Mastering Object-Oriented Technology (OOT)

UML & System Design Masterclass from Scratch (Module: ITP4909)

Welcome to my class. Today, we will demystify Object-Oriented Technology. I will teach you not just the core concepts, but exactly how to conquer your final exam. Learning OOT means learning how to think about system structure like an architect.

The Professor's Golden Rule: Explanations in simple terms for deep understanding. Keywords, UML notations, and exam terms strictly in professional English syntax for high marks.

The ITP4909 Exam Blueprint

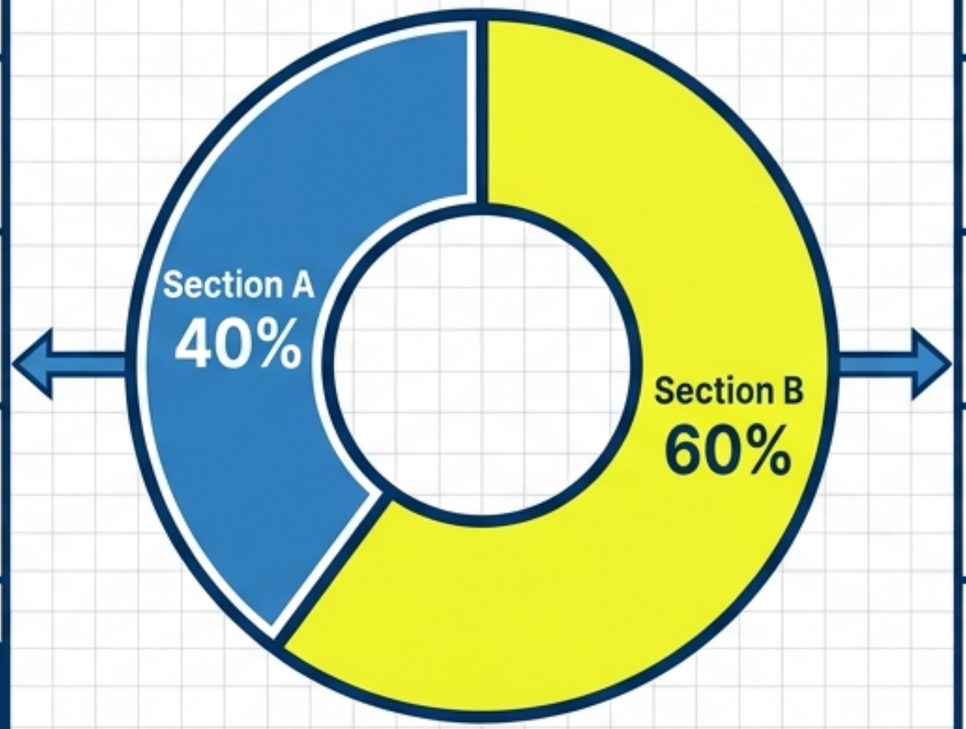
Section A (40 Marks) - Core Implementation

Class Diagram (12-13 marks): Drawing from scenario.

Java Implementation (12 marks): Translating design to code.

Behavior Diagrams (15 marks): Activity or State Machine diagrams.

Professor's Note: This section tests your fundamentals, especially the ability to translate static design directly into Java code.



Section B (60 Marks) - System Analysis & Design

Use Case Diagram (13-20 marks): System boundaries.

Textual Analysis (8 marks): Identifying candidate classes.

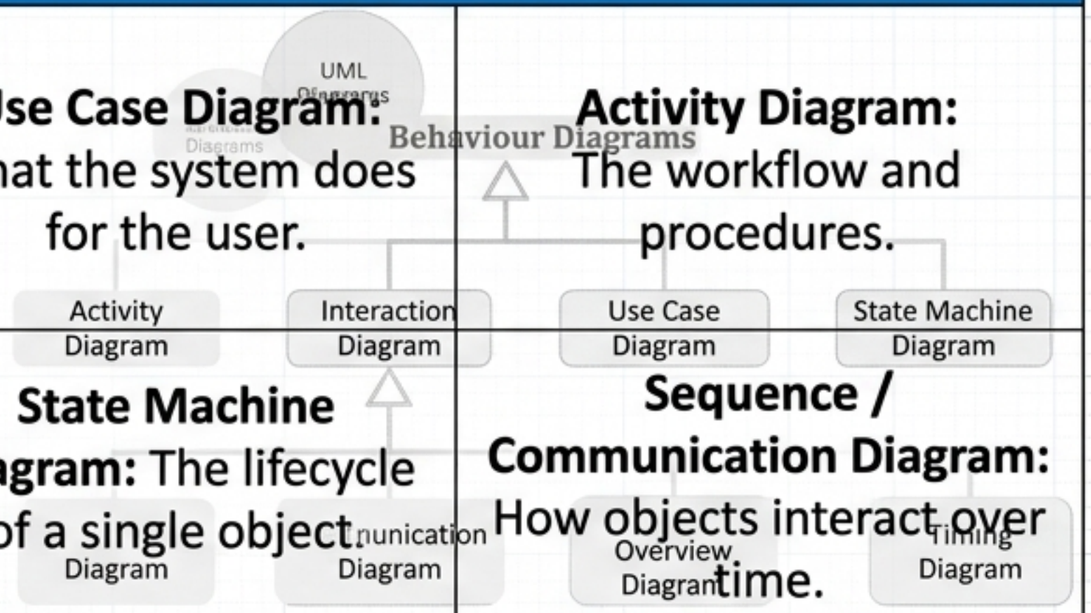
3-Tier Sequence Diagram (10-15 marks): Dynamic modeling.

Professor's Note: Section B is a continuous story! B1's answer affects B2, and B2's extracted classes must be used directly in B3's sequence diagram.

The UML Universe

Learning OOT means learning how to describe the exact same system from different angles. Static diagrams show the skeleton; dynamic diagrams show the interactions.

	Structure Diagrams (Static)	Behavior Diagrams (Dynamic)	
System-Level	**Class Diagram:** The system architecture and relationships.	**Use Case Diagram:** What the system does for the user.	**Activity Diagram:** The workflow and procedures.
Object-Level		**State Machine Diagram:** The lifecycle of a single object.	**Sequence / Communication Diagram:** How objects interact over time.



Assessment Focus: You must be able to draw *one Structure diagram (Class)* and at least three *Behavior diagrams (Use Case, Activity/State, Sequence) entirely from scratch.*

Phase 1: Gathering Requirements (Use Case Diagram)

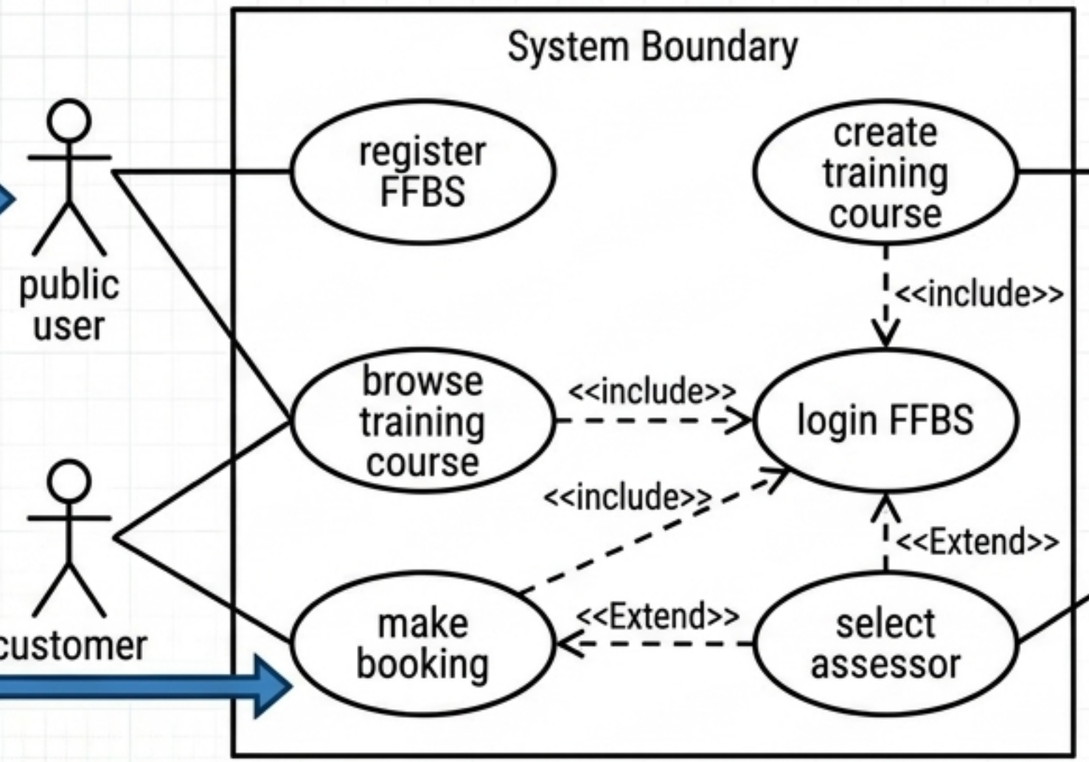
Describes the observable results of value the system provides to an actor.

Flashcard

Actor: An external entity interacting with the system. MUST be placed outside the System Boundary box!

Flashcard

Use Case: A sequence of actions yielding a distinct result.



Flashcard

<<include>>: Mandatory dependency. Dashed arrow points TO the included step.

Flashcard

<<extend>>: Optional behavior. Dashed arrow points FROM the extending step.

Exam Assessment (13-20 Marks): Deduction for excess irrelevant use cases or communication links. Do NOT invent unmentioned features! Only draw what the prompt requires.

Phase 2: Textual Analysis (The Secret Bridge)

How do we extract system classes from a long text prompt?
We identify and categorize the nouns.

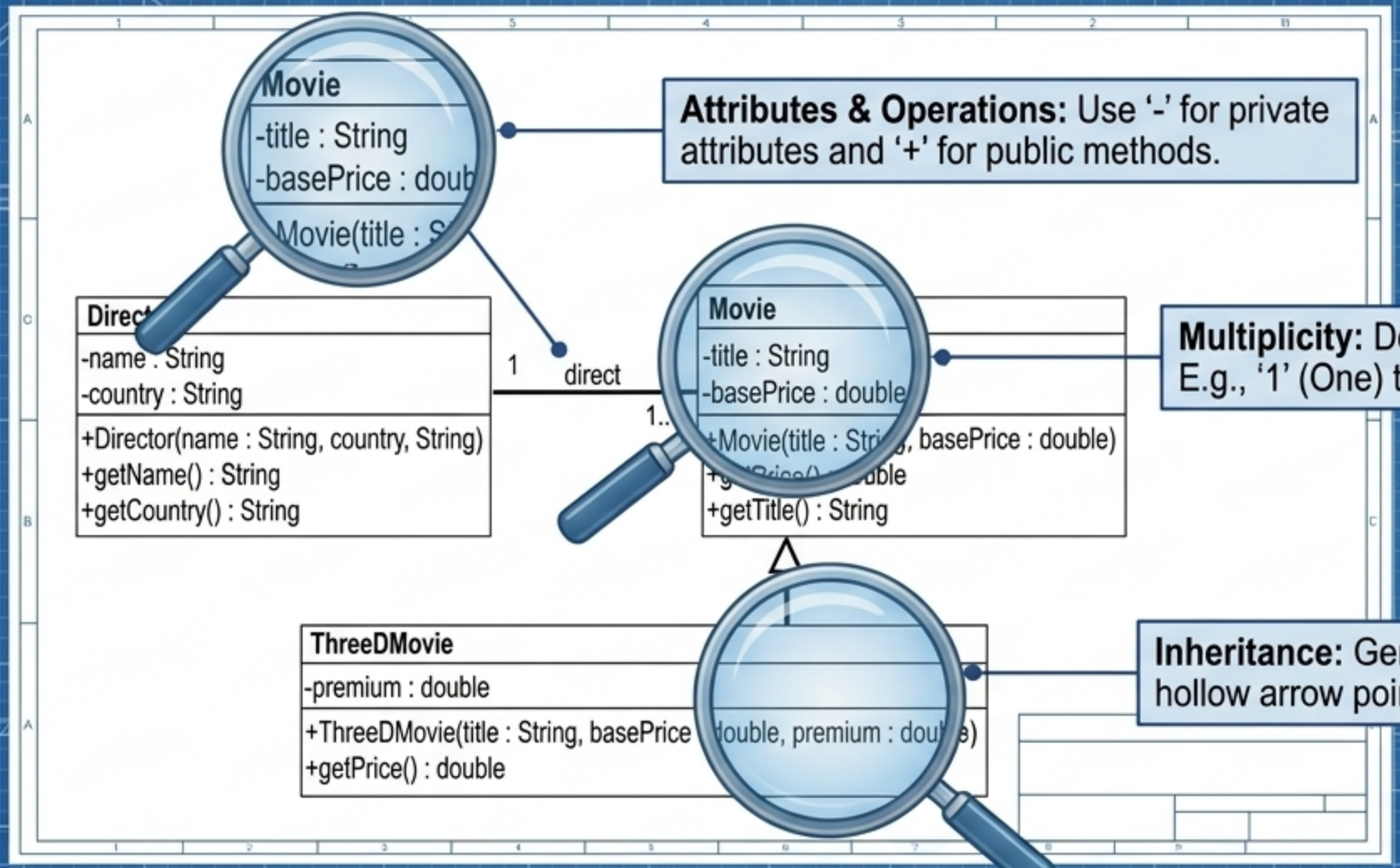
A [Customer] orders [Coffee]... The [Booking] is paid via a [Payment Gateway]...

The 5 Exam Categories (Memorize These)	
Category	**Examples**
Role Play	Customer, Clinic Nurse, Manager
Tangible Thing	Coffee, Vaccine, Car, Hardware
Conceptual Thing	Subscription, Account, Plan
Event	Booking, Delivery, Payment
External System	Payment Gateway, SMS Gateway

Exam Assessment (8 Marks): The question will ask for 'Candidate Classes' and their 'Reasons'. You must write down the exact 5 English terms above to get full marks.

Phase 3: Building the Blueprint (Class Diagram)

The static skeleton of the system, showing classes, attributes, and their mathematical relationships.



Attributes & Operations: Use '-' for private attributes and '+' for public methods.

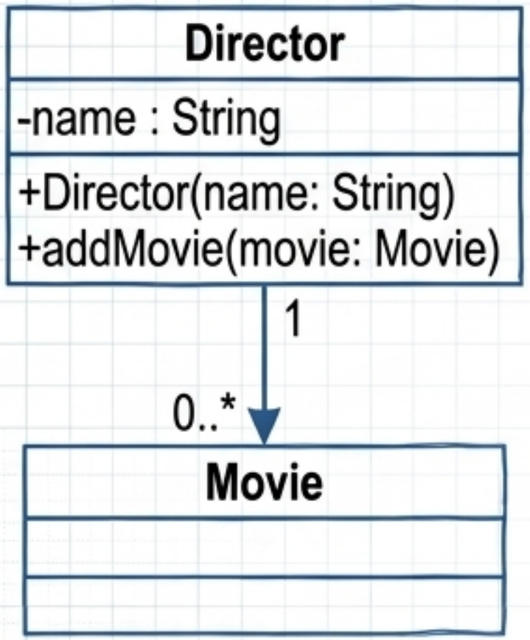
Multiplicity: Defines database relations. E.g., '1' (One) to '1..*' (One or more).

Inheritance: Generalization is drawn with a hollow arrow pointing to the Superclass.

Exam Assessment (13 Marks): Read the prompt carefully. Always show the name and multiplicity of each association. Include all explicitly mentioned attributes!

Translating Design to Code (Java Implementation)

Core Concept: The exam requires translating the visual Class Diagram directly into concrete Java code.



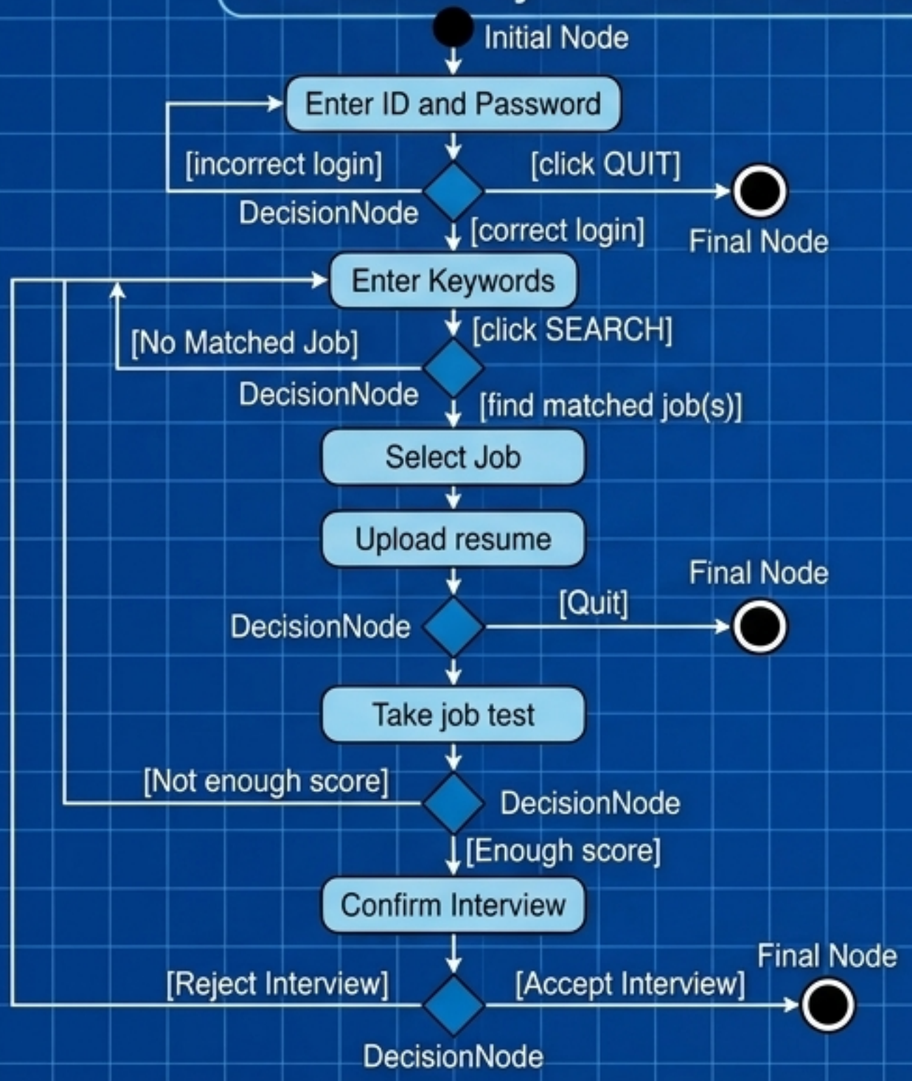

```java
public class Director {
    private String name;
    private Vector _movies;

    public Director(String name) {
        this.name = name;
        _movies = new Vector();
    }
    public void addMovie(Movie movie) {
        _movies.add(movie);
    }
}
```

Constructors:	Inheritance:	1-to-Many:
Always initialize your attributes inside the constructor.	Use the keyword ‘extends’. You MUST call ‘super(...)’ inside the child’s constructor.	Use Java ‘Vector’ to hold multiple objects, and ‘Enumeration’ to retrieve them.

Professor’s Hack (12 Marks): Look closely at the provided TestDriver.java in the exam! The ‘new’ keywords tell you exactly what parameters your Constructors must have.

Behavior Modeling I: Activity Diagram

Shows the procedural flow of control when the system executes a task. It models dynamic workflows and business processes.



Visual Dictionary

Initial Node: The starting point.

Flashcard

Action State: A single procedural step.

Flashcard

Decision Node: Branching based on conditions.

Flashcard

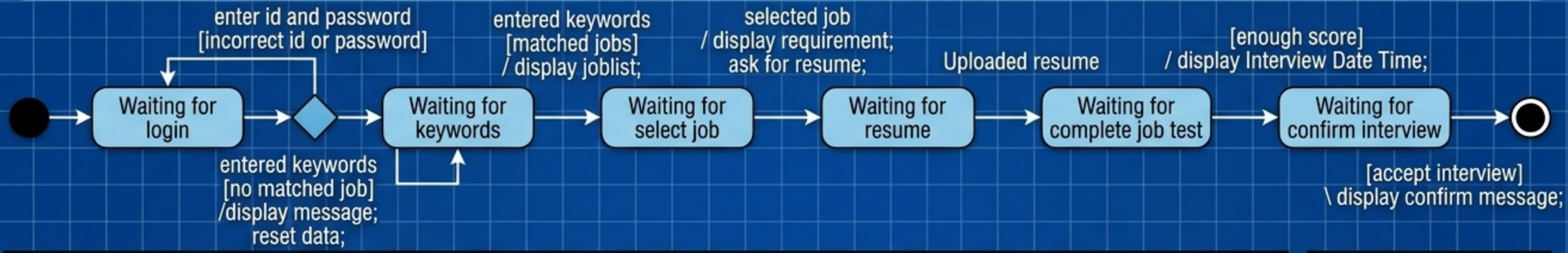
Final Node: The ending point.

Flashcard

Crucial Tip (~6 Marks): Every single branch exiting a Decision Node MUST have a [guard condition] written in square brackets (e.g., [correct login]). Missing this loses points!

Behavior Modeling II: State Machine Diagram

Core Concept: Describes the state changes of A SINGLE OBJECT during its entire lifecycle. A state occupies an interval of time.



Anatomy of a Transition (Memorize this Syntax!)

event [guard condition] / action

- **event** — The trigger (e.g., 'enter keywords')
- **[guard condition]** — The requirement to proceed (e.g., '[matched jobs]')
- **action** — The immediate result (e.g., '/ display joblist')

Inside the State Block:

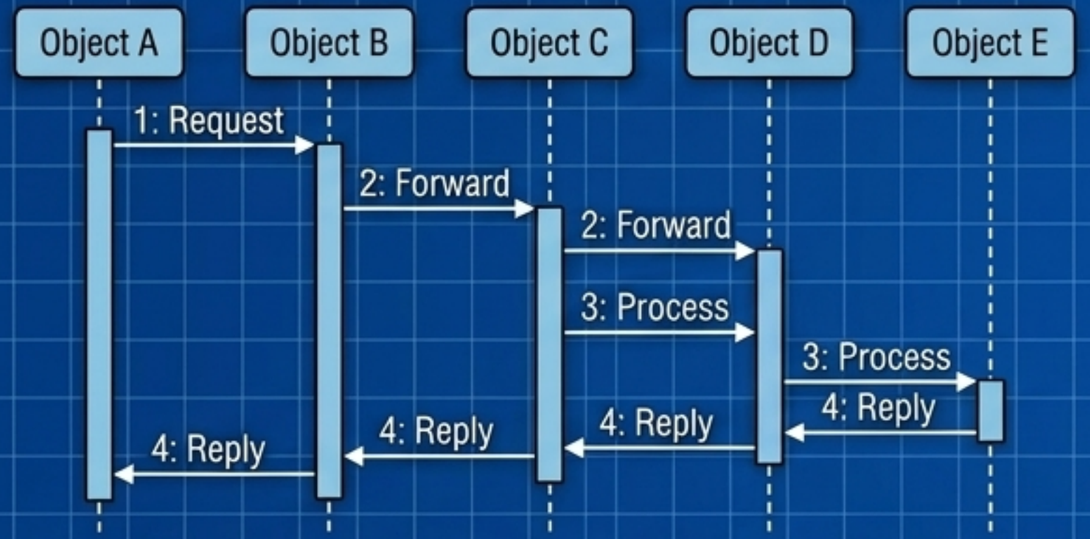
You can define 'entry / action', 'do / activity', and 'exit / action'.

Professor's Tip (~7 Marks): Activity diagrams show "where the overall process is". State diagrams show "what condition a specific object is in".

Interaction Diagrams (Scenario Modeling)

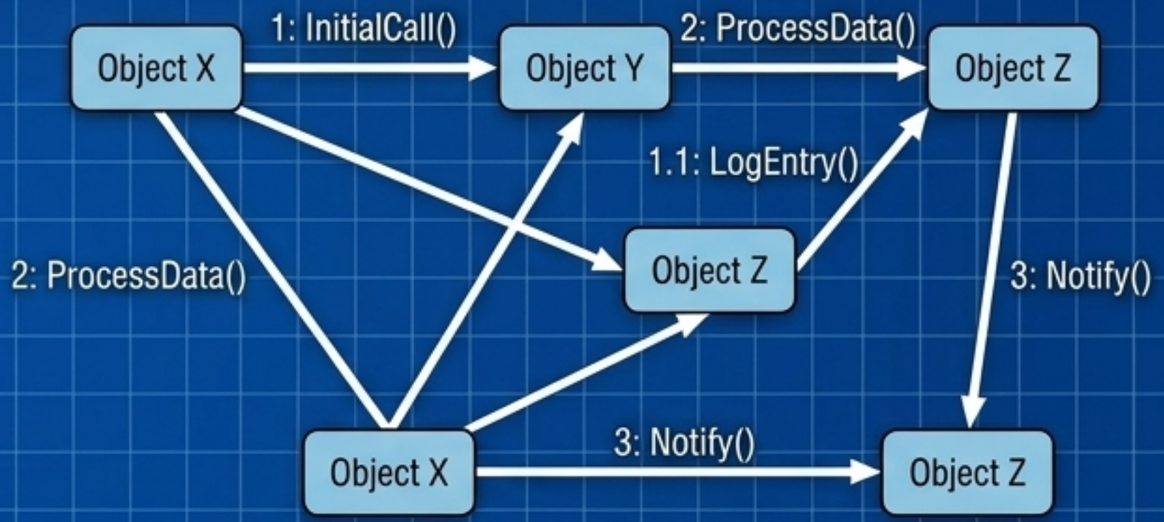
Core Concept: How do objects in the system communicate to achieve a goal? Each event involves objects passing messages to one another.

Sequence Diagram



Shows the detailed flow for a specific use case, ordered vertically by TIME.

Communication Diagram

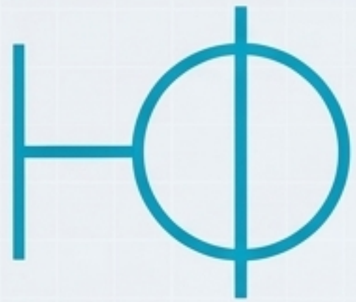


Focuses on the structural organization and networked links between collaborating objects.

The Pivot: While both are valid UML, the **ITP4909 Exam** places massive weight on the Sequence Diagram. Next, we prepare to draw the ultimate version: **The 3-Tier Architecture.**

The 3-Tier Architecture (MVC Framework)

Core Concept: To make systems easier to maintain, we divide objects into three distinct tiers. This strictly separates the user interface from business logic and data.



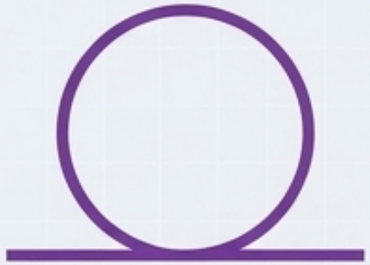
Boundary Object (View)

Handles user interactions with the system.

e.g., <<Boundary>> :SystemUI

Control Object (Controller)

The 'brain'. Management of control flow and transactions.
e.g., <<Control>> :SystemController



Entity Object (Model)

The data holders. Information retrieval and update.
e.g., <<Entity>> :Account

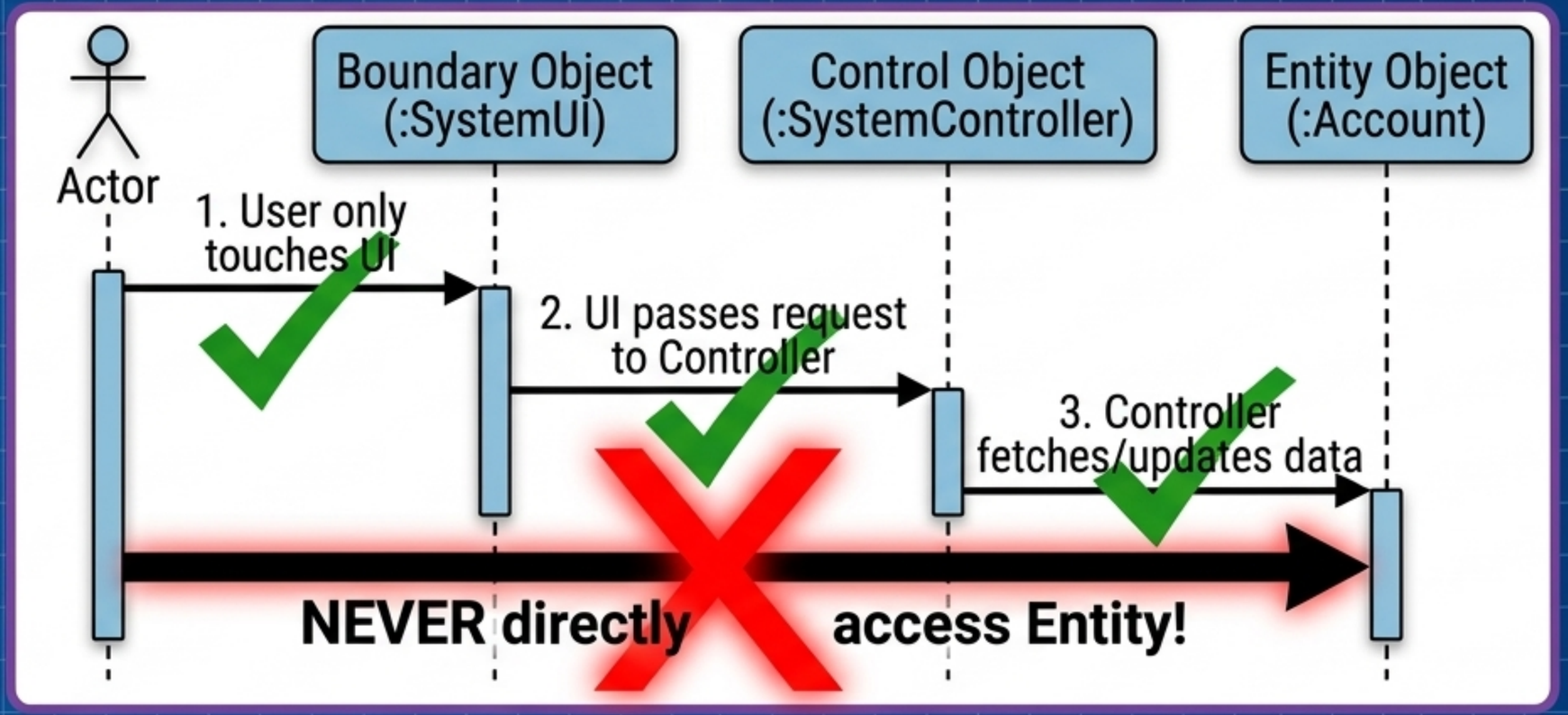
Exam Tip: The specific Boundary, Control, and Entity objects you use here will come directly from the **Candidate Classes** you identified in your Phase 2 Textual Analysis!

The Rules of the 3-Tier Sequence Diagram

Core Concept: You cannot draw arrows randomly. There is a strict, ironclad hierarchy of communication.

The Ironclad Rules (Memorize)

- 1. Actor ONLY talks to Boundary.
- 2. Boundary ONLY talks to Control.
- 3. Control is the boss. It commands the Entity.
- 4. Entity objects **NEVER** talk to the Actor directly.

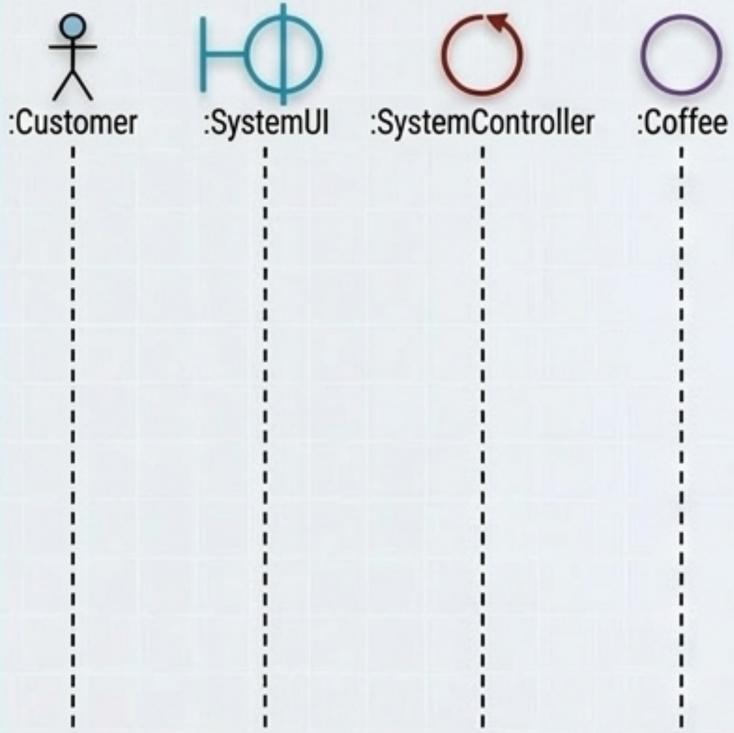


Exam Red: Constraint (10-15 Marks): Look for the exam note: "Only one boundary object and one controller object are required." Keep your architecture simple!

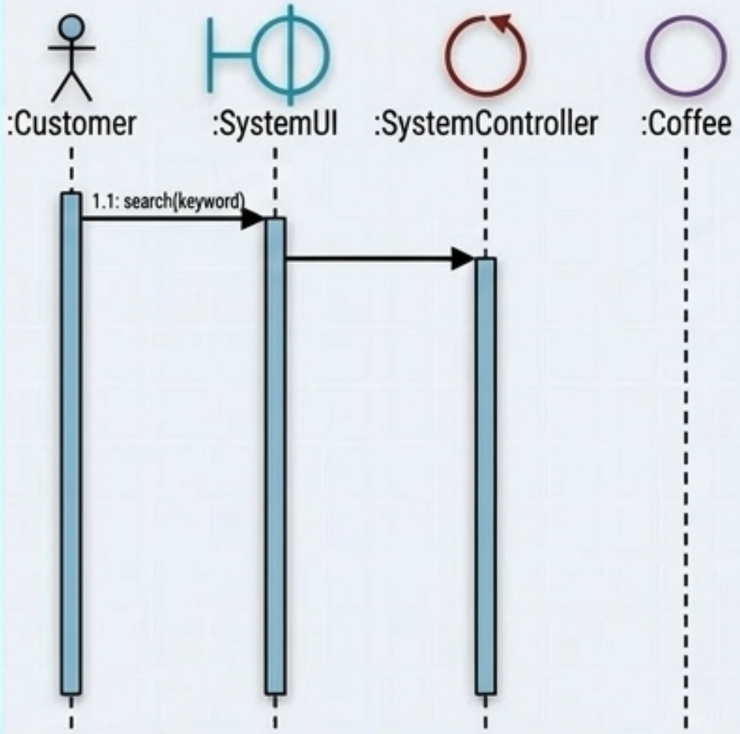
Sequence Diagram Mastery (Step-by-Step)

Core Concept: Construct your diagram methodically to ensure perfect logic and formatting.

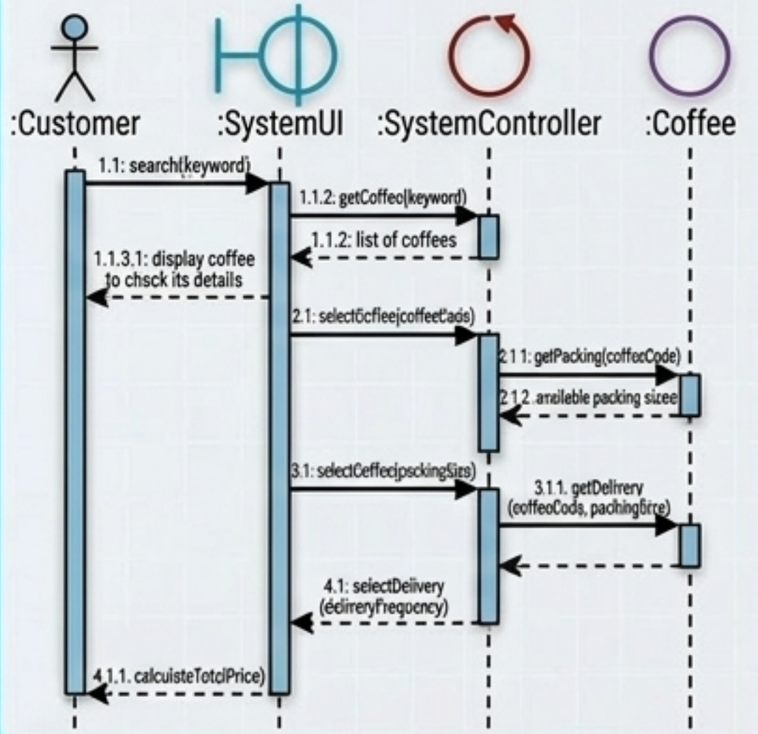
Step 1: Identify Objects



Step 2: Forward Messages



Step 3: Return Messages



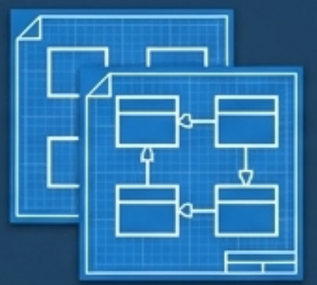
Professor's Warning: Time flow is strictly numbered! (1, 1.1, 1.1.1). Pay extreme attention to numbering your message method calls correctly!

Synthesis: The Complete OOT Journey

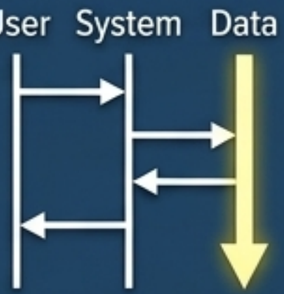
OOT is not a set of random diagrams; it is a unified, logical process. You are designing a building from the outside in.

1. Goal: Start with what the user wants (Use Case).

2. Extract: Read the scenario and extract the nouns (Textual Analysis).



3. Structure: Define those nouns as your building blocks (Class Diagram).



4. Animate: Show how those blocks talk across 3 tiers (Sequence Diagram).

5. Build: Translate design into Java Code.

Conclusion: You are no longer just coding.
You are thinking like a Software Architect.

The Professor's Final Exam Checklist

Before you hand in your paper, check these critical point-losers:

- [x] **Class Diagram:** Did I explicitly write down the multiplicities (e.g., 1..*) and all attributes?
- [x] **Java Code:** Did I use super(...) inside the constructors of inherited classes?
- [x] **Textual Analysis:** Did I use the exact English keywords (e.g., Tangible Thing, Event)?
- [x] **Sequence Diagram:** Did I ensure the Actor ONLY communicates with the Boundary object?
- [x] **State/Activity Diagrams:** Are my [guard conditions] written strictly inside square brackets?

Good luck! You have the blueprint. You are ready to conquer the exam.