Review Questions (OOAD Concepts)

1. What characteristics do all systems have in common, regardless of their type (e.g., electrical, transportation, education)?
2. How can a refrigerator be considered both a tool and a system, depending on the context?
3. Define a system. Why is collaboration between components essential?
4. In the restaurant example, what are the sub-systems, and how do they interact with the environment?
5. What are the main differences between a tool and a system?
6. List and explain at least three principles of software development mentioned in the slides.
7. Compare the **structured approach** and the **object-oriented approach** in system analysis and design.
8. What are the main purposes of ERD and DFD diagrams in the structured approach?
9. Explain the five main object-oriented principles (classification, inheritance, encapsulation, messaging, state).
10. What are the main benefits of applying OOAD in software development?

Essay Questions (System Analysis – OOAD Part 1)

1. Explain the difference between the "outside view" and the "inside view" of system analysis. Why are both perspectives necessary?
2. Describe the role of a system in its working environment. Use the online shopping website example to illustrate your answer.
3. What are system use cases, and how do actors interact with the system to fulfill them? Provide examples from the online shopping or storehouse management scenarios.
4. Discuss the importance of business rules in defining system use cases. How do business rules influence the responsibilities of a system?
5. What is UML, and why is it considered an essential tool for system modeling in OOAD?
6. Compare and contrast the relationships among use cases: generalization, inclusion, and extension. Provide examples to support your explanation.
7. How can sequence diagrams and communication diagrams be used together to analyze use cases? Illustrate with the "Open Account" use case example.
8. Explain the meaning and role of generalization, aggregation/composition, and association in class diagrams. Provide one real-world example for each.
9. Describe how a state model captures the dynamic behavior of objects. Use the traffic light example to explain state transitions.
10. Using the coffee cup or storehouse examples, explain how identifying interactions with external actors helps define use cases and system responsibilities.

20 Multiple-Choice Questions (System Analysis – OOAD Part 1)

1. In system analysis, the **outside view** considers the system as:

1. A set of components inside the system
2. A black box tool in its environment
3. A data flow diagram
4. A programming interface

2. The **inside view** of system analysis focuses on:

1. Actors only
2. System as a tool in environment
3. Internal components and their collaboration
4. Business rules

3. Which question is most important when starting system creation?

1. What programming language should be used?
2. What are the roles and responsibilities of the system in its environment?
3. How much budget is available?
4. How many UML diagrams are required?

4. System requirements are mainly derived from:

1. System designers' preferences
2. Market, laws, and company goals
3. Random test cases
4. End users only

5. In the online shopping example, which three problems must be solved?

1. Get orders, get payments, deliver products
2. Inventory, promotion, design
3. Marketing, packaging, reviews
4. Logistics, HR, taxation

6. In Online Shopping Case 1:

1. Customers order & pay online
2. Customers order via website, staff delivers & collects payment
3. Staff only designs website
4. Bank is always involved

7. In Online Shopping Case 2:

1. Website handles only order
2. Staff receives both orders and payments from website
3. Customer never pays online
4. Delivery is unnecessary

8. In UML, an **actor** is:

1. A programming module
2. An object outside the system interacting with it
3. A database schema
4. A child class

9. Which is NOT a correct statement about a use case?

1. A use case is a situation in which the system is used
2. A use case defines requirements on objects/system
3. A system function is always a use case
4.

4. Actors participate in use cases

10. In UML, "Use Case A **includes** Use Case B" means:

1. A may or may not use B
2. B must always occur for A
3. A replaces B
4. A inherits behavior from B

11. In UML, "Use Case A **extends** Use Case C" means:

1. C is optional and adds extra processing to A
2. C replaces A
3. A inherits C's behavior
4. A is always required with C

12. Use case generalization means:

1. One use case inherits actors and behavior from another
2. Two use cases always run together
3. Use case is split into multiple subsystems
4. Actors are removed

13. Which diagram shows the **order of messages** between objects?

1. Class diagram
2. Sequence diagram
3. Activity diagram
4. State diagram

14. A **communication diagram** is different from a sequence diagram because it:

1. Focuses on structure of communication
2. Shows message order only
3. Represents inheritance
4. Does not involve actors

15. In class diagrams, **generalization** means:

1. Child classes share all attributes & behaviors of parent
2. Objects interact with each other
3. One class contains another
4. System states are inherited

16. In class diagrams, **aggregation/composition** describes:

1. Logical relationship between classes
2. Part-whole relationship (e.g., car has wheels, engine)
3. Inheritance between classes
4. State transitions

17. In class diagrams, **association** is:

1. Logical relationship where classes exist independently
2. Always a whole-part dependency
3. A multi-inheritance relationship
4. A state model

18. A state model describes:

1. Database design
2. Objects' states and transitions caused by events
3. Class inheritance
4. Actor hierarchy

19. In the traffic light example, how many states exist at an intersection?

1. 2
2. 3
3. 4
4. 6

20. In the coffee cup example, which of the following is a use case?

1. Filling, pouring, drinking
2. Selling, marketing, advertisement
3. Inheriting, polymorphism, encapsulation
4. Sequence, state, activity

10 Essay Questions (System Analysis – Inside View)

1. Explain the purpose of analyzing the system from the **inside view** in system analysis. How does it complement the outside view?
2. Describe how **scenario analysis** helps identify candidate objects in a system. Use the "Borrow book" use case as an example.
3. What is the role of **CRC (Class–Responsibilities–Collaborators) cards** in OOAD? Why is role-playing scenarios useful?
4. In the "Open Account" use case, identify the main system components and explain how they collaborate to achieve the system's responsibilities.
5. Discuss how **sequence diagrams** and **communication diagrams** complement each other in describing object interactions.
6. Why is it important to specify both **class attributes** and **class methods** when designing object classes?
7. Compare and contrast **association, aggregation, and composition** relationships in class diagrams. Provide examples for each.
8. How does inheritance support system implementation, and what advantages does it provide in object class specification?
9. Explain how CRC cards are updated during system analysis. What steps ensure they remain accurate as the system evolves?
10. Discuss the importance of fine-tuning classes, attributes, and relationships during **use case realization**.

20 Multiple-Choice Questions (System Analysis –

Inside View)

1. The **inside view** of system analysis focuses on:

1. Business rules only
2. Actors in the environment
3. System components, attributes, and collaboration
4. External requirements

2. Scenario analysis identifies system objects by:

1. Checking hardware specifications
2. Searching program libraries
3. Referring to names, attributes, or methods in use case documents
4. Interviewing only developers

3. In the “Borrow book” scenario, which property belongs to the **Member object**?

1. Author
2. Member ID
3. Year of publication
4. Book title

4. CRC stands for:

1. Class, Relationships, Context
2. Class, Responsibilities, Collaborators
3. Collaboration, Responsibilities, Components
4. Class, Requirements, Cases

5. CRC cards are useful for:

1. Defining actors only
2. Listing business rules
3. Assigning responsibilities and identifying collaborators
4. Modeling hardware systems

6. In the “Open Account” use case, which of the following is **NOT** a key component?

1. Customer Manager
2. Account Manager
3. Database
4. Marketing Manager

7. In the open account scenario, the **Bank Manager (BM)** provides:

1. Technical diagrams
2. Customer and account information
3. Database storage functions
4. Catalogue printing

8. Which UML diagram shows the **dynamic order of interactions** between system components?

1. Communication diagram
2. Sequence diagram
3. Class diagram
4. Activity diagram

9. In a communication diagram, messages are:

1. Displayed in time sequence
2. Shown as numbered interactions between objects
3. Hidden from actors
4. Always bidirectional

10. In an activity diagram, "swimlanes" are used to:

1. Represent state transitions
2. Separate responsibilities of different objects
3. Show inheritance
4. Mark aggregation

11. A class attribute that exists independently from another class represents:

1. Association
2. Aggregation
3. Composition
4. Inheritance

12. A class attribute that has **no meaning without another class** represents:

1. Association
2. Aggregation
3. Composition
4. Generalization

13. Inheritance in UML means:

1. Child classes ignore parent class methods
2. Child classes share and may modify parent class attributes and behaviors
3. Only one class can inherit from another
4. Parent classes cannot interact with child classes

14. Which method would most likely belong to a **Doctor class** in an object diagram?

1. +Medication()
2. +Treatment()
3. +BorrowBook()
4. +PayBill()

15. Which relationship is described as "whole-part integration" (e.g., car with wheels)?

1. Association

2. Aggregation/Composition
3. Generalization
4. Polymorphism

16. Which of the following describes **role-playing CRC cards**?

1. Assign responsibilities to objects in use cases
2. Define hardware specifications
3. Create activity diagrams automatically
4. Test sequence diagrams

17. In class diagrams, **generalization** means:

1. Classes share responsibilities equally
2. Child classes inherit attributes and behaviors of parent class
3. Objects always exist independently
4. Messages are passed between components

18. During **use case realization**, what is refined?

1. Network connections
2. Classes, attributes, and relationships
3. External actors only
4. Software tools

19. Which is **NOT** typically extracted from UML diagrams?

1. Class attributes
2. Class methods
3. Business budgets
4. Relationships

20. Why is specifying object class methods important?

1. They describe what actions the class can perform
2. They identify external environment actors
3. They replace inheritance
4. They define hardware configuration

5 Sample Exercises (OOAD System Analysis – Part 1 & 2)

Exercise 1: Use Case Identification (Outside View)

Consider the **Online Shopping System**.

1. Identify at least **3 main use cases** for the system.
2. Define the **actors** involved in each use case.
3. Draw a **UML Use Case Diagram** to represent the system.

Hint: Refer to Part 1 examples (Online Shopping Cases 1 & 2).

Exercise 2: Use Case Validation (Outside View)

Take the **ATM System** as an example.

1. Write down the main use cases a customer interacts with.
2. Check if each use case really represents a **goal of the actor** (customer).
3. Refine the use case diagram by removing steps that are not valid use cases.

Hint: Compare with the ATM use case validation example in Part 1.

Exercise 3: CRC Card Analysis (Inside View)

For the **"Open Account" use case** in a Bank Account Management System:

1. List the **candidate classes** involved (e.g., Customer Manager, Account Manager, Database).
2. Create **CRC cards** for each class, including:
   - o Responsibilities (R)
   - o Collaborators (C)
3. Role-play a short scenario using CRC cards to check object interactions.

Exercise 4: Class Diagram Construction (Inside View)

Using the **"Borrow Book" use case** in a Library System:

1. Identify at least 3 classes (e.g., Librarian, Member, Book).
2. Define their **attributes** and **methods**.
3. Draw a **UML Class Diagram** showing relationships (association, aggregation, inheritance).

Exercise 5: State and Activity Modeling (Integration of Part 1 & 2)

Choose one system (Coffee Cup, Storehouse, or Online Shopping).

1. Identify at least 2 **use cases**.
2. For one use case, draw a **sequence diagram** showing interactions between system objects.
3. For the same use case, draw an **activity diagram** with swimlanes to show responsibilities.
4. Describe one **state transition diagram** for an important object in the system.

10 Sample Exercises (Extended Version)

Exercise 1: Identifying Use Cases in an Online Shopping System (Outside View)

A retail company wants to build an **online shopping website** to expand its market. Customers should be able to browse products, place orders, and make payments. Staff will handle deliveries and manage transactions. The system must comply with government regulations and support different payment methods.

Tasks:

1.

1. Identify at least **five use cases** for this system (e.g., Browse Products, Place Order, Make Payment, Manage Delivery, Track Order).
2. Define the **actors** and describe their interactions with the system.
3. Draw a **UML Use Case Diagram** with relationships (include, extend, generalization).
4. Write a short explanation of how business rules influence the responsibilities of this system.

Exercise 2: Validating Use Cases for ATM System (Outside View)

An ATM machine provides services such as withdrawing money, depositing funds, checking balance, and printing mini-statements. However, not all steps in the interaction are true "use cases" (e.g., inserting card, entering PIN are actions, not goals).

Tasks:

1. List all steps a customer performs at an ATM.
2. From this list, determine which are valid **use cases** and which are just actions.
3. Draw two UML diagrams:
   - One "incorrect" diagram showing every step as a use case.
   - One "correct" diagram with validated use cases only.
4. Explain why it is important to distinguish between **system functions** and **actor goals**.

Exercise 3: Use Case Analysis for Storehouse Management (Outside View)

A restaurant has a storehouse where ingredients are ordered, received, delivered to the kitchen, and suppliers are paid. The system must support collaboration between the storekeeper, suppliers, manager, and chef.

Tasks:

1. Identify at least **four use cases** for storehouse management.
2. Draw a **Use Case Diagram** with the relevant actors.
3. Write a **use case scenario** (textual description) for "Order Ingredients" use case.
4. Discuss how external environment factors (e.g., suppliers, government regulations, customers) affect the responsibilities of the storehouse system.

Exercise 4: Coffee Cup as a System (Outside View)

A coffee cup is a simple object, but when considered as a system, it has several responsibilities such as filling, pouring, holding, and enabling drinking.

Tasks:

1. Identify the **actors** of the coffee cup system.
2. Create a **Use Case Diagram** showing all possible interactions.
3. Discuss how this example demonstrates the principle: "A system is a tool in its environment."
4. Suggest one other everyday object and perform the same analysis (identify actors, use cases, diagram).

Exercise 5: Scenario Analysis for Borrow Book Use Case (Inside View)

In a library system, members can borrow books by providing their ID and selecting a book. Librarians record the transaction, and the system checks borrowing limits.

Tasks:

1. From the “Borrow Book” scenario, identify candidate objects (e.g., Member, Book, Librarian).
2. Specify the **attributes and methods** for each object.
3. Draw a **Class Diagram** with relationships (association, aggregation, inheritance).
4. Explain how scenario analysis helps bridge **use case descriptions** and **system design**.

Exercise 6: CRC Card Role-Playing (Inside View)

You are analyzing the **“Open Account” use case** in a Bank Account Management System. The process involves identifying customers, account types, balances, and storing account details in the database.

Tasks:

1. Create **CRC cards** for the objects: Customer Manager, Account Manager, Database.
2. Assign **Responsibilities (R)** and **Collaborators (C)** for each object.
3. Perform a **role-playing activity**: simulate the interaction step by step.
4. Discuss why CRC role-playing is useful compared to just drawing diagrams.

Exercise 7: Sequence and Communication Diagrams (Inside View)

The “Open Account” use case requires collaboration among multiple objects. The Bank Manager provides information, the Customer Manager verifies data, the Account Manager processes account creation, and the Database stores results.

Tasks:

1. Draw a **Sequence Diagram** showing message order for this process.
2. Draw a **Communication Diagram** for the same scenario.
3. Compare the two diagrams and explain what unique insights each provides.
4. Discuss how these diagrams help refine class attributes and methods.

Exercise 8: Class Relationships Analysis (Inside View)

In a healthcare system, the following classes exist: Person, Doctor, Patient, Pharmacist, Medicine, Treatment. These classes have inheritance, association, aggregation, and composition relationships.

Tasks:

1. Identify which relationships apply between the classes (e.g., Doctor inherits Person, Treatment associates Doctor and Patient, Medicine aggregates into Prescription).
2. Draw a **Class Diagram** showing all relationships.
3. Explain the difference between **aggregation** and **composition**, with examples from the healthcare system.
4. Write a paragraph discussing why modeling class relationships is critical in OOAD.

Exercise 9: Activity and State Diagrams (Integration)

Choose the **“Borrow Book” use case** in a library system.

Tasks:

1. Draw an **Activity Diagram** with swimlanes to represent the responsibilities of Member, Librarian, and System.
2. Create a **State Diagram** for the Book object, showing states such as Available, Borrowed, Returned, Overdue.
3. Explain how activity diagrams help identify responsibilities, while state diagrams capture object behavior.
4. Suggest how these diagrams can be used together during implementation.

Exercise 10: Full Use Case Realization (Integration)

Consider a **Food Delivery App** that allows customers to place orders, make online payments, and track deliveries. Staff manages restaurants, riders, and payments.

Tasks:

1. Identify **at least 5 use cases** and draw a **Use Case Diagram**.
2. Perform **scenario analysis** for "Place Order." Identify candidate objects.
3. Create **CRC cards** for the main objects (Customer, Order, Restaurant, Rider, Payment System).
4. Draw **Class, Sequence, Activity, and State Diagrams** for the Place Order use case.
5. Write a short reflection: how do outside view (use cases & actors) and inside view (objects & diagrams) complement each other in OOAD?

10 Essay Questions (OOAD Design – Part 1)

1. Explain the main goals of **system design** in OOAD. How does design differ from analysis?
2. Define **component, subsystem, and package** in system design. Provide examples to illustrate their roles.
3. What are the key **objectives of a good design**? How do these objectives reduce the lifetime cost of a system?
4. Discuss the concepts of **coupling and cohesion**. Why is low coupling and high cohesion desirable in object-oriented design?
5. Provide examples of how **encapsulation, inheritance, and interaction** contribute to coupling. What problems can arise with high coupling?
6. Explain the role of **cohesion at class, method, and inheritance levels**. Why is method cohesion critical for reusability?
7. Describe the **five SOLID principles** (SRP, OCP, LSP, ISP, DIP). How do they guide object-oriented design?
8. Compare the "bad" and "good" examples of **Dependency Inversion Principle** provided in the slides. Why is abstraction important?
9. Discuss the processes of **factoring, partitioning, and layering** in system design. How do they help manage complexity?
10. What are the main aspects of **designing for security**? Explain how privacy, authentication, integrity, and safety influence system architecture.

20 Multiple-Choice Questions (OOAD Design – Part 1)

1. System design is best described as:

1. Coding the solution directly
2.

2. Defining architecture, components, interfaces, and data
3. Documenting requirements only
4. Writing test cases

2. In OO design, a **component** is:

1. A physical file in a package
2. The smallest unit (object class) solving basic problems
3. A collection of subsystems
4. A database table

3. A **subsystem** in OO design:

1. Solves a single method call
2. Is a set of collaborating components solving a use case
3. Is always equal to a package
4. Must be reused in all systems

4. A **package** is primarily used for:

1. Reuse and grouping of components
2. Only storing source code
3. Inheritance among classes
4. Managing runtime threads

5. A good design minimizes:

1. Execution speed
2. Lifetime cost of creating, using, and upgrading
3. Number of classes
4. Use of packages

6. Which of the following is a characteristic of a good design?

1. Large scope for changes
2. Small scope of corrections
3. High redundancy
4. Maximum interdependence

7. Coupling refers to:

1. Relationship between attributes and methods in a class
2. Interdependence between components
3. Generalization of classes
4. Aggregation of objects

8. Cohesion refers to:

1. How tightly attributes and methods within a component work together
2. Dependency among subsystems
3. Relation between packages
4. Relationship between actors and use cases

9. Which of the following increases coupling risk?

1. High cohesion
2. Strong dependency on attribute types
3. Abstraction
4. Information hiding

10. Which statement about inheritance and coupling is true?

1. Subclasses never depend on parent classes
2. Subclasses always depend on inherited content
3. Parent classes depend on child classes
4. Inheritance eliminates coupling

11. Interaction coupling is caused by:

1. Message passing between objects
2. Database schemas only
3. Method overloading
4. Class generalization

12. High class cohesion means:

1. Class has many unrelated methods
2. All methods/attributes are required to represent one concept
3. Multiple classes are joined together
4. Class is completely abstract

13. The **Liskov Substitution Principle (LSP)** means:

1. Parent classes must replace child classes
2. Subclasses must be substitutable for parent classes
3. Subclasses must avoid using parent features
4. Only one subclass can exist

14. The **Interface Segregation Principle (ISP)** states:

1. Clients should not depend on methods they do not use
2. All clients must share the same interface
3. Interfaces must be large to cover all cases
4. Dependencies must always be direct

15. The **Dependency Inversion Principle (DIP)** requires:

1. High-level modules to depend on low-level ones
2. Both high and low-level modules depend on abstractions
3. Only low-level modules use abstractions
4. Remove inheritance completely

16. Factoring in design means:

1.

Separating commonalities into reusable classes
2. Removing abstract classes
3. Always merging subsystems
4. Designing with no inheritance

17. Abstraction in factoring refers to:

1. Creating higher-level ideas from multiple lower-level classes
2. Adding unnecessary details
3. Eliminating polymorphism
4. Removing generalization

18. Partitioning in system design is used to:

1. Group tightly coupled classes into subsystems
2. Create packages randomly
3. Ensure classes exist independently
4. Increase interdependency

19. Layering in OO design helps to:

1. Increase complexity
2. Reduce complexity and increase reuse
3. Eliminate the need for abstraction
4. Avoid partitioning

20. In **designing for security**, integrity refers to:

1. Preventing unauthorized access
2. Ensuring information is not altered or damaged
3. Hiding all data permanently
4. Reducing coupling

10 Essay Questions (OOAD Design – Part 2)

1. Define a **subsystem** in OOAD design. How does it differ from a package, and why is it not intended for reuse?
2. Explain the process of mapping an **analysis model to a design model**. How are communication diagrams used in partitioning a system into subsystems?
3. Discuss the role of a **boundary class (interface)** in subsystem design. Why should only the boundary be visible externally?
4. Describe the major responsibilities of **interfaces** in system design. How do they protect the system from external impacts?
5. What are the key benefits of **abstract interfaces**? Provide examples to illustrate how they enforce consistency and enable polymorphism.
6. Compare **abstract interfaces** and **abstract data types (ADT)**. How do both support reusability and implementation independence?
7. Discuss the importance of **message objects** in software development. Why is encapsulating message structures into objects preferable to using raw data structures?
8. Explain the strengths and weaknesses of different **database technologies** (RDBMS, ORDBMS, OODBMS, NoSQL) in supporting object-oriented systems.
9.

9. Describe how **multiple inheritance** can be mapped into relational and object-relational databases. What rules are applied in each case?
10. Explain the steps for mapping **domain objects to RDBMS**. How are attributes, associations, and methods represented in relational tables?

20 Multiple-Choice Questions (OOAD Design – Part 2)

1. A **subsystem** is:

1. A reusable package of classes
2. A group of cooperating objects performing system tasks
3. Always equivalent to a database table
4. Only used for security

2. Which statement about **subsystem design** is true?

1. Subsystems are primarily intended for reuse
2. Subsystems directly handle external input/output
3. Subsystems typically correspond to solving use cases
4. Subsystems always depend on boundary classes

3. Factoring in subsystem design means:

1. Splitting one subsystem into many smaller unrelated parts
2. Factoring out similarities into superclasses or subclasses
3. Removing inheritance from the model
4. Copying attributes across classes

4. In subsystem design, only the ____ should be visible externally:

1. Control classes
2. Boundary classes
3. All internal classes
4. Database classes

5. The role of an interface includes all except:

1. Security checks
2. Encoding/decoding input/output
3. Directly implementing all system logic
4. Detecting errors

6. Which SOLID principle is most related to interface design?

1. SRP
2. OCP
3. LSP
4. ISP

7. Abstract interfaces contain:

1. Both code and implementation logic
2. Only abstract methods without body
3. Only concrete methods
4. Only attributes

8. Which of the following is a benefit of **abstract interfaces**?

1. Hide polymorphism
2. Enforce consistency
3. Remove dependency inversion
4. Avoid testability

9. An **Abstract Data Type (ADT)** defines:

1. Both the operations and their implementation
2. Only what operations are performed, not how
3. Only the algorithms to be used
4. No reusable interface

10. Why are ADTs important?

1. They reduce reusability
2. They provide implementation-independent view
3. They enforce single inheritance only
4. They eliminate polymorphism

11. A **message** in OO design is best described as:

1. Data transferred between objects
2. A private attribute
3. A UML diagram
4. A stored procedure

12. Why should messages be encapsulated as objects?

1. To reduce compiler errors
2. To allow hiding of data structure details
3. To prevent polymorphism
4. To eliminate all public methods

13. Which database type best supports object-orientation directly?

1. RDBMS
2. ORDBMS
3. OODBMS
4. NoSQL

14. RDBMS primarily supports:

1. Complex object inheritance
2.

2. Referential integrity on simple data
3. Native object storage
4. Multi-valued attributes directly

15. ORDBMS extends RDBMS to:

1. Store objects in relational structures
2. Eliminate all tables
3. Fully support inheritance
4. Remove SQL usage

16. Which database type is still maturing but handles complex data?

1. OODBMS
2. NoSQL
3. RDBMS
4. Indexed files

17. Rule for multiple inheritance in ORDBMS:

1. Flatten inheritance hierarchy
2. Convert additional inheritance to association
3. Remove all superclasses
4. Use only single inheritance

18. Rule for multiple inheritance in RDBMS:

1. Add associations
2. Flatten hierarchy by copying attributes/methods
3. Implement polymorphism directly
4. Use abstract interfaces

19. Mapping domain objects to RDBMS requires:

1. Ignoring methods and relationships
2. Mapping each concrete class to a table
3. Using one table for the whole system
4. Only storing attributes

20. Multivalued attributes in RDBMS are mapped to:

1. New columns in the same table
2. New tables with one-to-many relations
3. Foreign keys only
4. Stored procedures

10 Extended Design Exercises (OOAD Design – Part 1 & 2)

Exercise 1: Subsystem Identification and Partitioning

A university wants to build a **Learning Management System (LMS)**. The system supports student registration, course management, online exams, and grade reports.

Tasks:

1. Identify at least **four subsystems** (e.g., Registration, Course Management, Examination, Reporting).
2. Use **communication diagrams** to show how subsystems interact for the "Take Exam" use case.
3. Define subsystem boundaries and show which classes belong to each subsystem.
4. Discuss how **factoring and partitioning** reduce coupling in this system.

Exercise 2: Applying Coupling and Cohesion Principles

A hospital system has modules for **Patient Records, Billing, Pharmacy, and Appointment Scheduling**.

Tasks:

1. Identify possible coupling issues among these modules.
2. Redesign the modules applying **low coupling and high cohesion**.
3. Draw **class diagrams** before and after redesign.
4. Write an explanation of how your new design improves **maintainability and reusability**.

Exercise 3: Package and Subsystem Design

A large e-commerce platform needs to organize classes for **User Management, Product Catalog, Shopping Cart, Payment, and Delivery**.

Tasks:

1. Group related classes into **packages** for reuse.
2. Partition the system into **subsystems**, each solving one use case (e.g., Place Order, Manage Delivery).
3. Draw a **package diagram** and a **subsystem diagram**.
4. Discuss the trade-offs between using **packages vs. subsystems**.

Exercise 4: Interface Design for ATM System

An ATM provides services such as withdrawing, depositing, and checking balance.

Tasks:

1. Identify the **boundary classes (interfaces)** visible to external actors.
2. Design **abstract interfaces** for services provided by the ATM.
3. Show how **Interface Segregation Principle (ISP)** applies here.
4. Write a short reflection on how well-designed interfaces improve **security and usability**.

Exercise 5: Applying SOLID Principles in a Library System

The library system includes classes **Book, Member, Librarian, LoanRecord, Payment**.

Tasks:

1. Identify violations of **SOLID principles** in a poorly designed version (e.g., one class doing too much).
2. Redesign the system to satisfy **SRP, OCP, LSP, ISP, DIP**.
3. Draw **before/after class diagrams**.
4. Explain how each SOLID principle guided your redesign.

Exercise 6: Abstract Interfaces vs. ADTs

A ride-hailing app (like Grab/Uber) needs to design the **Payment System** supporting multiple methods (Credit Card, E-Wallet, Cash).

Tasks:

1. Design an **abstract interface** for Payment.
2. Define possible **ADT representations** for storing payment transactions.
3. Discuss differences between **abstract interface** and **ADT** in this design.
4. Show a **UML class diagram** applying the **Dependency Inversion Principle (DIP)**.

Exercise 7: Database Design with OOAD Mapping

A **Hotel Management System** must be mapped into a relational database. Classes include **Guest, Room, Booking, Payment**.

Tasks:

1. Identify attributes and methods for each class.
2. Map the classes to **RDBMS tables**.
3. Show how **one-to-many and many-to-many associations** are implemented in the database.
4. Draw both the **class diagram** and the corresponding **ER diagram**.

Exercise 8: Managing Multiple Inheritance in Design

A university system has a class **TeachingAssistant** that inherits from both **Student** and **Staff**.

Tasks:

1. Show how multiple inheritance is represented in the **class diagram**.
2. Discuss problems when mapping to **RDBMS** and **ORDBMS**.
3. Apply the correct mapping rules:
    - o Flattening in RDBMS
    - o Converting extra inheritance into association in ORDBMS
4. Write a conclusion on when multiple inheritance should be avoided.

Exercise 9: Message Objects in Online Banking

In online banking, messages are exchanged for actions like **Transfer Money, Pay Bill, Check Balance**.

Tasks:

1. Define a **Message class hierarchy** to encapsulate these communications.
2. Show a **sequence diagram** for "Transfer Money" using message objects.
3.

3. Compare using message objects vs. raw data structures.
4. Explain how message objects improve **encapsulation and flexibility**.

Exercise 10: Full OOAD Design Case Study (Food Delivery App)

A Food Delivery App must support: customer registration, browsing restaurants, placing orders, payments, delivery tracking.

Tasks:

1. Identify at least **five use cases** and draw a **use case diagram**.
2. Partition the system into **subsystems** (e.g., User Management, Restaurant, Order, Payment, Delivery).
3. Design **interfaces** for each subsystem, ensuring low coupling.
4. Draw **class, sequence, activity, and state diagrams** for "Place Order".
5. Write a short essay explaining how **design principles (SOLID, cohesion, coupling, layering, security)** are applied.

Cấu trúc mỗi bài tập

- **Mô tả hiện trạng & bối cảnh** (chi tiết, 1–2 đoạn văn).
- **Vấn đề tồn tại / nhu cầu hệ thống**.
- **Nhiệm vụ của sinh viên**:
  - Xác định actors & use cases.
  - Vẽ use case diagram.
  - Tạo activity, state, sequence diagrams.
  - Xác định lớp miền (domain objects).
  - Thiết kế hệ thống (subsystems, packages, interfaces).
  - Đề xuất kiến trúc DB (mapping UML → RDBMS/ORDBMS).
  - Áp dụng SOLID, coupling/cohesion, security…

1. Quản lý Bệnh viện

Một bệnh viện vừa và nhỏ hiện quản lý hồ sơ bệnh nhân, lịch hẹn, đơn thuốc và thanh toán **bằng giấy và Excel**. Việc tìm kiếm thông tin bệnh nhân mất nhiều thời gian, dữ liệu thuốc không đồng bộ giữa bác sĩ và nhà thuốc, dễ dẫn đến sai sót.

Nhiệm vụ:

- Phân tích yêu cầu hệ thống quản lý bệnh viện (actors: bệnh nhân, bác sĩ, nhân viên y tế, thu ngân, nhà thuốc).
- Vẽ use case diagram (khám bệnh, kê đơn, cấp thuốc, thanh toán, xuất báo cáo).
- Mô hình hóa quy trình "khám & kê đơn thuốc" bằng activity + sequence diagram.
- Xác định domain objects và thiết kế class diagram.
- Thiết kế subsystems (Quản lý bệnh nhân, Quản lý thuốc, Quản lý thanh toán).
- Đề xuất sơ đồ cơ sở dữ liệu và mapping các lớp vào RDBMS.
- Áp dụng nguyên tắc **ISP, DIP** để tách interfaces.

2. Thư viện Số (Digital Library)

Một trường đại học muốn xây dựng hệ thống thư viện số, cho phép sinh viên **mượn sách online, tải tài liệu, đặt**

sách, và trả sách tự động qua kiosk. Hiện tại mọi thứ làm thủ công, phiếu mượn bị thất lạc nhiều.

Nhiệm vụ:

- Xác định actors: sinh viên, thủ thư, giảng viên, kiosk.
- Use case diagram (tìm kiếm sách, mượn, trả, gia hạn, xem lịch sử).
- Sequence diagram cho "mượn sách qua kiosk".
- Thiết kế domain model (Book, LoanRecord, User, Payment).
- Phân tích coupling & cohesion, đề xuất tái cấu trúc.
- Mapping domain model sang CSDL quan hệ (1-nhiều: user–loan, book–loan).

3. Hệ thống Quản lý Bán hàng

Một cửa hàng bán lẻ hiện nhập đơn hàng và hóa đơn **bằng giấy**, việc kiểm kho không đồng bộ. Muốn phát triển hệ thống bán hàng có **chức năng bán hàng, quản lý kho, giao hàng, báo cáo doanh thu**.

Nhiệm vụ:

- Use case diagram cho actors: nhân viên bán hàng, khách hàng, thủ kho, kế toán.
- Activity diagram "xử lý đơn hàng".
- Thiết kế các lớp: Order, Product, Customer, Invoice, Delivery.
- Chia subsystem: Bán hàng, Kho, Giao hàng, Báo cáo.
- Vẽ package diagram và sequence diagram.
- Thực hiện mapping database (Order, OrderDetail, Customer…).

4. Hệ thống Giao đồ ăn Online

Hiện nay các cửa hàng nhỏ muốn kết nối khách hàng với shipper nhưng chưa có nền tảng. Hệ thống cần hỗ trợ **đặt món, thanh toán, theo dõi đơn hàng**.

Nhiệm vụ:

- Use case diagram với actors: khách hàng, nhà hàng, shipper, admin.
- Sequence diagram cho "đặt món và giao hàng".
- Class diagram: Order, Menu, Restaurant, Driver, Payment.
- Phân tích thiết kế interface Payment với nhiều phương thức (card, ví điện tử, tiền mặt).
- Mapping sang RDBMS, xử lý multivalued attribute (OrderItems).

5. Hệ thống Ngân hàng Trực tuyến

Ngân hàng muốn xây dựng ứng dụng online để khách hàng **chuyển khoản, thanh toán hóa đơn, gửi tiết kiệm, xem lịch sử giao dịch**.

Nhiệm vụ:

- Use case diagram: Customer, Teller, Admin.
- Activity diagram "chuyển tiền online".
- Domain classes: Account, Transaction, Customer, Loan.
- Thiết kế messages (TransferRequest, TransferResponse).
- Thực hiện database mapping với quan hệ 1-nhiều Account–Transaction.
- Áp dụng nguyên tắc **DIP** trong Payment Service.

6. Hệ thống Quản lý Bán vé Xe/Phim

Hiện nay vé bán thủ công → gây trùng lặp chỗ ngồi, khó kiểm soát doanh thu.

Nhiệm vụ:

- Use case: chọn lịch chiếu/chuyến xe, đặt vé, thanh toán, hủy vé.
- Activity diagram "đặt vé".
- Class diagram: Ticket, Customer, Schedule, Payment.
- Chia subsystem: Quản lý lịch, Quản lý khách hàng, Quản lý thanh toán.
- Mapping database: Ticket (seat, time, status).

7. Hệ thống Quản lý Học phí

Trường đại học muốn xây dựng hệ thống thu học phí tự động thay vì thu qua giấy.

Nhiệm vụ:

- Use case: nộp học phí, xem công nợ, xuất biên lai.
- Sequence diagram "nộp học phí online".
- Class diagram: Student, FeeRecord, Payment, Receipt.
- Phân tích interface Payment (ISP).
- Mapping CSDL: FeeRecord (nhiều → 1 Student).

8. Hệ thống Quản lý Vận tải (Logistics)

Một công ty vận tải cần quản lý đội xe, đơn hàng, lộ trình vận chuyển. Hiện làm qua Excel nên thiếu chính xác.

Nhiệm vụ:

- Use case: tạo đơn hàng, gán xe, theo dõi trạng thái.
- State diagram "vòng đời đơn hàng".
- Class diagram: Shipment, Vehicle, Driver, Route.
- Subsystems: Quản lý đơn hàng, Quản lý đội xe, Quản lý vận hành.
- Mapping CSDL quan hệ với nhiều-to-nhiều: Vehicle–Route.

9. Hệ thống Quản lý Bán vé Máy bay

Hiện tại hệ thống đặt vé máy bay thủ công, khách hàng phải ra quầy.

Nhiệm vụ:

- Actors: khách hàng, nhân viên, admin.
- Use case: tìm chuyến, đặt vé, thanh toán, check-in online.
- Activity diagram "check-in online".
- Domain objects: Flight, Ticket, Passenger, BoardingPass.
- Subsystem: Booking, Payment, Check-in.

10. Hệ thống Quản lý Khách sạn

Khách sạn muốn quản lý đặt phòng, thanh toán, và dịch vụ (ăn uống, spa).

Nhiệm vụ:

- Use case: đặt phòng, hủy phòng, thanh toán dịch vụ.
- State diagram “vòng đời đặt phòng”.
- Class diagram: Room, Booking, Customer, Service, Invoice.
- Mapping DB: Booking (1-nhiều Room–Service).