

Session 3

Sequence diagrams

Sequence diagrams show the messages between objects over time for one scenario. Each one should trace a single use case from Session 2 through the classes of Session 1.

Objectives

✕ Do not copy. Read for understanding and the viva

- Complete questions 6 to 8 of the manual: sequence diagrams
- Prepare the deliverable before the lab and finish it during the session
- Be ready to explain every step in the viva

Questions Covered

✕ Do not copy. Read for understanding and the viva

Question	Requirement	Status
Q6	Draw Sequence Diagram for Online Shopping System	Complete
Q7	Draw Sequence Diagram for Online Examination System	Complete
Q8	Draw a Sequence diagram for Employee Management System	Complete

Preparation

✕ Do not copy. Read for understanding and the viva

- The manual asks for a problem description of 300 to 500 words and a list of assumptions before every diagram. Write both first; they fix the scope the evaluator marks you against.
- Pick one main-flow scenario per diagram and write its steps as a numbered list first.
- Lifelines are objects, not classes: name them like `cart : ShoppingCart`.

- Show activation bars and return messages; loop and alt fragments only where the scenario branches.

Question 6

Problem Statement

■ Write in lab record

Draw Sequence Diagram for Online Shopping System.

Solution

■ Write in lab record

Assumptions

This diagram traces the use case Place Order from Session 2 through the classes of the Session 1 class diagram. The scenario starts when a logged-in customer with a non-empty cart presses Checkout and ends when the order is confirmed with an order number, or when the payment is declined.

The system first asks the cart to total itself. The cart walks its cart items; each item asks its product for the current price and returns quantity times price. The cart then checks, item by item, that the product still has enough stock; if any product is out of stock the cart returns an error and the scenario stops before an order is created. If everything is available, the cart creates an Order object, copying each cart item into an order line with the frozen unit price, and attaching the delivery address the customer picked. The order is returned to the customer in status Placed with the total shown.

The customer now chooses a payment mode and confirms. The order creates a Payment object for the total and asks it to process. The payment sends an authorisation request with the amount and the tokenised card or UPI details to the external Payment Gateway, which replies approved or declined. On approval the payment marks itself Success, the order changes its status to Paid, reduces the stock of every product in its lines, and returns the order id and status to the customer. On decline the payment is marked Failed, the order stays Placed with a note Payment Failed, and the customer is told to retry; stock is not touched. Cash on delivery follows the approved branch with a gateway reply that is generated locally.

The lifelines are the actor Customer, `cart : ShoppingCart`, `item : CartItem`, `p : Product`, `order : Order`, `pay : Payment`, and the external actor Payment Gateway. Shipment does not appear because it is created later in the Ship Order use case by the administrator. Nothing in the diagram invents a class that is absent from Session 1.

Assumptions:

- The customer is already logged in and the cart holds at least one item; Login and Manage Cart are separate use cases.
- The delivery address is chosen before checkout and passed as a parameter; address management is not shown.
- One loop fragment covers the totalling and stock check together to keep the diagram short; the two loops can be drawn separately.
- The gateway responds synchronously; timeouts are treated as declined.
- Stock is reduced only in the approved branch.

Steps

1. Model, Add Diagram, Sequence Diagram; name it `PlaceOrder`.
2. Toolbox Lifeline: place Customer (set stereotype actor in the Editor pane, or use Toolbox Actor Lifeline), then the five object lifelines left to right in the order they first receive a message. In the Editor set Name to `cart` and Type to `ShoppingCart`; StarUML shows `cart: ShoppingCart`.
3. Toolbox Message: click the sending lifeline, drag to the receiver, type the message text. Set Message Sort to synchCall for calls and reply for returns (dashed). For object creation use Toolbox Create Message; the receiver's head drops to the creation point.
4. Toolbox Combined Fragment: draw the loop around messages 1.1 to 1.2 and set Interaction Operator to loop with guard `[each item]`; draw the alt around 2.3 to 2.5 with operands `[approved]` and `[declined]`.
5. Activation bars appear automatically; drag their ends to match the call and return.
6. Save and export as PNG.

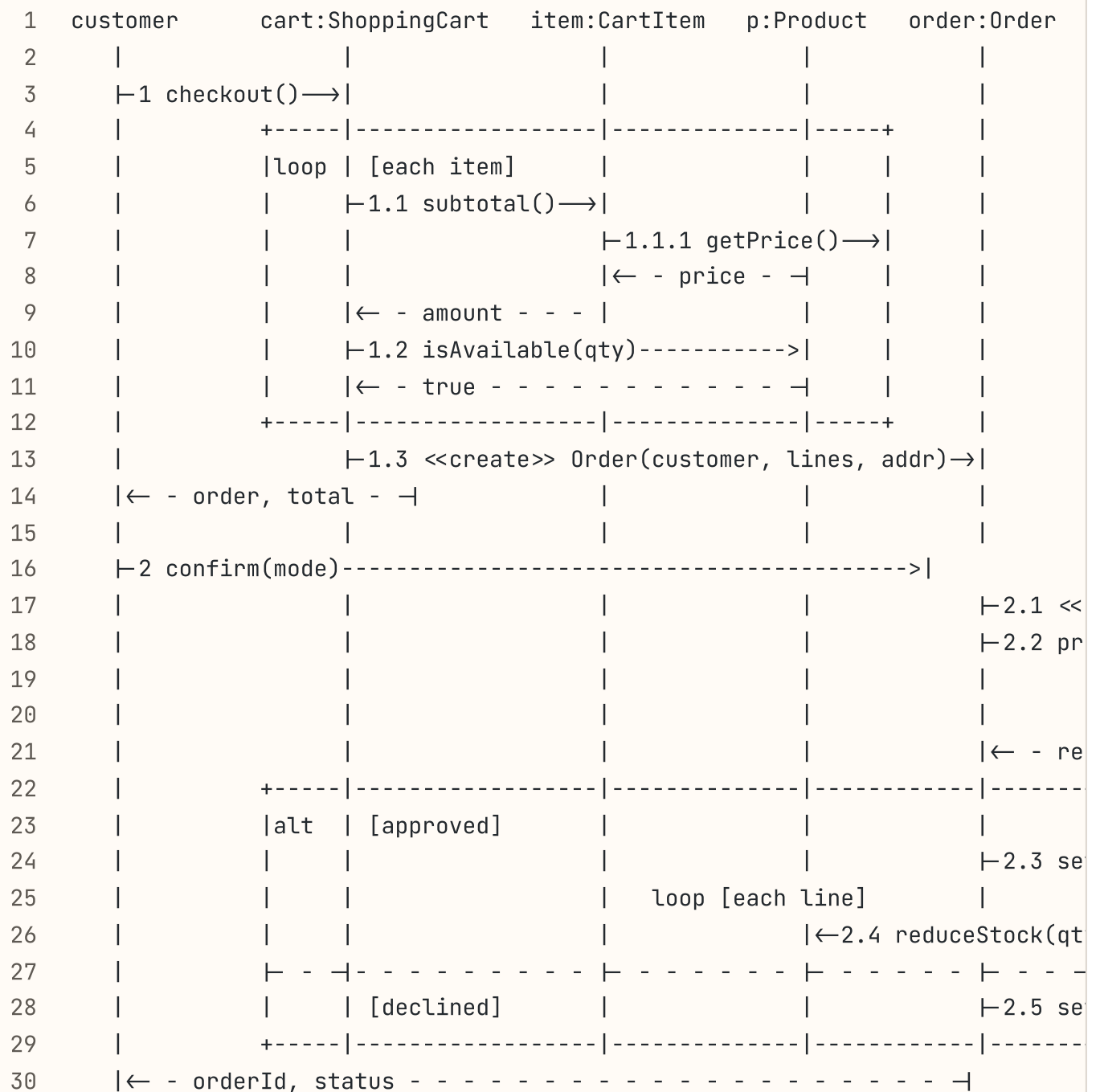
Diagram

Lifeline	Class	Role in Place Order
customer	actor Customer	Starts checkout, chooses payment mode, receives confirmation
cart	ShoppingCart	Totals items, checks stock, creates the order
item	CartItem	Returns its subtotal
p	Product	Gives price and availability, reduces stock
order	Order	Freezes lines, owns payment, changes status
pay	Payment	Talks to the gateway, records the result
gateway	actor Payment Gateway	Approves or declines the authorisation

No	From	To	Message	Returns
1	customer	cart	checkout(address)	order, total
1.1	cart	item	subtotal() (loop, each item)	amount
1.1.1	item	p	getPrice()	price
1.2	cart	p	isAvailable(qty) (same loop)	true or false
1.3	cart	order	create Order(customer, lines, address)	order
2	customer	order	confirm(mode)	orderId, status
2.1	order	pay	create Payment(total, mode)	pay
2.2	order	pay	process()	approved or declined
2.2.1	pay	gateway	authorize(amount, details)	approved or declined
2.3	order	order	setStatus(Paid) (alt approved)	none
2.4	order	p	reduceStock(qty) (loop, each line)	none
2.5	order	order	setStatus(PaymentFailed) (alt declined)	none

■ Lab record: every tab is one file of the answer. Write all of them.

Sketch



Explanation

The diagram reads top to bottom as time. Every solid arrow is a synchronous call and has a dashed return under it, which is what lets the reader see that the customer waits for the gateway. The loop fragment says the same two messages repeat for each item; the alt fragment splits the flow after the gateway answers and its guard conditions are mutually exclusive. The two create messages point at the head of the lifeline because those objects do not exist before that instant, matching the Session 1 rule that Order composes OrderLine and owns one Payment. Every message here is an operation that already exists on the Session 1 class, and every pair of lifelines

that exchange a message is connected by an association there, which is the consistency check the evaluator applies.

Question 7

Problem Statement

■ Write in lab record

Draw Sequence Diagram for Online Examination System.

Solution

■ Write in lab record

Assumptions

This diagram traces the use case Take Exam through the Session 1 class diagram of the Online Examination System. It begins when an enrolled student, already logged in, opens a scheduled exam at its start time, and ends when the attempt has been evaluated and a result object exists waiting for the examiner to publish it.

The student selects the exam and presses Start. The exam object checks that the current time lies inside its window (start time to start time plus duration) and that this student has not already attempted it. If both hold, the exam creates an Attempt for the student, which records its start time and status In Progress. The attempt returns the list of questions to the student; each question is shown with its options.

The student then answers questions one at a time. Every selection is sent to the attempt as answer(question, option). The attempt creates an Answer object holding the chosen option for that question; if an answer for that question already exists it is replaced. This loop runs until the student presses Submit or the server timer expires. In the second case the exam calls submit on the attempt itself; the diagram shows this as an alternative entry to the same message.

On submit the attempt records its end time and evaluates itself. For every stored answer it asks the question whether the selected option is correct and adds the question's marks on a correct answer. It then creates a Result with marks obtained, percentage against the exam's total marks, and Pass or Fail against the exam's pass marks. The result is created with published set to false.

The attempt returns a message to the student that the exam has been submitted and the result will be available after publication. Publishing is a separate use case owned by the examiner and is not drawn.

The lifelines are the actor Student, `exam : Exam`, `attempt : Attempt`, `ans : Answer`, `q : Question` and `result : Result`. Option objects are passed as parameters and do not need a lifeline.

Assumptions:

- Login and enrolment checks are complete before the scenario starts.
- One attempt per student per exam; the check for an existing attempt is a self-message on Exam.
- Answers can be changed until submission; the diagram shows the create case only.
- Marks are whole numbers and there is no negative marking.
- The server clock drives the timer; the client timer is only a display.

Steps

1. Model, Add Diagram, Sequence Diagram; name it `TakeExam`.
2. Add the Student actor lifeline and five object lifelines with Name and Type set in the Editor pane (`exam` and `Exam`, and so on).
3. Draw messages 1 to 3.3 with Toolbox Message. Use Self Message for 1.1, 3.1 and the evaluate step; use Create Message for 1.2, 2.1 and 3.2.
4. Draw a Combined Fragment loop around message 2 with guard `[until submit or timer expiry]`, and a loop inside 3.1 with guard `[each answer]`.
5. Draw a Combined Fragment alt with operands `[student submits]` and `[timer expires]` around message 3, showing the two possible senders.
6. Add return messages (Message Sort reply) for every call that returns data. Save and export as PNG.

Diagram

Lifeline	Class	Role in Take Exam
student	actor Student	Starts the exam, answers, submits
exam	Exam	Checks the window, creates the attempt, fires the timer
attempt	Attempt	Stores answers, evaluates, creates the result
ans	Answer	Remembers the option chosen for one question
q	Question	Says whether an option is correct and how many marks it carries
result	Result	Holds marks, percentage and pass or fail status

No	From	To	Message	Returns
1	student	exam	start(studentId)	question list
1.1	exam	exam	isOpen() and noAttemptYet(studentId)	true
1.2	exam	attempt	create Attempt(student, exam)	attempt
1.2.1	attempt	attempt	begin() sets startTime, status In Progress	none
2	student	attempt	answer(q, o) (loop until submit or timer)	saved
2.1	attempt	ans	create Answer(q, o)	ans
3	student or exam	attempt	submit() (alt student submits or timer expires)	acknowledgement
3.1	attempt	attempt	evaluate() sets endTime	marks
3.1.1	attempt	q	isCorrect(ans.option) (loop each answer)	true or false
3.1.2	attempt	q	getMarks() (when correct)	marks
3.2	attempt	result	create Result(marks, percentage, status)	result
3.3	attempt	student	submitted, result pending publication	none

■ Lab record: every tab is one file of the answer. Write all of them.

Sketch

adding a Timer class that Session 1 does not have. Result is created inside the attempt's activation so the diagram agrees with the 1 to 0..1 multiplicity between Attempt and Result.

Question 8

Problem Statement

■ Write in lab record

Draw a Sequence diagram for Employee Management System.

Solution

■ Write in lab record

Assumptions

The Employee Management System keeps the records of a company's staff. Each Employee has an employee id, name, designation, date of joining and salary, and belongs to one Department. Each department has one head, a Manager, who is also an employee. The system stores daily Attendance, runs monthly Payroll, and handles Leave. Every employee has a LeaveBalance per leave type (casual, earned, sick) that is credited at the start of the year. A LeaveRequest records the dates, type, reason and status of one application: Pending, Approved or Rejected.

This diagram traces the use case Apply for Leave, the most common interaction in the system and the one that involves the most objects. The scenario starts when an employee fills in the leave form and ends when the manager's decision has been recorded and the balance updated.

The employee submits the request with the from date, to date and leave type. The employee object asks its leave balance whether the requested number of days is available for that type. If not, the request is refused at once and the employee is told the shortfall. If the balance is sufficient, the employee object creates a LeaveRequest in status Pending and forwards it to the manager of its department, then returns the request id to the employee as an acknowledgement.

Later, the manager opens the pending request and approves or rejects it. The manager object sets the status on the request. The request then notifies the employee object of the outcome. On approval the employee object deducts the days from its leave balance; on rejection the balance is

untouched. The manager receives a confirmation that the decision has been recorded. Payroll, attendance and department transfers are separate use cases and do not appear.

The lifelines are the actor Employee, `emp : Employee`, `bal : LeaveBalance`, `req : LeaveRequest`, `mgr : Manager` and the actor Manager. The message numbers here (1, 1.1 to 1.4, 2, 2.1 to 2.3) are reused unchanged in the collaboration diagram of Session 4 Question 11.

Assumptions:

- One approval level: the department manager decides; HR is not in the loop.
- Balance is checked at application time and deducted only on approval; weekends inside the range count as leave days.
- The manager is reached through the employee's department; the link `emp` to `mgr` exists in the class model as reports to.
- The two actors act at different times; the diagram shows this as two top-level messages 1 and 2.
- Notification is a message to the employee object; e-mail delivery is outside the diagram.

Steps

1. Model, Add Diagram, Sequence Diagram; name it `ApplyForLeave`.
2. Add the Employee actor lifeline on the far left and the Manager actor lifeline on the far right; place `emp : Employee`, `bal : LeaveBalance`, `req : LeaveRequest` and `mgr : Manager` between them.
3. Draw messages 1 to 1.4 and 2 to 2.3 with Toolbox Message; use Create Message for 1.2.
4. Draw a Combined Fragment alt around 1.2 to 1.4 with operands `[balance sufficient]` and `[insufficient]`, and a second alt around 2.2.1 with operands `[approved]` and `[rejected]`.
5. Add reply messages for 1.1, 1.4 and 2.3. Save and export as PNG.

Diagram

Lifeline	Class	Role in Apply for Leave		
employee	actor Employee	Applies for leave, receives the acknowledgement and outcome		
emp	Employee	Checks balance, creates the request, forwards it, updates the balance		
bal	LeaveBalance	Knows the days left per leave type		
req	LeaveRequest	Holds dates, type and status; notifies the employee		
mgr	Manager	Receives the pending request and records the decision		
manager	actor Manager	Approves or rejects		

No	From	To	Message	Returns
1	employee	emp	applyLeave(from, to, type)	requestId or refusal
1.1	emp	bal	hasBalance(type, days)	true or false
1.2	emp	req	create LeaveRequest(from, to, type) (alt sufficient)	req
1.3	emp	mgr	forward(req)	none
1.4	emp	employee	requestId (reply)	none
2	manager	mgr	approve(requestId) or reject(requestId)	confirmation
2.1	mgr	req	setStatus(Approved or Rejected)	none
2.2	req	emp	notify(status)	none
2.2.1	emp	bal	deduct(type, days) (alt approved)	none
2.3	mgr	manager	decision recorded (reply)	none

■ Lab record: every tab is one file of the answer. Write all of them.

Sketch

```

1  employee      emp:Employee      bal:LeaveBalance      req:LeaveRequest      mgr:Ma
2      |          |                  |                  |                  |
3      |←1 applyLeave(from,to,type)→| |                  |                  |
4      |          |←1.1 hasBalance(type, days)→| |      |                  |
5      |          |← - true - - - - | |                  |                  |
6  +--|-----|-----|-----|-----|-----|-----|
7  |alt [balance sufficient]          | |                  |                  |
8  | |          |←1.2 <<create>> LeaveRequest(from,to,type)→| |      |                  |
9  | |          |←1.3 forward(req)----->| |      |                  |
10 | |← - 1.4 requestId - - - - | |      |                  |
11 | - - - - - | - - - - - | - - - - - | - - - - - |
12 | [insufficient]          | |                  |                  |
13 | |← - refused: shortfall - - - - | |      |                  |
14 +--|-----|-----|-----|-----|-----|-----|
15 |          |          |          |          |          |
16 |          |          |          |          |          |
17 |          |          |          |          |←2.1 setStatus(Ap
18 |          |←2.2 notify(status)-----|          |
19 +--|-----|-----|-----|-----|-----|-----|
20 |alt [approved] |          |          |          |          |
21 | |          |←2.2.1 deduct(type, days)→| |      |          |
22 | - - - - - | - - - - - | - - - - - | - - - - - |
23 | [rejected] | (no change) |          |          |          |
24 +--|-----|-----|-----|-----|-----|-----|
25 |          |          |          |          |          |

```

Explanation

Two actors on one sequence diagram is normal when a use case has a hand-off; the gap between message 1.4 and message 2 is real time (hours or days) and needs no special notation. The balance check comes before the request is created so that a refused application leaves no object behind, which is why 1.2 sits inside the alt. The request notifies the employee object and the employee object updates its own balance, rather than the manager reaching into the balance directly; this keeps every message on an existing association (Employee owns LeaveBalance, Employee creates LeaveRequest, Employee reports to Manager) and makes the Session 4 collaboration diagram drawable without new links. The numbering is hierarchical from the start so that the collaboration diagram uses exactly the same labels.

Viva Questions

× Do not copy. Read for understanding and the viva

Q: What does the vertical axis of a sequence diagram represent? **A:** Time, increasing downwards. The horizontal axis carries the lifelines and has no meaning beyond grouping.

Q: What is an activation bar? **A:** The thin rectangle on a lifeline showing the period during which the object is executing an operation. It starts at the incoming call and ends at the return.

Q: Difference between a synchronous and an asynchronous message? **A:** A synchronous call (filled arrowhead) makes the sender wait for the reply. An asynchronous message (open arrowhead) lets the sender continue. Returns are dashed lines.

Q: Why does a create message point at the lifeline head? **A:** Because the object does not exist before that message; its lifeline starts at the point of creation, lower than the others.

Q: When do you use a loop or alt fragment? **A:** Loop when the same messages repeat under a guard; alt when the flow branches on mutually exclusive guards. Do not add fragments for a single straight-line scenario.

Q: How does a sequence diagram relate to the class diagram? **A:** Every lifeline is an object of a class in the class diagram, every message is an operation on the receiver's class, and every pair of communicating objects must be linked by an association.

Q: Why is the Payment Gateway an actor and not an object? **A:** It is outside the system boundary; the system does not implement it, it only sends it messages. Actors can appear on sequence diagrams as lifelines with the stick-figure symbol.

Q: What is the difference between a sequence and a collaboration diagram? **A:** Same messages, different emphasis: the sequence diagram shows time order by position, the collaboration diagram shows the object links and numbers the messages to give the order.

Common Mistakes

× Do not copy. Read for understanding and the viva

- Naming lifelines with class names only (`ShoppingCart`) instead of objects (`cart : ShoppingCart`).

- Sending a message to an object that has no association with the sender in the class diagram; either add the association in Session 1 or route the message differently.
- Drawing a message to an object before its create message.
- Leaving out return messages, so the reader cannot tell what the customer sees or when the activation ends.
- Modelling one use case per lifeline rather than one use case per diagram; a sequence diagram traces exactly one scenario.
- Inventing new classes (Timer, EmailService, Database) on the sequence diagram that were not in the class diagram and then forgetting to go back and add them.

Session Summary

■ Write in lab record

- Question 6: Place Order sequence diagram for Online Shopping, seven lifelines, twelve messages, one loop and one alt fragment
- Question 7: Take Exam sequence diagram for Online Examination, six lifelines, twelve messages, two loops and one alt fragment
- Question 8: Apply for Leave sequence diagram for Employee Management, six lifelines including two actors, ten messages, two alt fragments, numbered for reuse in Session 4
- Lifeline and message tables, ASCII sketch and PlantUML file for each diagram; every message maps to a Session 1 operation

 [Edit this page](#)

[< Previous](#)
Session 2

Session 4 [Next >](#)