SAD CT

1. Describe a pyramid-structured interview you would conduct for analyzing a Student Admission System.

A pyramid-structured interview for analyzing a Student Admission System is conducted by starting with open-ended questions to gather broad information and then gradually moving to more specific questions, including closed questions to confirm details.

First, an open-ended question like, “Can you describe the overall student admission process?” helps the interviewer understand the general workflow and context. Next, a slightly narrower open-ended question such as, “How are student applications submitted and recorded?” collects information about the channels and methods used. Then, a more specific open-ended question, for example, “What documents and information are required from applicants?” helps identify the exact data needed for processing.

After collecting broad and specific details, closed questions are used to confirm certain facts. For instance, “Do you check for duplicate applications before processing? (Yes/No)” ensures that exception handling practices are clarified. Finally, a detail-oriented closed question, such as, “Are reports generated for management after each admission cycle? (Yes/No),” confirms reporting requirements.

This approach follows a pyramid structure, moving from general understanding to specific requirements, ensuring that all important information for system analysis is captured effectively.

2. Describe a funnel-structured interview for a Hospital Management System and justify your order of

questions.

A **funnel-structured interview** for a Hospital Management System is conducted by starting with **broad, open-ended questions** and gradually narrowing down to **specific, closed questions**. This approach allows the interviewer to understand the general workflow before focusing on detailed system requirements.

Initially, an **open-ended question** like, “Can you describe the overall process of patient admission and care in the hospital?” helps gather a general understanding of hospital operations. Next, another **open-ended question**, such as, “How are patient records created and maintained?” collects information about the data flow and documentation practices.

After understanding the general processes, the interview moves to **more specific open-ended questions**, for example, “What types of reports do doctors and administrators require from the system?” This identifies detailed information needs. Finally, **closed questions** are asked to confirm exact requirements, such as, “Are patient records updated immediately after treatment? (Yes/No)” or “Does the system generate automatic notifications for scheduled appointments? (Yes/No).”

The order of questions is justified because the **funnel structure** starts broad to ensure no important area is missed and gradually narrows to specific, verifiable details. This ensures both a complete understanding of hospital operations and accurate identification of system requirements.

3.In what situation would you choose the diamond structure for an interview? Give one realistic example.

The **diamond-structured interview** is used when the interviewer wants to **start with broad questions, narrow down to specific questions, and then broaden again** to explore implications, feedback, or overall opinions. This structure is useful when understanding both the detailed processes and the bigger picture is important.

For example, in a **Hospital Management System**, the interview could begin with a broad **open-ended question** like, “Can you describe the overall workflow of patient admissions?” Then it would narrow down to specific questions, such as, “How are patient records entered and verified in the system?” After gathering these details, the interview would broaden again with questions like, “How do these procedures impact patient satisfaction and hospital efficiency?”

The diamond structure is chosen in situations where it is important to **analyze detailed procedures while also understanding their impact or implications** in the larger context.

3. Imagine your team is building a Mobile Banking App. Outline the roles of each participant in the JAD session.

In a **Joint Application Design (JAD) session** for building a Mobile Banking App, each participant has a clearly defined role to ensure effective collaboration and requirement gathering.

1. **JAD Facilitator:** The facilitator guides the session, ensures that discussions stay on track, and manages conflicts. They are responsible for keeping the meeting focused on objectives and encouraging participation from all stakeholders.
2. **System Analyst:** The system analyst documents requirements, asks probing questions to clarify needs, and translates business requirements into technical specifications for the development team.
3. **End Users / Customers:** These participants provide insights into their needs and expectations from the mobile banking app, such as features for fund transfers, bill payments, or account management. Their input ensures that the system meets actual user requirements.
4. **Developers / Technical Team:** The developers provide technical input, assess the feasibility of requirements, and suggest possible solutions or alternatives for implementing features in the app.

5. **Project Sponsor / Management:** This participant represents the business perspective, prioritizes requirements based on business goals, and ensures that the project aligns with strategic objectives.

4. Where should you conduct a JAD workshop for a large insurance company? Mention location, setup, and timing.

A **JAD workshop** for a large insurance company should be held in a **quiet and comfortable meeting room** that is big enough for everyone, including users, analysts, developers, and managers. The room should have **tables, chairs, a whiteboard or projector, and internet access** so everyone can work and share ideas easily.

The **setup** should let participants sit in a **U-shape or around a big table** so they can see each other and talk freely. Materials like **markers, sticky notes, and printed agendas** should be ready to help the discussion.

Timing is important. The workshop should be held during **normal working hours** when all key people are available. It can last **1–2 full days** or several half-day sessions. Make sure to include breaks so everyone stays focused.

5. Describe a JAD agenda for designing a University Admission Portal interface.

A **JAD (Joint Application Design) agenda** for designing a University Admission Portal should include a clear plan to make the session organized and productive.

1. **Introduction (15–20 minutes):** Welcome participants, explain the purpose of the workshop, and outline the goals. Introduce the team and clarify roles.

2. **Overview of Current Process (30–40 minutes):** Discuss how the current admission system works and identify problems or challenges.
3. **Requirements Gathering (1–2 hours):** Collect user needs and expectations for the portal. Use **open-ended questions** to understand what students, staff, and administrators want.
4. **Brainstorming & Interface Design Ideas (1–2 hours):** Encourage participants to suggest features, layouts, and functionality. Use sketches, mockups, or diagrams to visualize ideas.
5. **Prioritization of Requirements (30–45 minutes):** Decide which features are most important and should be implemented first.
6. **Wrap-up and Next Steps (15–20 minutes):** Summarize decisions, clarify any doubts, and outline what will happen after the session, like designing mockups or finalizing requirements.

List four benefits and two drawbacks of using JAD in developing an ERP system.

Benefits:

1. **Better Communication:** Users, developers, and managers discuss requirements together, reducing misunderstandings.
2. **Faster Requirement Gathering:** Important decisions are made during the session, saving time.
3. **User Involvement:** End users actively participate, so the system meets their real needs.
4. **Clear Documentation:** Decisions and requirements are recorded clearly for future reference.

Drawbacks:

1. **Time-Consuming:** Organizing all key participants for the session can take a lot of time.

2. **Costly:** Bringing everyone together and dedicating full days for the workshop can be expensive.

Suppose your JAD report was incomplete. What consequences could that have on the project outcome?

- **Missed Requirements:** Some important user needs or system features may not be included, leading to an incomplete system.
- **Misunderstandings:** Developers and stakeholders may interpret requirements differently, causing mistakes in design and development.
- **Delays:** Missing information can require extra meetings or rework, slowing down the project.
- **Increased Costs:** Fixing problems caused by incomplete requirements can be expensive.
- **Lower User Satisfaction:** The final system may not meet user expectations, reducing its usefulness.

Prepare a questionnaire (5 questions) to assess employee satisfaction with a new HR system.

- How easy is it for you to use the new HR system?

  (Options: Very Easy / Easy / Neutral / Difficult / Very Difficult)
- Does the HR system help you complete your tasks faster than before?

  (Options: Yes / No / Sometimes)
- Are you satisfied with the accuracy of the information provided by the HR system?

(Options: Very Satisfied / Satisfied / Neutral / Unsatisfied / Very Unsatisfied)

- How would you rate the support provided for any issues with the HR system?
  (Options: Excellent / Good / Average / Poor / Very Poor)
- What improvements would you suggest to make the HR system better?
  (Open-ended question for detailed feedback)

Write three nominal-scale and three interval-scale questions suitable for an e-commerce satisfaction survey.

Nominal-Scale Questions: *(categorical, no order implied)*

1. Which payment method do you use most often?
   - Credit Card / Debit Card / PayPal / Cash on Delivery
2. What type of device do you usually use to shop online?
   - Mobile / Tablet / Laptop / Desktop
3. Which category of products do you buy most frequently?
   - Electronics / Clothing / Books / Home & Kitchen

Interval-Scale Questions: *(numerical scale with equal intervals, can measure satisfaction or rating)*

1. On a scale of 1 to 5, how satisfied are you with the delivery speed?
   - 1 = Very Dissatisfied, 5 = Very Satisfied
2. On a scale of 1 to 10, how would you rate the website's ease of navigation?
   - 1 = Very Difficult, 10 = Very Easy
3. On a scale of 1 to 5, how likely are you to recommend this e-commerce site to a friend?
   - 1 = Not Likely, 5 = Very Likely

Define reliability and validity in the context of questionnaire design, with examples.

1. **Reliability:**

   Reliability refers to the **consistency of a questionnaire**. A reliable questionnaire produces similar results if it is repeated under the same conditions.

Example:

If you ask employees, “How satisfied are you with the HR system?” using the same scale, and they give similar answers when asked again after a week, the questionnaire is reliable.

1. **Validity:**

   Validity refers to the **accuracy of a questionnaire**, meaning it measures what it is intended to measure.

Example:

If the question “How satisfied are you with the HR system?” truly reflects employees’ satisfaction and not something else (like their workload), then the question is valid.

Prepare CRC cards for Student, Course, Instructor, and Registration classes in a University System.

1. Student

- **Class Name:** Student
- **Responsibilities:**
  - Store student personal details (name, ID, contact)
  - Enroll in courses

    - View grades
- **Collaborators:**
    - Course (to enroll)
    - Registration (to register courses)

2. Course

- **Class Name:** Course
- **Responsibilities:**
    - Store course details (name, code, credits)
    - List enrolled students
    - Assign instructors
- **Collaborators:**
    - Student (students enroll)
    - Instructor (assigns instructor)
    - Registration (register students)

3. Instructor

- **Class Name:** Instructor
- **Responsibilities:**
    - Store instructor details (name, ID, contact)
    - Teach assigned courses
    - Enter grades for students
- **Collaborators:**
    - Course (teaches courses)
    - Registration (to record grades)

4. Registration

- **Class Name:** Registration
- **Responsibilities:**

    - Record which students are enrolled in which courses
    - Track student grades
    - Maintain semester-wise course registration
- **Collaborators:**
    - Student (registers student)
    - Course (registers course)
    - Instructor (records grades)

Explain how UML packages are used for system partitioning and modular design. Give one diagrammatic example.

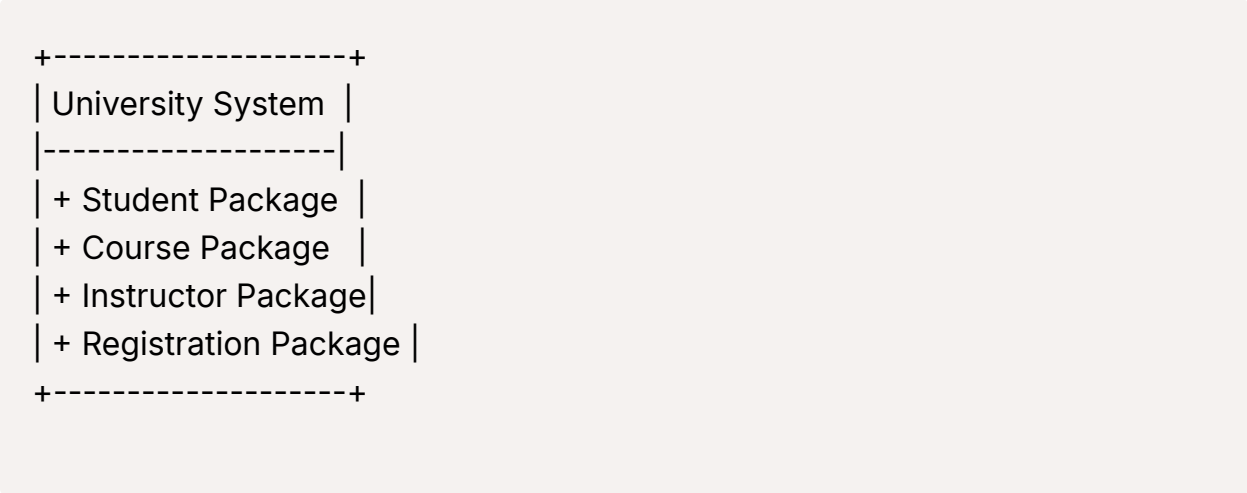
UML Packages:

A **UML package** is a way to **group related classes, interfaces, or components** in a system. Packages help **organize large systems** by dividing them into smaller, manageable parts. This makes the system easier to understand, maintain, and reuse.

How they help in system partitioning and modular design:

1. **Partitioning:** Packages divide a system into logical groups based on functionality or domain.
    - Example: In a University System, you can have separate packages for `Student Management`, `Course Management`, and `HR`.
2. **Modular Design:** Each package can be developed, tested, and maintained independently, reducing complexity.
3. **Reusability:** Packages can be reused in other projects or systems.
4. **Clear Dependencies:** Packages show which parts of the system depend on each other.

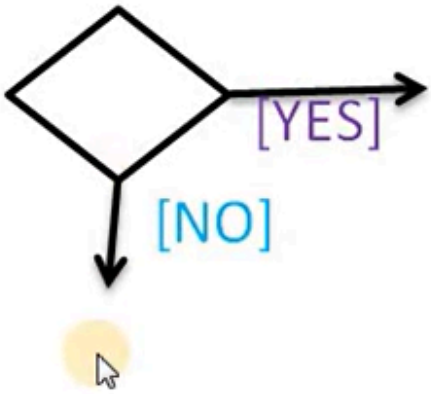
Diagrammatic Example:



- Here, the **University System** is divided into **four packages**: Student, Course, Instructor, and Registration. Each package contains classes related to its domain.

START

Activity

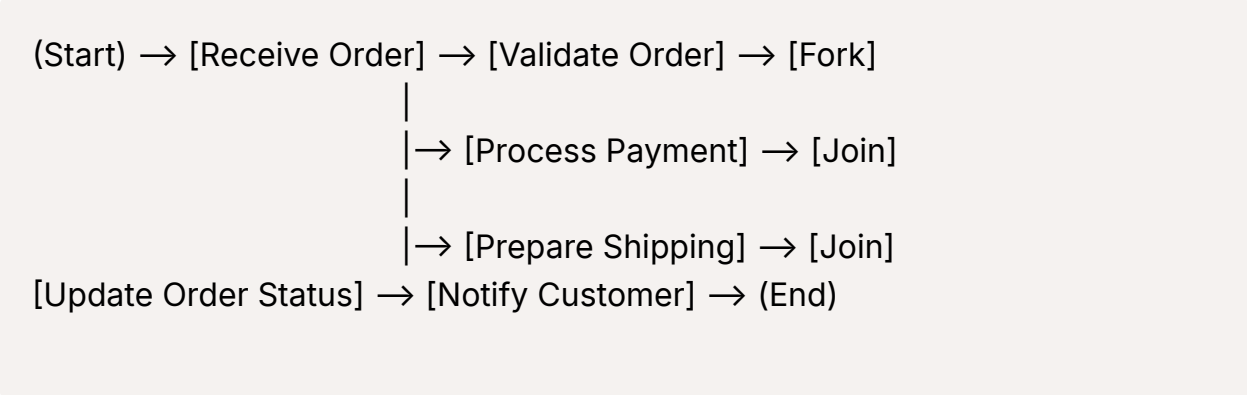


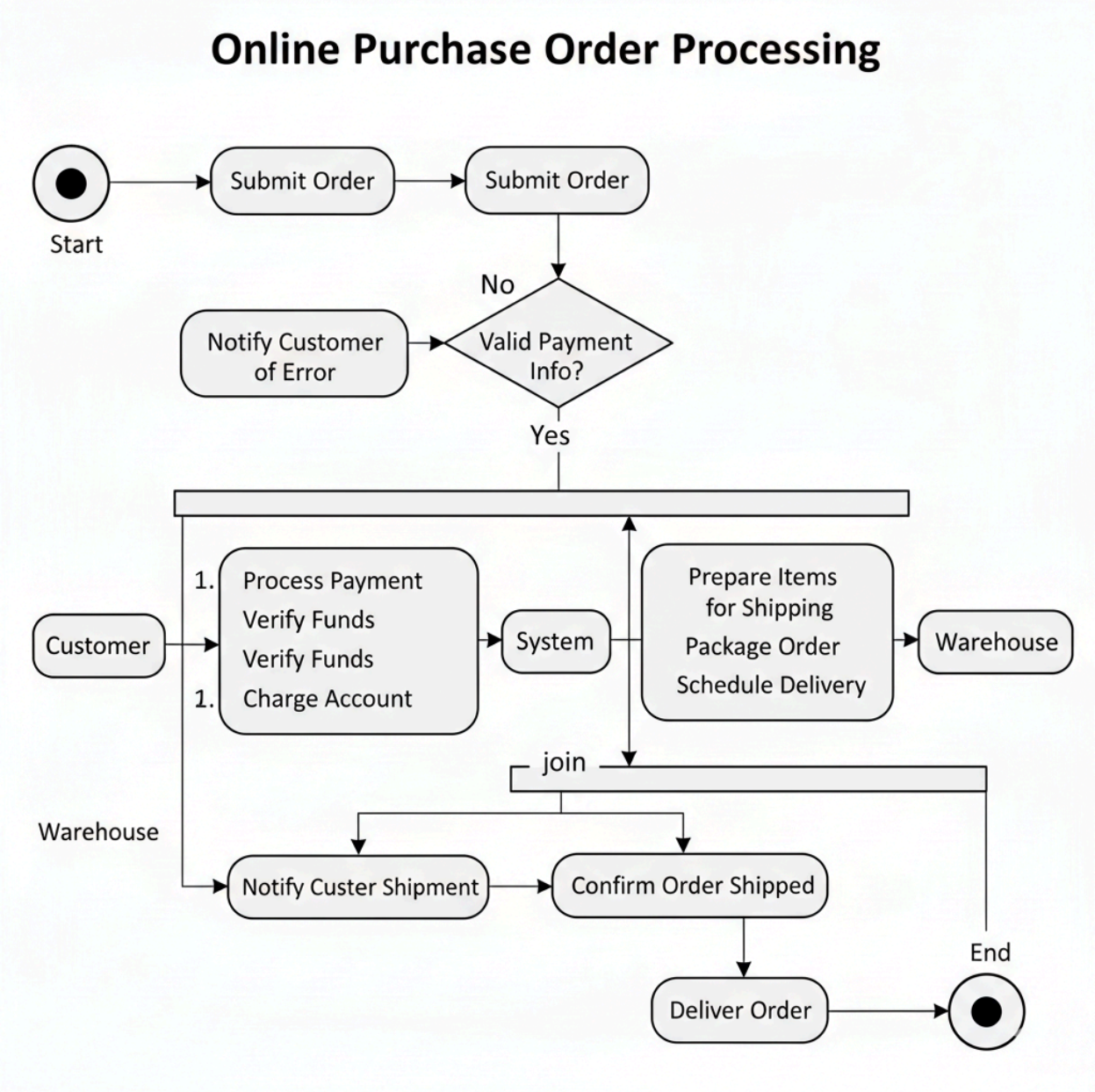
Condition

Parallel activity

END

Draw an activity diagram for processing an online purchase order including parallel payment and shipping activities.





Explanation:

1. **Start:** The process begins when an order is received.
2. **Receive Order:** System receives the purchase order.
3. **Validate Order:** Check if items are in stock and order details are correct.
4. **Fork:** Payment and shipping are processed **in parallel**.

- **Process Payment:** Handle payment confirmation.
- **Prepare Shipping:** Pack items and arrange delivery.

5. **Join:** Both parallel activities must complete before continuing.
6. **Update Order Status:** System updates order as completed.
7. **Notify Customer:** Customer is informed that order is processed.
8. **End:** Process ends.

Explain the use of swimlanes in an activity diagram with an example of a Hospital Billing process.

Swimlanes:

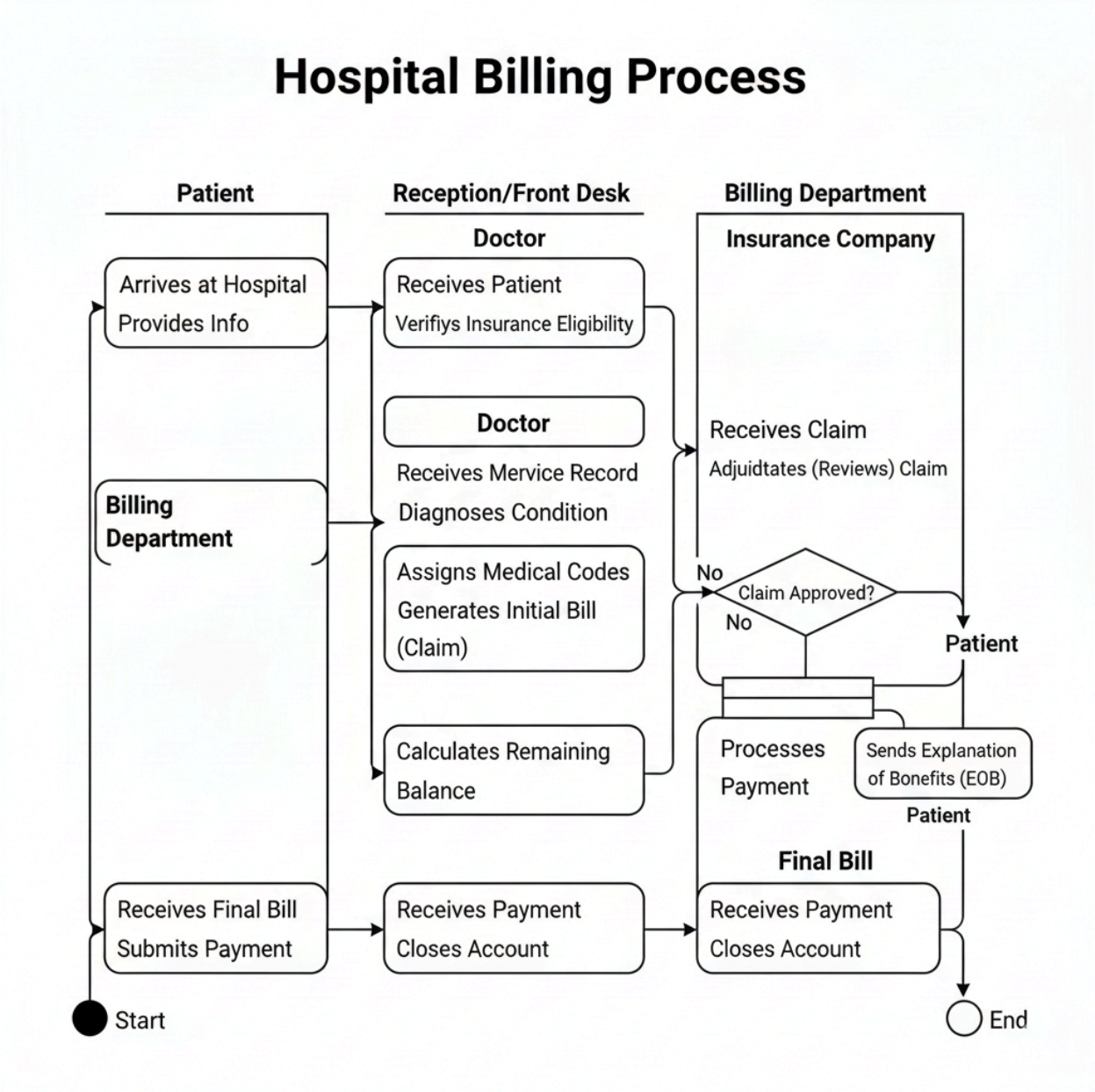
Swimlanes in an activity diagram are **vertical or horizontal lanes** that separate activities based on **who (role or department) is responsible** for performing them. They help make the diagram **clearer**, showing which actor or department performs each action.

Benefits:

- Clarifies responsibilities in the process.
- Helps identify handoffs between roles.
- Improves understanding of workflow across departments.
- **Patient Lane:**
  - The process starts here. The patient **Arrives at Hospital** and **Provides Info** (personal details and insurance card) to reception.
- **Reception/Front Desk Lane:**
  - Reception takes over: **Receives Patient** and **Verifies Insurance Eligibility**.
  - *Handoff:* The patient is then sent to the doctor.
- **Doctor Lane:**

    - The doctor **Examines Patient**, **Diagnoses Condition**, and **Records Services** (e.g., tests, procedures, consultation).
    - *Handoff:* This service record is sent to the billing department.
- **Billing Department Lane:**
    - This lane is often the most complex. The department **Receives Service Record**.
    - It then **Assigns Medical Codes** to each service.
    - It **Generates an Initial Bill (Claim)**.
    - *Handoff:* The claim is sent to the insurance company.
- **Insurance Company Lane:**
    - The insurance company **Receives Claim**.
    - It **Adjudicates (Reviews) Claim**.
    - A decision is made:
        - **If Denied:** It **Sends Denial Notice** back to the Billing Dept.
        - **If Approved:** It **Processes Payment** and **Sends Explanation of Benefits (EOB)** to both the Billing Dept. and the Patient.
    - *Handoff:* The result of the review is sent back.
- **Billing Department Lane (Again):**
    - The billing department **Receives EOB/Payment**.
    - It **Calculates Remaining Balance** (the amount the patient owes).
    - It **Sends Final Bill** to the patient.
    - *Handoff:* The final bill is sent.
- **Patient Lane (Again):**
    - The patient **Receives Final Bill** and **Submits Payment**.
    - *Handoff:* The payment is sent.
- **Billing Department Lane (Final):**

- The department **Receives Payment** and **Closes Account**.
- The process ends.



Describe how activity diagrams can be used to prepare test plans for system verification.

Answer:

Activity Diagrams for Test Plans:

Activity diagrams show the **flow of activities and decisions** in a system. They are very useful for preparing **test plans** because they help testers understand the **sequence of operations, decision points, and parallel processes**.

How they help:

1. **Identify Test Scenarios:** Each activity and decision point can become a **test case** to verify the system works as expected.
2. **Verify Workflow:** Ensures that all steps in a process happen in the correct order.
3. **Check Parallel Processes:** Helps test that **simultaneous actions** (like payment and shipping in an e-commerce system) are handled correctly.
4. **Traceability:** Testers can trace each activity back to requirements, ensuring all functionality is tested.
5. **Edge Cases & Exceptions:** Decision nodes in the diagram highlight **conditions and alternative flows**, which can be included in the test plan.

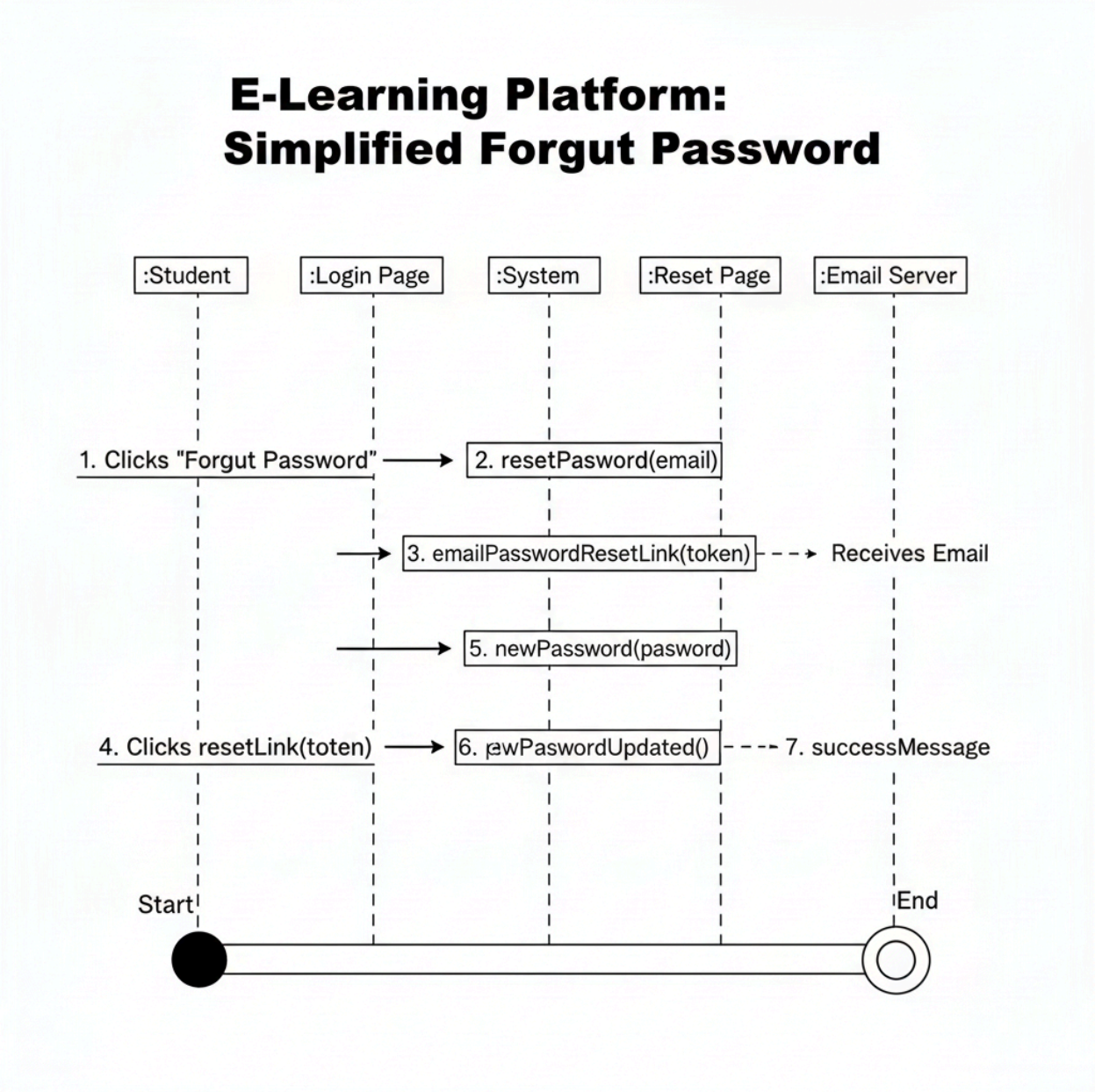
Example:

In an **online purchase system**, an activity diagram shows steps like “Receive Order,” “Validate Order,” “Process Payment,” “Prepare Shipping,” and “Notify Customer.” Testers can create test cases for each step:

- Check if order validation works for valid/invalid orders.
- Verify payment processing for different payment methods.
- Test parallel processing of payment and shipping.

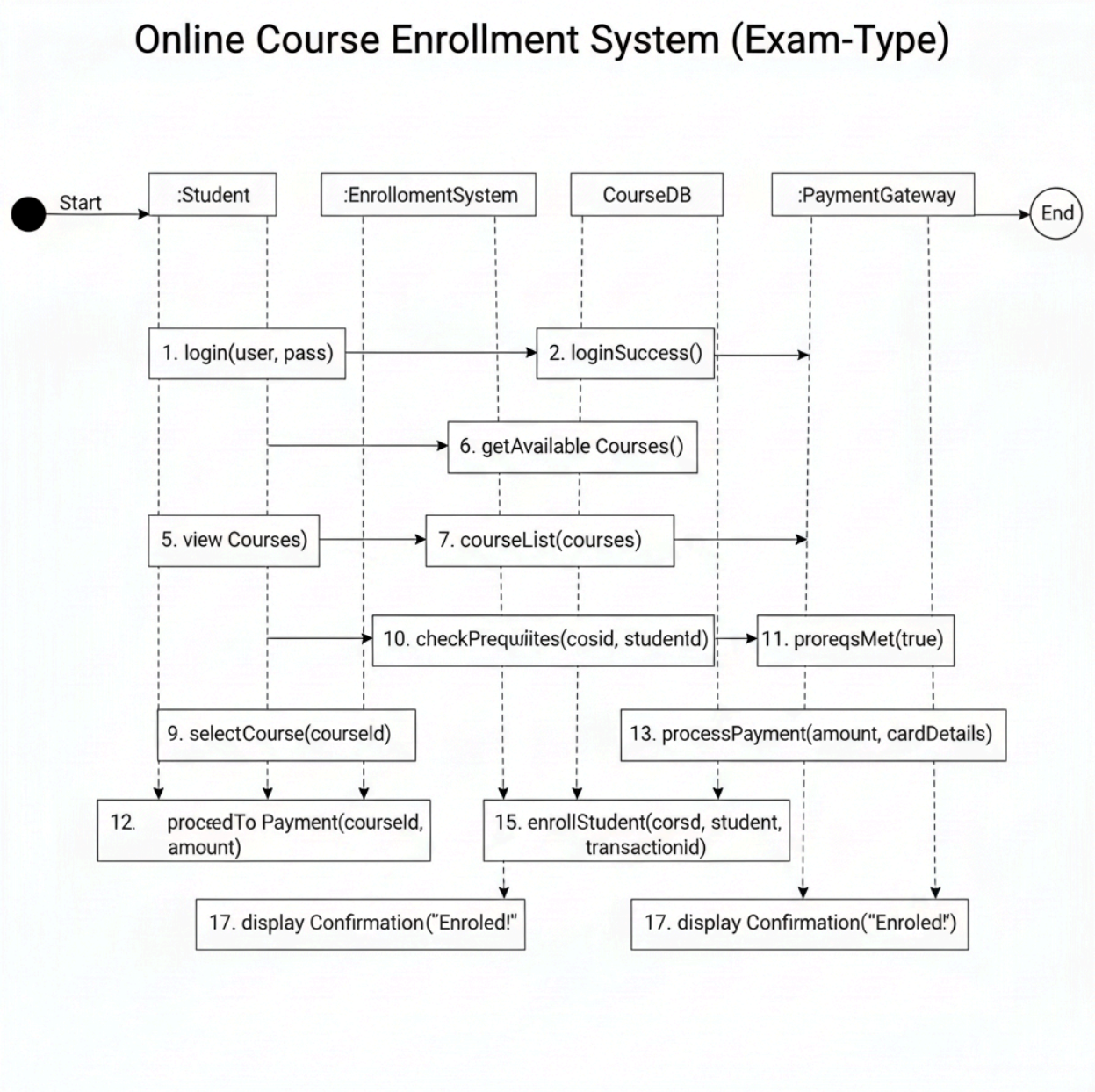
Construct a sequence diagram for the Forgot Password use case in an e-learning platform.

1. **Student Initiates:** The `Student` clicks "Forgot Password" on the `Login Page`.
2. **Request Reset:** The `Login Page` sends a `resetPassword(email)` request to the `System`.
3. **Process Request:** The `System` finds the user, generates a reset token, and sends an `emailPasswordResetLink(token)` to the `Student`.
4. **Student Clicks Link:** The `Student` receives the email and clicks the `resetLink` (which goes to the `Reset Page`).
5. **New Password Submission:** The `Student` enters and submits a `newPassword` on the `Reset Page`.
6. **Update Password:** The `Reset Page` sends `updatePassword(token, newPassword)` to the `System`.
7. **Confirmation:** The `System` updates the password and sends a `passwordUpdated()` confirmation back to the `Reset Page`, which then displays a `successMessage` to the `Student`.



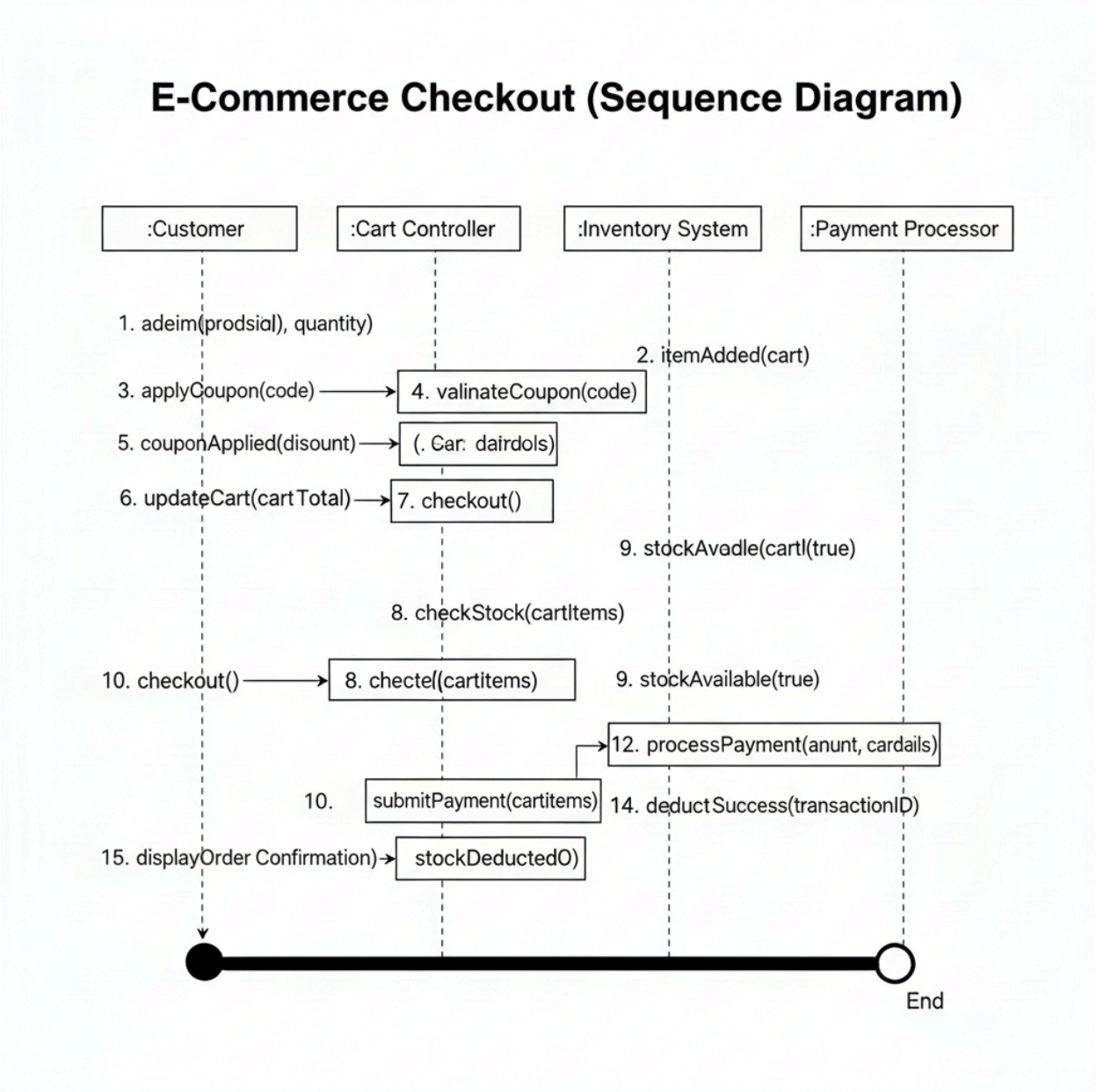
Draw a sequence diagram for an "Online Course Enrollment System" where a student logs in, views courses, checks prerequisites, and enrolls (including payment).

- **Actors/Lifelines:** The diagram clearly shows four key participants: the `:Student` (initiator), the `:EnrollmentSystem` (the application's front-end and business logic), the `:CourseDB` (database for course and student records), and the `:PaymentGateway` (external service for transactions).
- **Login & View Courses:** The student first logs in, and the `EnrollmentSystem` verifies credentials with the `CourseDB` before displaying available courses.
- **Prerequisite Check:** Upon selecting a course, the `EnrollmentSystem` queries the `CourseDB` to ensure the student meets all prerequisites.
- **Payment & Enrollment:** If prerequisites are met, the student proceeds to payment, where the `EnrollmentSystem` interacts with the `PaymentGateway`. Upon successful payment, the `EnrollmentSystem` updates the student's enrollment status in the `CourseDB`.
- **Confirmation:** Finally, the system confirms the successful enrollment to the student.



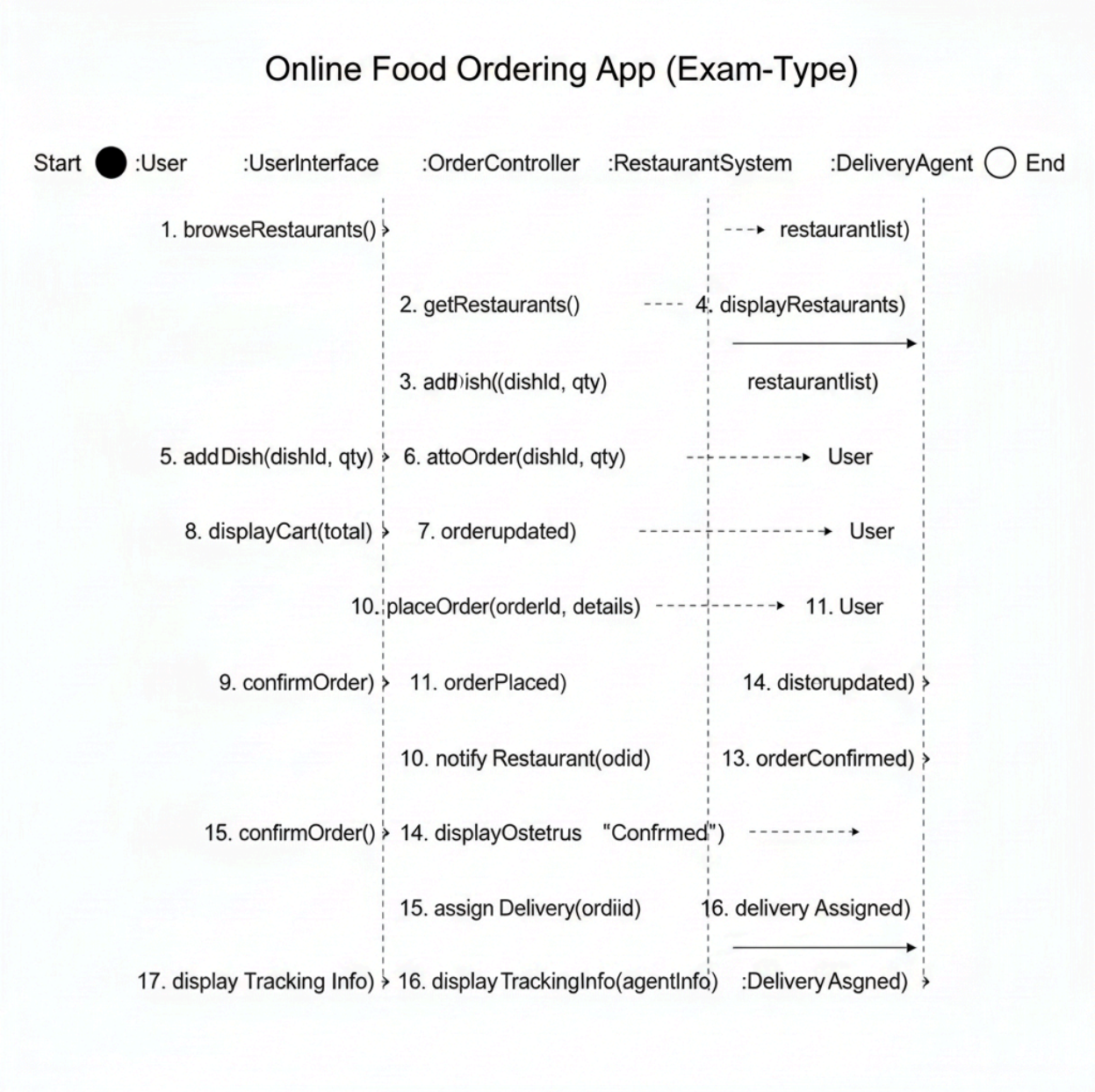
E-Commerce Checkout (Sequence Diagram)
A customer adds items to a cart, applies a coupon, and proceeds to payment. The system verifies stock and processes payment.

Q: Create a sequence diagram showing time-ordered messages between Customer, Cart Controller, Inventory System, and Payment Processor.



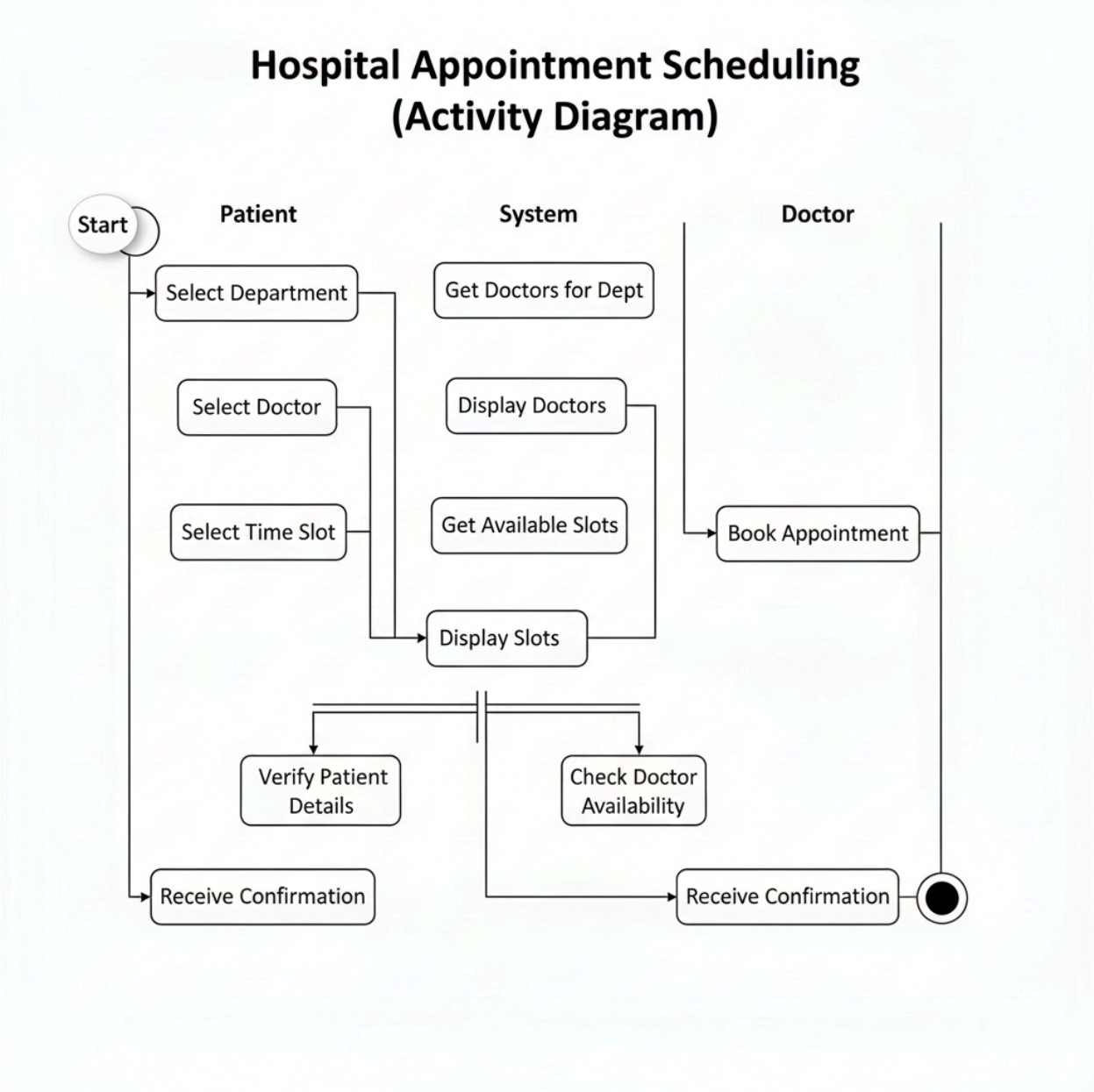
Online Food Ordering App (Sequence Diagram)
A user browses restaurants, adds dishes, confirms the order, and tracks delivery.
Q: Show interactions among User Interface, Order Controller, Restaurant System, and Delivery Agent in a sequence diagram.

- **Participants:** The main actors/systems are the `:User`, the `:UserInterface` (the app itself), the `:OrderController` (handling order logic), the `:RestaurantSystem` (where the food is prepared), and the `:DeliveryAgent` (for delivery).
- **Browsing & Ordering:** The user interacts with the `UserInterface` to browse restaurants and add dishes to their order. The `OrderController` manages these additions.
- **Order Confirmation:** When the user confirms, the `UserInterface` tells the `OrderController` to `placeOrder()`. The `OrderController` then notifies the `RestaurantSystem` about the new order.
- **Delivery Assignment & Tracking:** After the restaurant confirms, the `OrderController` assigns the order to a `DeliveryAgent`. The `UserInterface` then provides tracking information to the user, completing the order and tracking process.



Hospital Appointment Scheduling (Activity Diagram)
A patient selects a department, chooses a doctor, checks available time slots, confirms an appointment, and receives a confirmation email.

Q: Draw an activity diagram showing sequential and parallel activities (like doctor and patient verification running concurrently). Use swimlanes for Patient, System, and Doctor.

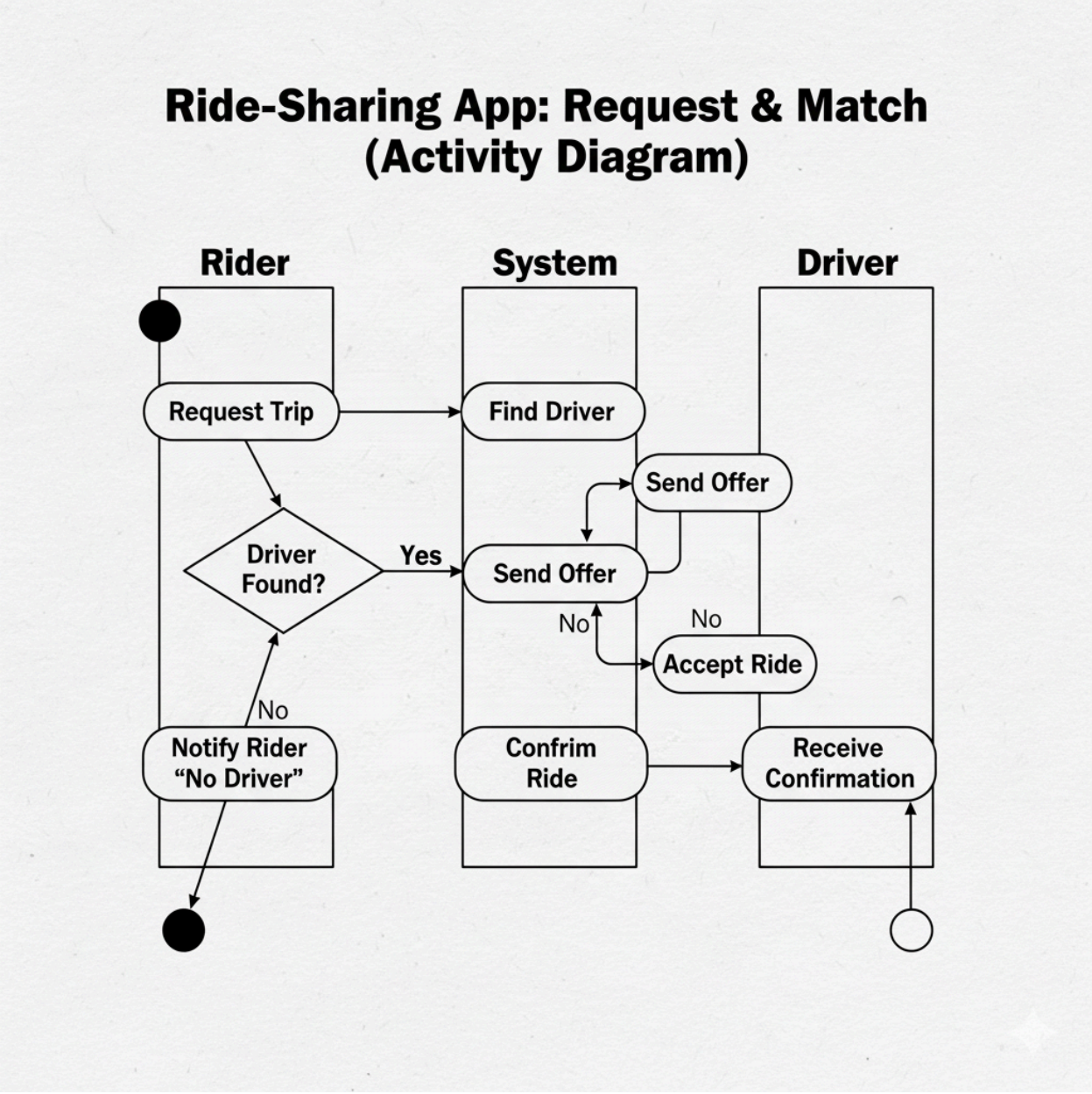


This activity diagram models the process of a patient scheduling a hospital appointment.

- **Swimlanes:** It clearly separates responsibilities into three lanes: `:Patient` (the user actions), `:System` (automated processes and UI interactions), and `:Doctor` (actions related to the doctor's schedule or information).
- **Sequential Flow:** The initial steps, from selecting a department to choosing a time slot, follow a clear sequential path, primarily driven by the patient's choices and the system's responses.
- **Fork & Join (Parallel Activities):** A critical part of this diagram is the **fork** after the patient selects a time slot. This splits the flow into two parallel activities handled by the `System`:
    1. **Patient Verification:** The system verifies the patient's details.
    2. **Doctor Schedule Check:** Concurrently, the system performs a final check on the doctor's schedule to ensure the slot is still available and valid. These two paths then **join**, meaning both must complete successfully before the process can continue to the next step.
- **Confirmation:** After parallel checks, the system books the appointment, generates a confirmation, and sends it to the patient, ending the process.

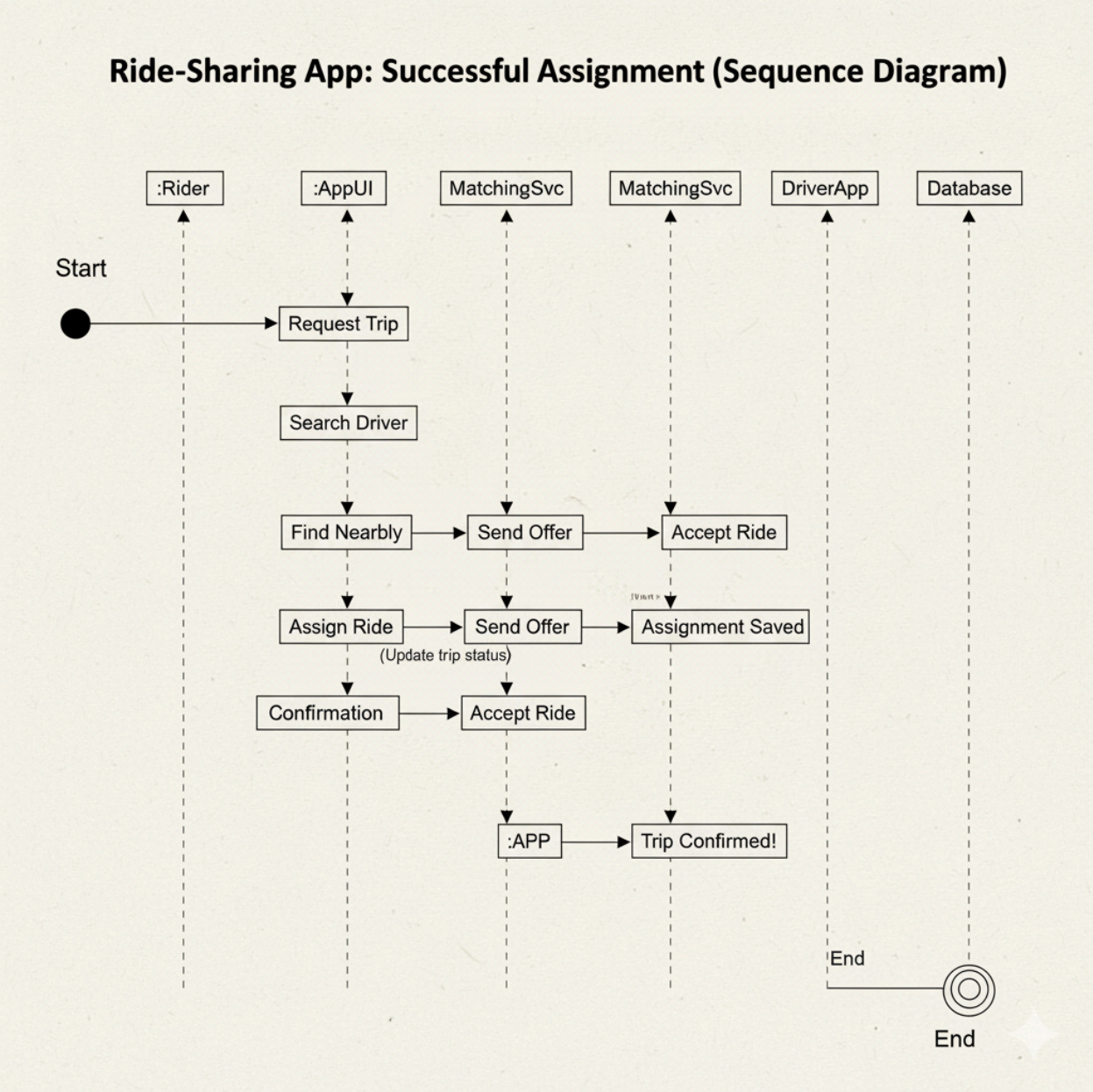
Ride-Sharing App (Activity + Sequence Diagram)
A rider requests a trip, the system matches a nearby driver, and the driver accepts or declines.
Q:
a) Draw an activity diagram showing the decision points.

b) Create a sequence diagram for successful ride assignment.



Key Points (Activity Diagram):

- **Swimlanes:** Rider, System, Driver.
- **Flow:** Rider requests trip → System finds driver.
- **Decisions:**
  - If `[No Driver?]`, notify Rider.
  - If `[Driver Found?]`, `Send Offer` to Driver.
  - Driver `[Accept?]` (Yes/No).
- **Loops:** `Decline` leads to `Search Next Driver` (implied loop).
- **Success:** `Accept` leads to `Confirm Ride` for Rider.



Key Points (Sequence Diagram):

- **Lifelines:** Rider, AppUI, MatchingSvc, DriverApp, Database.
- **Sequence:**
  1. Rider `Request Trip` via `AppUI`.

2. `AppUI` asks `MatchingSvc` to `Search Driver`.
3. `MatchingSvc` `Find Nearby` drivers (using `Database`).
4. `MatchingSvc` `Send Offer` to `DriverApp`.
5. `DriverApp` sends `Accept Ride` to `MatchingSvc`.
6. `MatchingSvc` `Assign Ride` (updating `Database`).
7. `AppUI` receives `Confirmation` for Rider.