UNIT–2 : Basic Behavioural and Architectural Modeling (PYQ Solution)

AKTU · Object Oriented System Design with C++ (BCS054) · Solutions by **Chirag Jain**, Ycotes

SECTION A Answers (≈70 Words Each)

Q1. Define association and aggregation. (Jan 2026)

An association is a structural relationship showing that objects of one class are connected to objects of another, shown as a solid line, e.g. Teacher teaches Student. Aggregation is a special association representing a whole–part ("has-a") relationship in which the part can exist independently of the whole, shown with a hollow diamond at the whole, e.g. Department has Professors.

Q2. List the features of Component Diagram. (2023)

- **Shows the physical, implementation-level parts of a system: source files, libraries, executables.**
- **Each component is a replaceable unit with well-defined provided and required interfaces.**
- **Shows dependencies between components.**
- **Supports modelling of architecture, reuse and deployment planning.**
- **Components can be grouped into packages and later mapped to nodes in a deployment diagram.**

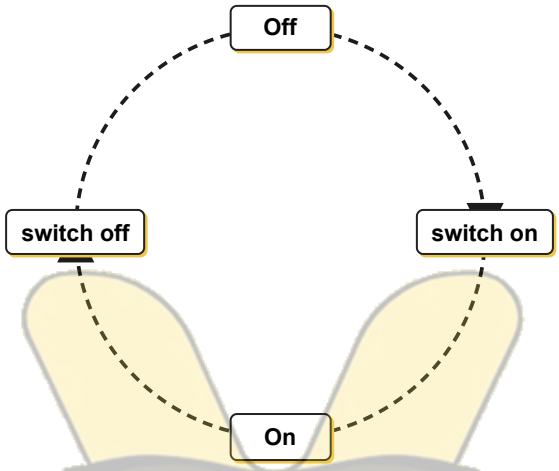
Q3. Explain the existence of swimlanes in activity diagram. (2023)

Swimlanes (partitions) divide an activity diagram into vertical or horizontal columns, each representing the person, department or class responsible for the activities placed in it. They show who does what and how control passes across responsibilities. Example: in order processing, lanes for Customer, Sales and Warehouse.

Q4. Draw a state diagram for electric bulb. (2022)

States: Off and On (plus optional Fused).

```
(●) → [Off] --switch on--> [On] --switch off--> [Off]
        |                    |
        +---- bulb burns ----+--> [Fused] → (◉)
```



State diagram of an electric bulb

The initial state is Off. The event "switch on" moves it to On, "switch off" returns it to Off, and "burn out" moves either state to Fused, the final state.

Q5. Define and Differentiate Link and Association with example. (2021)

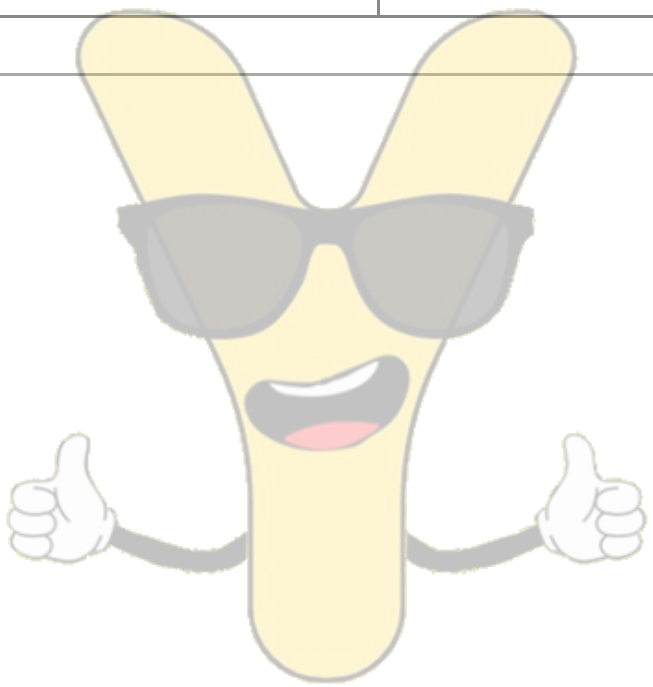
A link is a connection between two objects; an association describes a group of links with common structure and meaning between classes.

Link	Association
Object-level, run-time instance	Class-level, design-time
Example: Ravi studies in Delhi University	Example: Student studies in University
Drawn in object diagram	Drawn in class diagram

Q6. Define and Differentiate Generalization and Inheritance with example. (2021)

Generalization is the UML relationship in which a general class (superclass) is related to a more specific class (subclass), an "is-a" relation found by factoring common features. Inheritance is the mechanism by which the subclass automatically receives attributes and operations of the superclass.

Generalization	Inheritance
Modelling concept in analysis/design	Implementation mechanism in language
Bottom-up: extract common parts	Top-down: derive and reuse
Example: `Car` and `Bike` generalised to `Vehicle`	Example: `class Car : public Vehicle`



SECTION B Answers (≈250 Words Each)

Q1. Describe sequence diagrams in detail. Explain synchronous and asynchronous messages, call-back mechanisms, and broadcast messages. (Jan 2026)

A **sequence diagram** is an interaction diagram that shows how objects communicate through **messages arranged in time order**. Objects are placed across the top; each has a vertical **lifeline**; horizontal arrows are messages; time flows downward. A thin **activation bar** shows when an object is active.



Objects exchanging messages in a login sequence

Message types

- **Synchronous:** the sender sends a message (solid line, filled arrowhead) and **waits** until the receiver finishes and returns control (dashed return arrow). Example: `server → db : query()`.
- **Asynchronous:** the sender (open or half arrowhead) **continues without waiting**. Example: sending an e-mail or posting to a queue.
- **Call-back:** the caller passes a reference so the callee can **call it back later** when work completes. A asks B to start a task (asynchronous), and B later invokes `A.done()`. Used in event handling and listeners.
- **Broadcast:** one message sent to **many receivers** at once (e.g. a notification to all subscribers), drawn as several arrows from the same point, or as a message to a group.
- Others: **create** (new object appears mid-diagram), **destroy** (X on lifeline), **self message**, **return**.

Combined fragments: `alt` (if/else), `opt`, `loop`, `par` express conditions and loops.

Steps to draw: identify the scenario, list participating objects, place them horizontally, add messages in order, then add activations and returns.

Aspect	Synchronous	Asynchronous
Sender	Blocks	Continues
Arrow	Filled head	Open head
Return	Required	Not required

In conclusion, sequence diagrams make dynamic behaviour of a use case clear and are useful for design and testing.

Q2. Discuss the purpose of Use Case Diagram and explain its different notations. (2023)

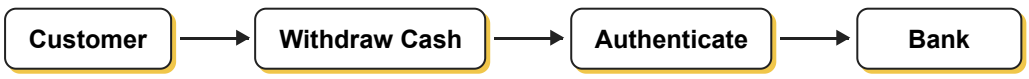
A **use case diagram** shows the functional requirements of a system from the user's viewpoint: what the system does, not how.

Purpose

- Capture and communicate requirements with customers.
- Define system **scope and boundary**.
- Identify **actors** and their goals.
- Serve as a base for design, testing and user manuals.

Notations

- **Actor:** stick figure outside the system (user or external system).
- **Use case:** oval with a verb phrase (Withdraw Cash).
- **System boundary:** rectangle containing use cases.
- **Association:** line between actor and use case.
- **Include:** dashed arrow `<<include>>`, mandatory shared behaviour (Withdraw includes Authenticate).
- **Extend:** dashed arrow `<<extend>>`, optional or conditional behaviour (Withdraw extends to Print Receipt).
- **Generalization:** hollow-triangle arrow between actors or use cases.



ATM use case: actors and use cases

Example: for an ATM, actors are Customer and Bank; use cases are Withdraw, Deposit, Check Balance, each including Authenticate. In conclusion, use case diagrams give a simple, user-centred overview of system functionality.

Q3. What is Data Abstraction? How is it different from encapsulation? Explain with proper example. (2021)

Data abstraction is showing only the essential features of an object and hiding the unnecessary details; it works at the **design level** ("what" an object does). **Encapsulation** is wrapping data and functions into one unit and controlling access with access specifiers; it works at the **implementation level** ("how" it is protected).

Abstraction	Encapsulation
Hides complexity	Hides data and binds it with code
Achieved with abstract classes and interfaces	Achieved with classes and private/public
Design-level	Implementation-level
Focus on what	Focus on how and access

Example

```
class Car {
    int fuel;                  // hidden (encapsulation)
    void burn() { fuel--; }    // internal detail
public:
    void start() { burn(); }   // abstraction: driver just calls start()
};
```

A driver uses `start()` without knowing the engine mechanics (abstraction), and `fuel` cannot be changed directly (encapsulation). Abstraction defines the interface; encapsulation enforces it.

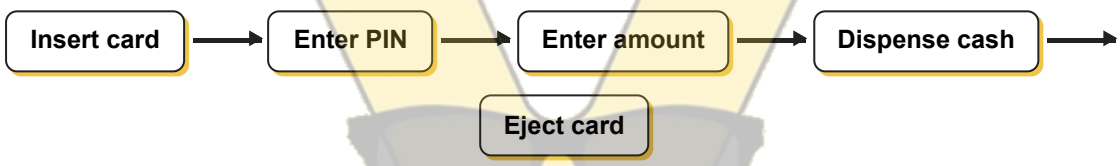
SECTION C Answers (≈500 Words Each)

Q1. Explain activity diagrams and state machine diagrams with suitable illustrations. (Jan 2026)

Activity diagram: shows the **flow of control from activity to activity**, like an advanced flowchart for business processes and operations; it supports decisions, parallelism and responsibilities.

Elements: initial node (filled circle), activity (rounded rectangle), action flow (arrow), decision/merge (diamond), fork/join (thick bar for parallel flows), final node (bullseye), **swimlanes** (who is responsible), object flow.

Example, ATM withdrawal: Insert card → Enter PIN → [valid?] → yes: Enter amount → [balance enough?] → Dispense cash → Eject card; no: Show error → Eject card.

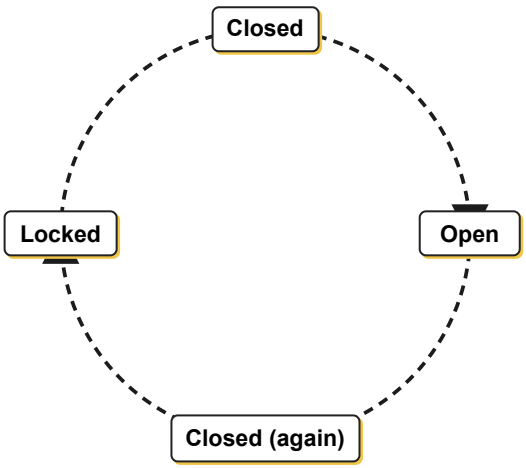


Activity diagram of ATM withdrawal

State machine diagram: models the **life history of one object** as states and transitions triggered by events.

Elements: state (rounded box), initial and final states, **transition** `event [guard] / action`, entry/exit/do activities, composite (nested) states, history state.

Example, a Door: Closed --open--> Open --close--> Closed; Closed --lock--> Locked --unlock--> Closed.



State machine of a door

Activity diagram	State machine diagram
Flow of activities, many objects	States of a single object
Nodes are activities	Nodes are states
Transition when activity completes	Transition on an event
Models processes, workflows	Models reactive objects

Uses: activity diagrams for use case realisation and business workflow; state diagrams for objects with complex behaviour such as a traffic light or order status.

In conclusion, activity diagrams show how work flows across the system, while state machines show how an individual object reacts over time.

Q2. Describe architectural modeling. Explain component diagrams and deployment diagrams and their significance in system design. (Jan 2026)

Architectural modeling describes the high-level structure of a system: its major parts, their interfaces, and how they are mapped to hardware. It addresses the **implementation view** (components) and **deployment view** (nodes) of UML, and helps manage complexity, reuse and scalability.



From code to deployed system

Component diagram

- A **component** is a modular, replaceable physical part (executable, library, file, database) that exposes **provided interfaces** (lollipop) and uses **required interfaces** (socket).
- Shows components and **dependencies** among them.
- Used to model source-code structure, executable releases, physical databases and adaptable systems.
- Example: an e-commerce system with UI, Order Service, Payment Gateway and Database components.

Deployment diagram

- Shows the **run-time configuration** of hardware **nodes** (server, PC, device, shown as 3D boxes) and the components deployed on them, with **communication links** (HTTP, TCP/IP).

- Used to model embedded, client–server and distributed systems.
- Example: Browser node → Web Server node (Order Service) → Database Server node.

Component diagram	Deployment diagram
Logical software modules	Physical hardware nodes
Dependencies and interfaces	Network connections
Implementation view	Deployment view

Significance

- Plans development by dividing work among teams.
- Shows impact of changing a component.
- Helps hardware sizing, performance and security planning.
- Gives clients and operators a picture of installation.

In conclusion, component and deployment diagrams link software design to its real physical realisation.

Q3. What do we mean by a collaboration diagram? Illustrate its significance, explain various terms and symbols used, draw one for reserving a room in a hotel from its website, and show how polymorphism is described using it. (2023, 2021)

A **collaboration (communication) diagram** is an interaction diagram that shows the **objects, the links between them, and the messages exchanged**, organised by structure rather than time.

Symbols and terms

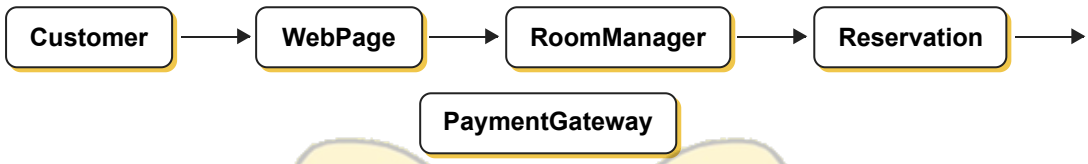
- **Object:** rectangle `name : Class`.
- **Link:** a line joining two objects that communicate.
- **Message:** arrow with a **sequence number** and operation name, e.g. `2: checkAvailability()`.
- **Actor:** stick figure that starts the interaction.
- **Self link / self message** and **multi-object** (stack of rectangles).
- **Return value:** `x := op()`; **iteration** `*`; **guard** `[condition]`.

Significance

- Shows object organisation and links clearly.
- Good for discovering which classes must be associated.
- Equivalent to a sequence diagram (convertible), but more compact.

Hotel room reservation

```
Customer --1: selectDates()--> :WebPage
:WebPage --2: checkAvailability()--> :RoomManager
:RoomManager --3: getRooms()--> :Room
:WebPage --4: reserve(room)--> :Reservation
:Reservation --5: charge()--> :PaymentGateway
:Reservation --6: confirm()--> Customer
```



Hotel room reservation collaboration

Polymorphism: the same message is sent to objects of different classes and each responds in its own way. Example: `Bill` sends `2: calcFee()` to a `:Room` object. If the receiver is `:DeluxeRoom`, it executes its own `calcFee()`; if `:StandardRoom`, a different one. The collaboration diagram shows the receiver as a base-class role with alternate subclass objects bound to it, each with the same numbered message.

In conclusion, collaboration diagrams effectively show structural organisation and message flow, including polymorphic calls.

Q4. A farmer wants to cross the river in a boat along with a bag of grass, a goat and a lion. Only one thing can be carried at a time. If the goat is left alone with the grass the grass will be eaten. If the goat is left alone with the lion the goat will be eaten. (i) Prepare a scenario in which everything is safely transported. (ii) Prepare the event trace diagram. (2022)

(i) Scenario (L = left bank, R = right bank). The unsafe pairs are goat+grass and goat+lion when the farmer is absent.

Step	Farmer's action	Left bank	Right bank
0	start	F, Goat, Lion, Grass	–
1	takes goat across	Lion, Grass	F, Goat
2	returns alone	F, Lion, Grass	Goat
3	takes lion across	Grass	F, Lion, Goat
4	brings goat back	F, Goat, Grass	Lion
5	takes grass across	Goat	F, Grass, Lion
6	returns alone	F, Goat	Grass, Lion
7	takes goat across	–	F, Goat, Lion, Grass

Check: lion and grass may be left together (lion does not eat grass); at no step is the goat left with the lion or the grass. Total: 7 crossings.

(ii) Event trace (objects: Farmer, Boat, Goat, Lion, Grass; events in time order):

```
1. Farmer -> Goat   : load
2. Farmer -> Boat   : crossToRight
3. Farmer -> Goat   : unload
4. Farmer -> Boat   : returnToLeft (empty)
5. Farmer -> Lion   : load ; crossToRight ; unload
6. Farmer -> Goat   : load ; returnToLeft ; unload
7. Farmer -> Grass  : load ; crossToRight ; unload
8. Farmer -> Boat   : returnToLeft (empty)
9. Farmer -> Goat   : load ; crossToRight ; unload
```



Order of the main trips

All four are finally on the right bank, nothing is eaten, and the event trace shows the sequence of events between the objects.

Q5. What is use case driven OOA? How is it different from OOD? Explain OOA process with the help of a diagram in the unified approach. (2021)

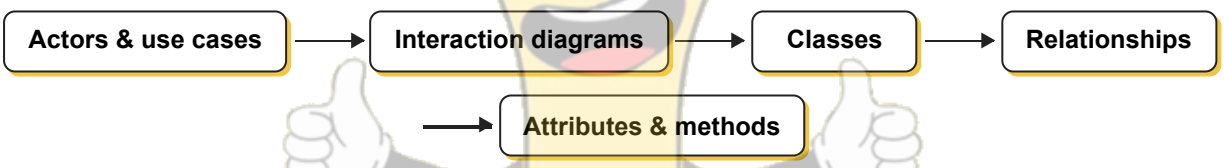
Use case driven OOA is object-oriented analysis in which **use cases** (scenarios of how actors use the system) are the primary source of requirements and drive the discovery of classes, relationships and behaviour. Every later model is traced back to a use case.

OOA vs OOD

OOA	OOD
Studies the problem domain: **what** the system does	Defines the solution: **how** it will be done
Identifies domain classes and requirements	Adds design classes, interfaces, persistence, UI
Technology independent	Considers language, hardware, performance
Output: use case, domain class models	Output: detailed class, interaction, component diagrams

OOA process in the Unified approach

1. **Identify actors** and develop use cases / scenarios.
2. **Develop interaction (sequence/collaboration) diagrams** for each use case.
3. **Classify and identify classes** from nouns (candidate classes), responsibilities and collaborations.
4. **Define relationships:** association, aggregation, generalization.
5. **Define attributes and methods** of each class.
6. **Iterate and refine**, adding the user interface, packages and state diagrams.



OOA process in the Unified approach

Techniques for class discovery: common class patterns, use case driven, CRC cards, noun phrase approach.

Example: in a library system, use cases Issue Book and Return Book yield classes Member, Book, Loan, Librarian and relationships Member borrows Book.

In conclusion, use case driven OOA ensures the system is built around actual user needs, and OOD then turns this model into an implementable design.

Q6. List the properties of a state chart diagram. Draw a state chart for a coin vending machine present at a railway station. (2021)

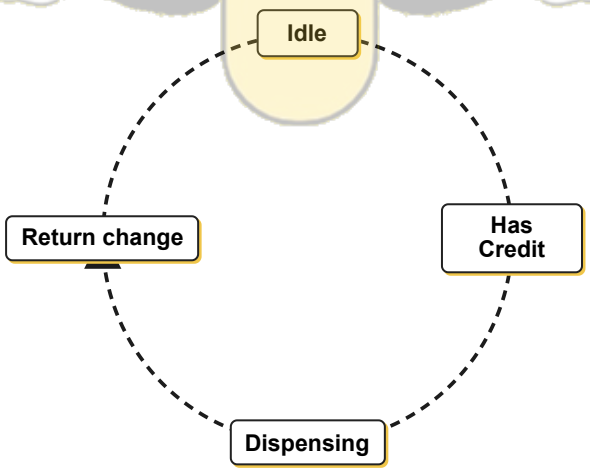
Properties of a state chart diagram

- Models the **dynamic behaviour of one object** over its life.
- Consists of **states**, **transitions**, events, guards and actions.
- Has one **initial** state and zero or more **final** states.
- Supports **nested (composite) states** and **concurrent** regions.
- Transitions are labelled `event [guard] / action`.
- States may have **entry, exit and do** activities.
- Shows history states for returning to the previous state.

Coin vending machine (e.g. platform ticket or water)

States: Idle, Coin Inserted (waiting), Item Selected, Dispensing, Out of Stock.

From	Event [guard] / action	To
Idle	insertCoin / add credit	Has Credit
Has Credit	insertCoin / add credit	Has Credit
Has Credit	select [credit ≥ price and stock > 0]	Dispensing
Has Credit	select [credit < price] / show "insert more"	Has Credit
Has Credit	cancel / refund	Idle
Dispensing	dispensed / return change	Idle
Idle	stock = 0	Out of Stock
Out of Stock	refill	Idle



Main cycle of the coin vending machine

The start dot enters Idle; the machine loops through Idle → Has Credit → Dispensing → Idle for each sale. In conclusion, the state chart documents all possible

behaviours and exceptional events of the machine.

