

Session 5

Activity diagrams

Activity diagrams model workflow: the order of actions, decisions, and parallel work. They are the right diagram for processes that span several use cases, such as placing and paying for an order.

Objectives

✕ Do not copy. Read for understanding and the viva

- Complete questions 12 to 13 of the manual: activity 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
Q12	Activity Diagram Online Banking System	Complete
Q13	Draw Activity Diagram for Online Examination 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.
- Use swimlanes for each actor or subsystem so responsibility is visible.
- Every decision node needs guard conditions on all outgoing edges that together cover every case.

- Fork and join must pair up; an unmatched fork is the most common marking error.

Question 12

Problem Statement

■ Write in lab record

Activity Diagram Online Banking System.

Solution

■ Write in lab record

Assumptions

The Online Banking System lets a registered customer of a bank use banking services through a web browser without visiting a branch. The customer logs in with a user ID and password issued by the bank. After a successful login the system shows a dashboard from which the customer can view the balance of a linked account or transfer money to a registered beneficiary. A fund transfer needs a second factor: the bank sends a one-time password (OTP) to the customer's registered mobile number, and the transfer proceeds only when the OTP is entered correctly within its validity period. The core banking system holds the accounts. It checks the available balance, debits the sender, credits the beneficiary and records the transaction. After a successful transfer the system sends an SMS alert, sends an email receipt and updates the transaction history. These three actions do not depend on each other, so they run in parallel. The customer can perform several services in one session and ends the session by logging out.

The activity diagram models this whole session as one workflow. It is the right diagram here because the interesting part of the problem is the order of actions, the points where the flow branches (login check, balance check, OTP check) and the point where it splits into parallel work (notifications), not the structure of the classes. Three swimlanes show who is responsible for each action: the Customer, the Web Server that runs the net banking application, and the Core Banking system that owns the accounts. Every decision node carries guards on all of its outgoing edges, and the guards on one decision together cover every possible case. The fork and join around the notifications pair up. The flow ends at a final node after logout, and also after a failed login, because those are the two ways a session can end. A wrong OTP or an insufficient balance

does not end the session; the flow returns to the dashboard through a merge node so the customer can try another service.

The diagram is deliberately limited to two services. Bill payment, cheque book request and mini statement follow the same shape as the two shown and would only add width to the drawing. Beneficiary registration and account locking are separate use cases with their own workflows.

- The customer is already registered with the bank and has a user ID, password and registered mobile number.
- The beneficiary was added and activated earlier; adding a beneficiary is a separate use case and is not shown.
- Three wrong passwords lock the account through a separate process; the diagram shows one login attempt with a final node on failure.
- The OTP is valid for 3 minutes and is verified by the web server, not by the customer's bank branch.
- The balance check also enforces the per-transaction limit: the guard "balance sufficient" means the amount is at most the available balance and at most the daily limit.
- Debit and credit run inside one database transaction in core banking, so the diagram shows them as two consecutive actions with no failure between them.
- SMS, email and history update are independent and are shown between a fork and a join.
- Only view balance and fund transfer are modelled; the other services are analogous.

Steps

1. Open StarUML, press Ctrl+N for a new project and choose Model, Add Diagram, Activity Diagram. Name it "Online Banking Activity".
2. In the Toolbox choose Swimlane (Vertical) and click on the canvas three times. Select each lane and set Name in the Editor to Customer, Web Server and Core Banking.
3. Choose Initial Node and click in the Customer lane. Choose Action and place every action from the table below in its lane; double-click each one to type its name.
4. Choose Decision Node for the four decisions (credentials valid, service, balance sufficient, OTP correct) and Merge Node where the service branches rejoin before "another service".
5. Choose Fork Node and Join Node for the notification bar. Draw three Control Flows out of the fork and three into the join.
6. Choose Control Flow and connect the nodes in order. For every flow that leaves a decision, select the flow and type the guard in the Editor field guard (StarUML draws it in square brackets).

7. Place an Activity Final node after End session and another after Show login error.
8. Check the diagram against the PlantUML below, then export it with File, Export Diagram As, PNG for the lab record.

Diagram

Element	Kind	Swimlane	Notes
Initial node	Initial	Customer	Session starts when the customer opens the site
Open net banking site, Enter user ID and password, Choose service, Enter beneficiary amount and remarks, Enter OTP, Logout	Action	Customer	Actions performed by the person
Validate credentials, Show dashboard, Show login error, Display balance, Send OTP, Check OTP, Show OTP error, Show insufficient balance, Show transfer confirmation, Send SMS alert, Send email receipt, Update transaction history, End session	Action	Web Server	Application logic
Read account balance, Check available balance, Debit sender account, Credit beneficiary account, Record transaction	Action	Core Banking	Account owner
credentials valid	Decision	Web Server	Guards [yes] and [no]
service	Decision	Customer	Guards [view balance] and [fund transfer]
balance sufficient	Decision	Core Banking	Guards [yes] (amount at most balance and daily limit) and [no]
OTP correct	Decision	Web Server	Guards [yes] (matches and within 3 minutes) and [no]

Element	Kind	Swimlane	Notes
another service	Decision	Customer	Guards [yes] loops to Choose service, [no] goes to Logout
Merge node	Merge	Web Server	Joins the four service outcomes before "another service"
Fork and join	Fork, Join	Web Server	Three parallel branches: SMS, email, history
Final nodes	Activity final	Web Server	One after End session, one after Show login error

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

Sketch

```

1  Customer                      | Web Server                      | Core Banking
2  -----+-----+-----
3  (*)                           |                                 |
4  |                             |                                 |
5  [Open net banking site]      |                                 |
6  |                             |                                 |
7  [Enter user ID, password]---->[Validate credentials] |
8  |                             |                                 |
9  |         <credentials valid?> |                                 |
10 |         [no]|         |[yes] |                                 |
11 | [Show login |         |                                 |
12 | error]      v         |                                 |
13 |         |         [Show dashboard] |                                 |
14 |         (X)         |                                 |
15 +<-----+-----+-----+
16 |                             |                                 |
17 [Choose service]<-----+ [yes] |
18 |                             |         |                                 |
19 <service?>         |         |                                 |
20 |[view balance]----->[Read account ba
21 |                             | [Display balance]<-----+
22 |[fund transfer] |         |         |                                 |
23 [Enter beneficiary, |         |         |                                 |
24 amount, remarks]----->[Check available
25 |         |         |         |         |                                 |
26 |         |         |         |         |         <balance suff
27 | [Show insufficient |         |         |         |         [no]|         |
28 | balance]<-----+
29 |         |         |         |         |                                 |
30 | [Send OTP]<-----+
31 [Enter OTP]<-----+         |         |         |                                 |
32 +----->[Check OTP]         |         |         |                                 |
33 |         |         |         |         |                                 |
34 |         <OTP correct?> |         |         |                                 |
35 |         [no]|         |[yes] |         |                                 |
36 | [Show OTP |         |         |         |                                 |
37 | error]      +----->[Debit sender a
38 |         |         |         |         |         |                                 |
39 |         |         |         |         |         |         [Credit benefici
40 |         |         |         |         |         |         |                                 |
41 |         |         |         |         |         |         [Record transact
42 |         |         |         |         |         |         |                                 |
43 |         |         |         |         |         |         |         ===== fork <-----+

```

```

44          |      | [Send SMS] [Send email] [Update history]
45          |      | ===== join
46          |      |           |           |
47          | [Show transfer confirmation] |
48          |      |           |           |
49          |      (merge) ← all four service outcomes
50          |      |           |           |
51  <another service?>-----+           |
52  |[no]                   |           |
53  [Logout]----->[End session]       |
54          |      |           |           |
55          |      (X)           |           |

```

Explanation

The diagram reads top to bottom, one action per rounded box, and each box sits in the lane of the party that performs it. The four decisions are the places where the workflow can go two ways, and each decision has exactly two guards that are mutually exclusive and together exhaustive: valid or not, sufficient or not, correct or not, another service or not. The `service` decision has two guards because only two services are modelled; adding a third service means adding a third guard, not a second decision. The fork bar after Record transaction says that SMS, email and history update can happen in any order or at the same time; the join bar says that the confirmation screen waits for all three. The `repeat` block in the PlantUML source is the merge node plus the “another service” decision: every service outcome, successful or not, flows back to the same point. Two final nodes are correct because a failed login and a logout are both legitimate ends of the session.

Question 13

Problem Statement

■ Write in lab record

Draw Activity Diagram for Online Examination System.

Solution

■ Write in lab record

Assumptions

The Online Examination System conducts university examinations over the web. A student logs in with an enrolment number and password. The exam server checks the credentials and lists the examinations scheduled for that student. The student selects one. The server verifies that the examination window is open and that the student has not already attempted it, then creates an attempt, loads the question paper and starts a countdown timer. The paper contains objective questions (multiple choice, marked automatically) and descriptive questions (free text, marked by a human evaluator). While time remains the student answers questions and the server saves every answer as it is given, so that a lost connection does not lose work. The attempt ends when the student presses Submit or when the timer reaches zero, in which case the server submits the saved answers on the student's behalf. After submission the server locks the attempt so nothing can be changed. Evaluation then proceeds on two independent tracks: the server marks the objective answers against the answer key, and an evaluator reads and marks the descriptive answers. When both tracks finish, the server adds the marks, compares the total with the pass mark, records pass or fail and publishes the result. The student logs in later to view it.

The activity diagram follows one student through one examination from login to result. Three swimlanes hold the actions: the Student, the Exam Server (the application and its database together) and the Evaluator. The interesting control-flow features are the two guarded decisions that can end the flow early (invalid credentials, exam not available), the loop that repeats answer and save until time runs out or the student submits, the fork that lets automatic and human marking proceed in parallel, the join that waits for both before totals are computed, and the final pass or fail decision. As in Question 12, each decision has guards that cover all cases, and the fork is matched by a join.

The scope excludes exam creation by the faculty, question bank maintenance, re-evaluation requests and proctoring, which are separate workflows.

- One student attempts one examination at a time; a student may attempt a given examination only once.
- The examination window is a fixed date and time range set by the university; a student who arrives after it closes cannot start.
- Answers are saved on the server immediately after each question, so the loop body has one save action.
- When the timer reaches zero the server auto-submits whatever is saved; the loop guard "time left and not submitted" covers both exits.

- Objective questions are marked automatically from an answer key; descriptive questions are marked by one evaluator.
- The pass mark is a fixed number per paper; the guard “total at least pass marks” is evaluated once.
- Publishing the result means making it visible on the student’s dashboard and sending a notification; the notification is inside the Publish result action.

Steps

1. Choose Model, Add Diagram, Activity Diagram and name it “Online Examination Activity”.
2. Add three vertical swimlanes named Student, Exam Server and Evaluator.
3. Place the Initial Node in the Student lane and the actions from the table in their lanes.
4. Add Decision Nodes for credentials valid, exam available, time left and not submitted, and total at least pass marks. Add a Merge Node where the pass and fail branches rejoin before Publish result and one where the loop returns to Answer current question.
5. Add one Fork Node after Lock attempt and one Join Node before Compute total marks. Draw the objective branch inside the Exam Server lane and the descriptive branch through the Evaluator lane.
6. Draw Control Flows and type each guard in the Editor. Add Activity Final nodes after Show login error, after Show exam not available, and after View result.
7. Export the diagram as PNG.

Diagram

Element	Kind	Swimlane	Notes
Initial node	Initial	Student	Student opens the portal
Log in with enrolment number and password, Select exam, Answer current question, View result	Action	Student	Student actions
Validate credentials, Show login error, List scheduled exams, Create exam attempt, Load question paper and start timer, Show exam not available, Save answer, Lock attempt, Auto-evaluate objective answers, Compute total marks, Mark result PASS, Mark result FAIL, Publish result	Action	Exam Server	Application actions
Evaluate descriptive answers, Enter marks	Action	Evaluator	Human marking
credentials valid	Decision	Exam Server	[yes] and [no]
exam available	Decision	Exam Server	[yes] (window open and no previous attempt) and [no]
time left and not submitted	Decision	Student	[yes] loops back to Answer current question, [no] leaves the loop
total at least pass marks	Decision	Exam Server	[yes] PASS and [no] FAIL
Fork and join	Fork, Join	Exam Server	Objective branch and descriptive branch

Element	Kind	Swimlane	Notes
Merge nodes	Merge	Exam Server	Before Publish result and at the loop head
Final nodes	Activity final	Exam Server, Student	After login error, after exam not available, after View result

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

Sketch

```

1  Student                                | Exam Server                                | Eva
2  -----+-----+-----+-----+-----+
3  (*)                                  |                                  |
4  |                                  |                                  |
5  [Log in with enrolment             ---->[Validate credentials]      |
6  number and password]               |          |                      |
7  |          <credentials valid?>    |                                  |
8  |          [no]||                  |[yes]                          |
9  | [Show login          v            |                                  |
10 | error]      [List scheduled exams] |                                  |
11 |          |                      |                                  |
12 |          (X)                    |                                  |
13 [Select exam]<-----+-----+-----+
14 |                                  |                                  |
15 +-----> <exam available?>      |                                  |
16 |          [no]||                  |[yes]                          |
17 | [Show exam not    v            |                                  |
18 | available] [Create exam attempt] |                                  |
19 |          |                      |                                  |
20 |          (X)      [Load question paper, |                                  |
21 |                  start timer]         |                                  |
22 +<-----+-----+-----+
23 |                                  |                                  |
24 (merge)<-----+-----+ [yes]    |                                  |
25 |                                  |                                  |
26 [Answer current question]--->[Save answer] |                                  |
27 |          |                      |                                  |
28 <time left and not submitted?><----+ |                                  |
29 |[no]          |                      |                                  |
30 +----->[Lock attempt]           |                                  |
31 |          |                      |                                  |
32 | ===== fork =====          |                                  |
33 |          |                      |                                  |
34 | [Auto-evaluate          +----->[Evalu |
35 | objective answers]      |          |   ans
36 |          |                      |                                  |
37 |          |                      |                                  | [Ent
38 |          |                      +-----+
39 | ===== join =====          |                                  |
40 |          |                      |                                  |
41 | [Compute total marks]          |                                  |
42 |          |                      |                                  |
43 | <total at least pass marks?>   |                                  |

```

```

44          | [yes] |          | [no] |
45          | [Mark result] [Mark result] |
46          | PASS]      FAIL] |
47          | |          | |
48          | (merge)<-----+ |
49          | |          | |
50          | [Publish result] |
51 [View result]<-----+ |
52 |          |          |
53 (X)        |          |

```

Explanation

The two early decisions protect the rest of the flow: a student who fails login or picks an unavailable exam never reaches an attempt, so the remaining actions can assume a valid, open attempt. The answer loop is drawn as a merge node, two actions and a decision whose `[yes]` edge returns to the merge node; in PlantUML this is the `repeat` block. Its single guard, "time left and not submitted", is deliberately worded so that the two ways of leaving the loop (student submits, timer expires) both fall under `[no]`, which keeps the guards complete without a second decision. The fork after Lock attempt is the key modelling decision: automatic marking and human marking are independent, so neither should wait for the other, and only Compute total marks needs both, which is why the join sits immediately before it. The descriptive branch crosses into the Evaluator lane and returns, which shows the hand-off to a different actor without leaving the parallel region. The final pass or fail decision merges again before Publish result because publishing is the same action for either outcome.

Viva Questions

✖ Do not copy. Read for understanding and the viva

Q: What is the difference between a decision node and a fork node? **A:** A decision node chooses one outgoing edge using guards; exactly one branch runs. A fork node starts all outgoing edges; every branch runs, concurrently or in any order.

Q: Why must the guards on a decision be complete? **A:** If some case matches no guard the token has nowhere to go and the workflow deadlocks. Complete guards, such as `[yes]` and `[no]`, guarantee an edge for every case.

Q: What does a join node wait for? **A:** A token on every incoming edge. In Question 13 the join waits for both the automatic marking and the evaluator's marks before Compute total marks runs.

Q: What do swimlanes add to an activity diagram? **A:** Responsibility. Each action sits in the lane of the actor or subsystem that performs it, so hand-offs between customer, web server and core banking are visible.

Q: Why are there two final nodes in the banking diagram? **A:** A session can end in two ways: a failed login and a normal logout. Both are valid ends, so each has an activity final node.

Q: How is a loop shown in an activity diagram? **A:** With a merge node at the loop head, the loop body, and a decision whose one guard returns to the merge node and whose other guard leaves the loop. PlantUML writes this as `repeat ... repeat while`.

Q: Where would you show that debit and credit must both succeed or both fail? **A:** Not in the activity diagram, which shows order, but in the assumptions: they run inside one database transaction in core banking. The diagram shows them as consecutive actions.

Q: What is the difference between an activity diagram and a flowchart? **A:** An activity diagram adds swimlanes, fork and join for concurrency, object flows and signals. A flowchart has only sequential control flow with decisions.

Common Mistakes

✗ Do not copy. Read for understanding and the viva

- Leaving a decision with a single guard, or with guards such as `[balance sufficient]` and `[OTP valid]` that are not alternatives of each other.
- Drawing a fork without a matching join, or joining branches that came from a decision (that needs a merge, not a join).
- Putting an action in the wrong lane, for example Debit sender account in the Customer lane.
- Using verbs for lanes and nouns for actions; lanes are named after actors or subsystems, actions after verb phrases.
- Ending the flow with an arrow into nothing instead of an activity final node.
- Writing the 300 to 500 word description after the diagram; the description sets the scope and the evaluator marks against it.

Session Summary

■ Write in lab record

- Question 12: activity diagram for Online Banking with lanes Customer, Web Server, Core Banking; four guarded decisions, a three-branch fork and join, a service loop; PlantUML source `online_banking_activity.puml`
- Question 13: activity diagram for Online Examination with lanes Student, Exam Server, Evaluator; answer loop, fork for objective and descriptive marking, pass or fail decision; PlantUML source `online_examination_activity.puml`
- Problem description (300 to 500 words) and assumptions for every diagram, written before the drawing
- Element table and ASCII sketch for each diagram, ready to redraw in StarUML and export as PNG

[✎ Edit this page](#)

< Previous
Session 4

Next >
Session 6