use Cases

## System Sequence Diagram

- System sequence diagrams are actually a sub-type of sequence diagrams.
- Sequence diagrams show the progression of events over a certain amount of time, while system sequence diagrams go a step further and present sequences for specific use cases.



# Sequence Diagrams and Use Cases

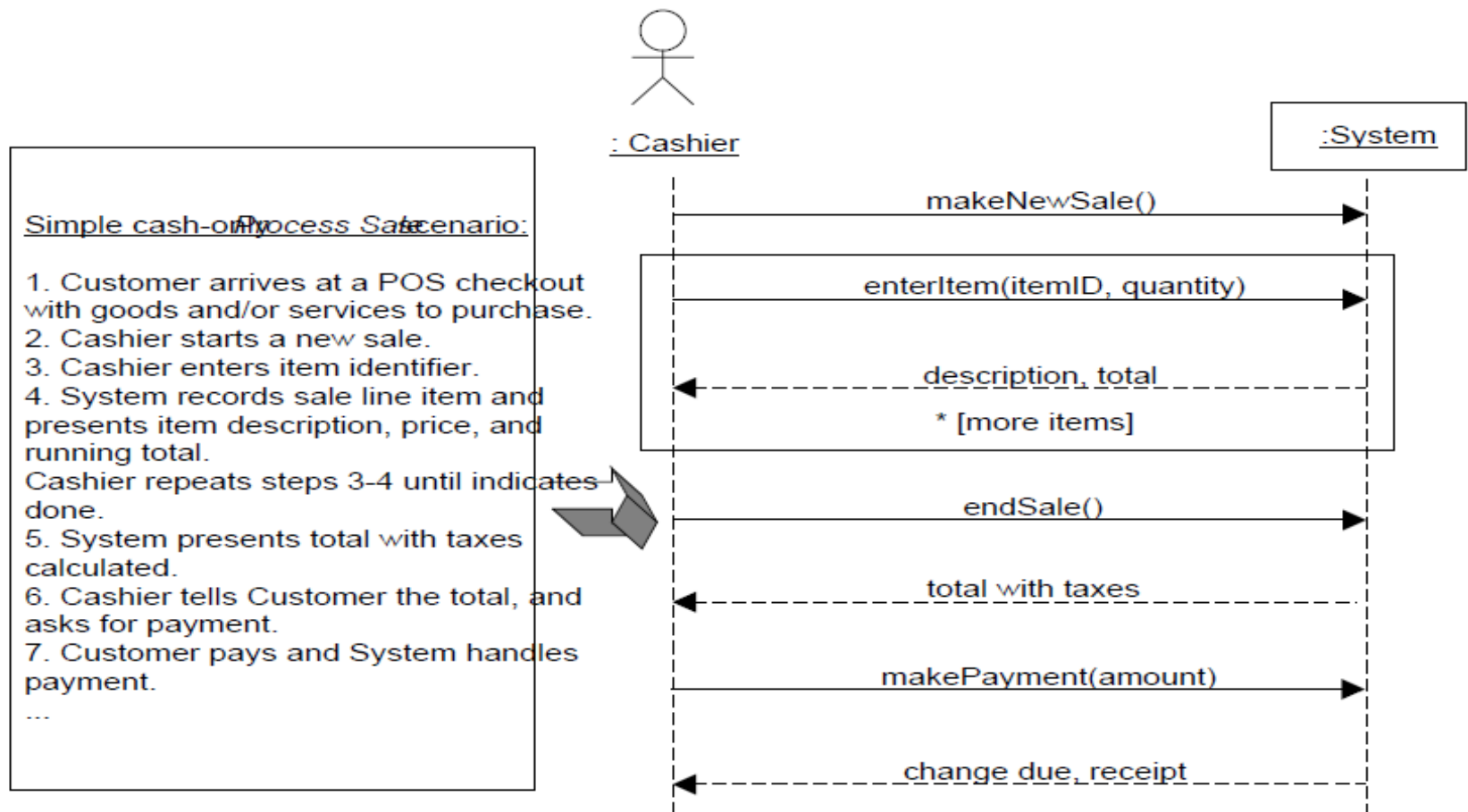
## System Sequence Diagram

- SSD: The elements participating (exchanging messages) in a system sequence diagram are Actors and Systems. The messages exchanged by these elements could be any type depending on the systems (from web service calls to data input from a human).
- SD: The elements participating in a sequence diagram are objects (instances of various classes). The messages exchanged by these elements are method invocations.

# System Sequence Diagram (SSD)

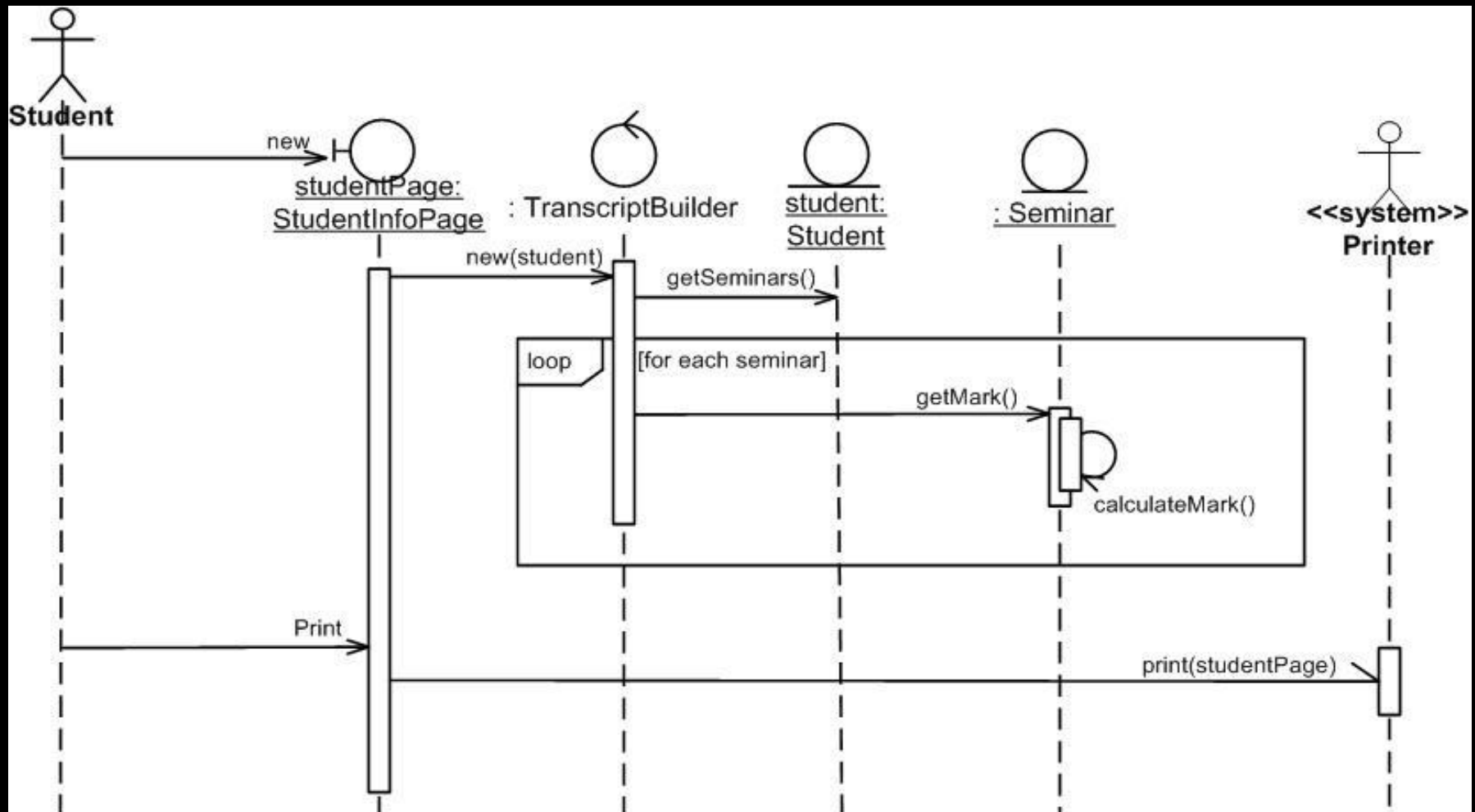
- A system sequence diagram (SSD) is a picture that shows, for a particular scenario of a use case, the events that external actors generate, their order, and inter-system events.
- All systems are treated as a black box; the emphasis of the diagram is events that cross the system boundary from actors to systems.
- An SSD should be done for the main success scenario of the use case, and frequent or complex alternative scenarios.

# System Sequence Diagram (SSD)

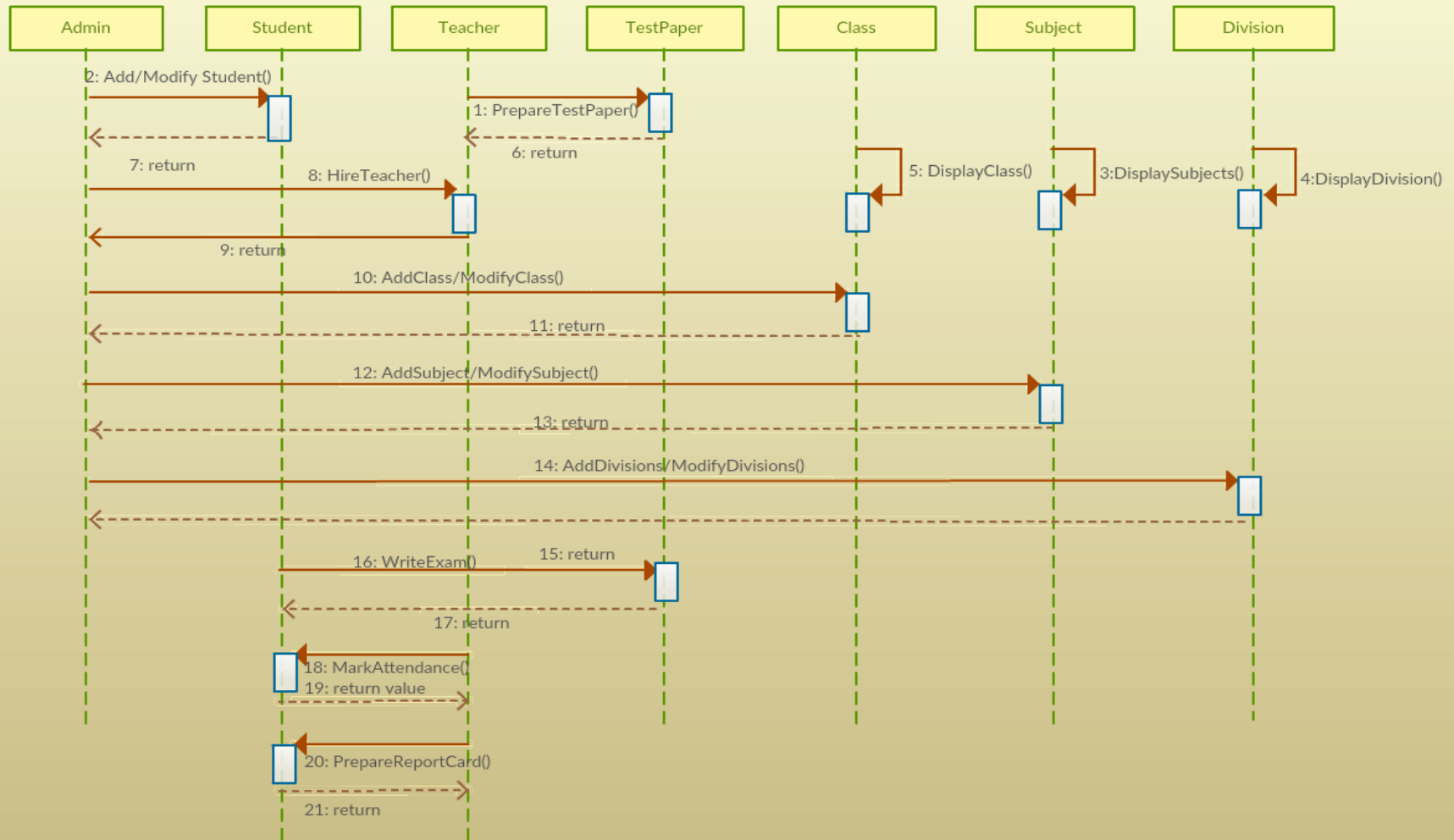


SSDs are derived from use cases.

# Sequence Diagram for Students Seminar Transcript



# Sequence Diagram of an Online Exam System



# Your Turn

## Add Calendar Appointment:

The scenario begins when the user chooses to add a new appointment in the UI. The UI notices which part of the calendar is active and pops up an Add Appointment window for that date and time. The user enters the necessary information about the appointment's name, location, start and end times. The UI will prevent the user from entering an appointment that has invalid information, such as an empty name or negative duration. The calendar records the new appointment in the user's list of appointments. Any reminder selected by the user is added to the list of reminders. If the user already has an appointment at that time, the user is shown a warning message and asked to choose an available time or replace the previous appointment. If the user enters an appointment with the same name and duration as an existing group meeting, the calendar asks the user whether he/she intended to join that group meeting instead. If so, the user is added to that group meeting's list of participants.

# Your Turn (SD by ECB classes)

- Customer requests reservation clerk for the room reservation.
- Reservation clerk checks the availability of hotel rooms through hotel control system.
- Customer can also request discount.
- Reservation clerk provide information about total cost by checking the rate and bill from the hotel control system.
- Customer confirms the reservation of room and room is reserved.
- Customer is charged by the hotel control system through the credit card.

# Your Turn

- Check in Process Use Case
- Person arrives at desk, gives name.
- Receptionist gets map of available Rooms, chooses one.
- Receptionist gets details of user
- Receptionist enters Person's details on Room Management system against chosen room for room confirmation
- Receptionist gives key to Person.



# Your Turn: Example for Withdraw

- Joe withdraws \$20 from the ATM (flow of events)
  - The process begins when Joe inserts his card into the card reader. The card reader reads the number on Joe's card, then tells the ATM screen to initialize itself
  - The ATM verifies the card against account and prompts Joe for his PIN.
  - Joe enters PIN.
  - Joe's PIN is validated and the ATM prompts him for a transaction
  - Joe selects Withdraw Money
  - The ATM prompts Joe for an amount.
  - Joe enters \$ 20.
  - The ATM verifies that Joe's account has sufficient funds and subtracts \$ 20 from his account.
  - The ATM dispenses \$ 20 and ejects Joe's card

# Your Turn

- The convener selects a case on the Disbursement GUI (graphical user interface) screen.
- The Disbursement GUI sends the message `QueryCase()` to the Disbursement Control object, requesting it to query payment-related details about the case.
- The Disbursement Control object services this request by passing a number of messages to the Case object. These include `GetPaymentAmount()`, `GetPcMember()`, and `GetPcAccount()`. These are requests to retrieve payment and Peace Committee member information relevant to the case.
- The convener approves the disbursement for the case.
- The GUI responds to the approval by sending the message `CreatePayments()` to the Disbursement Control object.
- The Disbursement Control object responds by sending a `Create()` message to each required Payment object.
- The Payment object sends a `Withdraw()` message to the cash account and a `Deposit()` message to the Peace Committee member account.
- The Disbursement Control object finishes the process by sending the message `SetPaidStatus()` to the Case object to indicate that payments have been made.

# Your Turn

## Search page by author name

- The customer specifies an author on the search page and then presses the search button.
- The system validates the customer's search criteria.
- The system searches the catalog for books specified with the author.
- When the search is complete, the systems displays the search results on the search page.



That is all