Sure! Here are some **big scenario-based questions** with in-depth reasoning, perfect for practicing long-answer or case-based midterm questions. These go deep into your understanding of **SDLC, system analysis, feasibility studies, methodologies, and OOAD**—all from Lecture 1.

**🔍 Scenario-Based Long Questions with Answers**

**Question 1: SDLC & Feasibility**

**Scenario**:  
A government university plans to build a **centralized online admission system** that allows students to apply, upload documents, and track admission progress. The system must be integrated with the national education board’s database and payment gateways. The university has limited technical staff and a tight deadline.

**Q**: As a system analyst, how would you apply the **System Development Life Cycle (SDLC)** to guide this project from planning to maintenance? Also, identify **which feasibility studies** should be conducted and explain the factors to consider for each.

**Answer**:

* **Planning Phase**:
  + Identify the problems in the current manual admission process.
  + Define project objectives, like reducing application time and improving transparency.
  + Produce a project schedule.
* **Feasibility Studies**:
  + **Technical Feasibility**: Check if current servers and databases can support such a system. Determine if technical staff can handle integration.
  + **Economic Feasibility**: Cost-benefit analysis for developing vs. outsourcing.
  + **Legal Feasibility**: Ensure the system complies with data privacy laws.
  + **Scheduling Feasibility**: Evaluate if the system can be delivered before the next admission session.
  + **Operational Feasibility**: Assess if staff and students are willing and able to adapt to the new system.
* **Analysis and Specification**:
  + Gather requirements from admission office, IT department, and students.
  + Develop a Software Requirements Specification (SRS) document.
* **Design**:
  + Design UI/UX, database, user roles (admin, student, reviewer).
  + Prepare system architecture with payment and board integration.
* **Implementation**:
  + Developers write code, test modules, integrate APIs.
* **Testing & Deployment**:
  + Conduct UAT (User Acceptance Testing).
  + Deploy the system on university servers.
* **Maintenance**:
  + Provide ongoing support, fix bugs, and upgrade features.

**Question 2: System Analyst Role & Object-Oriented Design**

**Scenario**:  
A logistics company wants to modernize their fleet management system to track deliveries, vehicle maintenance, driver behavior, and fuel usage. They want the new system to be scalable, modular, and maintainable in the long term. You are assigned as the lead system analyst.

**Q**: Describe your role as a system analyst in this project. How would you approach system design using **object-oriented principles** like encapsulation, inheritance, and polymorphism? Provide specific examples.

**Answer**:

* **As a System Analyst**, I would:
  + **Understand Requirements**: Interview logistics managers, drivers, and support staff.
  + **Analyze Current System**: What tools are used now? Any pain points?
  + **Document User Requirements** using SRS.
  + **Communicate with Developers**: Translate business needs into technical specs.
* **Object-Oriented Design Application**:
  + Use **UML Class Diagrams** to define entities:
    - Vehicle (attributes: ID, fuelType, maintenanceDate)
    - Driver (attributes: name, licenseNo)
    - Delivery (attributes: deliveryID, route, destination)
* Apply **Encapsulation**:
  + Hide internal tracking logic inside classes. Example: calculateFuelEfficiency() only accessible through a public method.
* Use **Inheritance**:
  + Truck and Van can inherit from Vehicle class with shared properties.
* Apply **Polymorphism**:
  + The method scheduleMaintenance() behaves differently for electric vehicles and diesel trucks based on vehicle type at runtime.

**Question 3: Methodology Selection**

**Scenario**:  
A startup plans to launch a new **mobile banking app**. They want to release a basic version in 3 months, gather user feedback, and then iterate. The app must support login, balance check, fund transfer, and notifications. Security is a high priority.

**Q**: Which **SDLC methodology** would be best suited for this project and why? Compare it with at least one alternative approach and justify your selection.

**Answer**:

* **Recommended Methodology: Agile Development**
  + Agile is **iterative** and **incremental**, allowing the team to deliver core features (login, balance check) in early sprints.
  + Feedback is collected quickly and used to improve the next version.
  + Agile supports **changing requirements**—crucial for startups based on user feedback.
* **Security in Agile**:
  + Security tests and penetration tests can be part of each sprint cycle.
  + Continuous integration allows fast patching of vulnerabilities.
* **Comparison with Waterfall**:
  + Waterfall requires completing each phase before the next—too rigid for fast-moving startups.
  + Inflexible to changes in user requirements.
* **Conclusion**:
  + Agile offers the flexibility, speed, and iterative learning necessary for a modern banking app startup.

Want me to generate a few more of these case-style questions with answers from the next lecture or expand on one of these topics more deeply (like UML diagrams or requirement gathering)?