Object Oriented Analysis and Design Ques. Bank Solution

- Aditya Dhiman

Q1: What is a UML Activity Diagram?

Answer:
A UML Activity Diagram is a flowchart that represents the flow of control or sequence of activities in a system or process. It models the dynamic aspects of the system by showing the flow of activities between various actions.

Q2: State the role and application of activity diagrams.

Answer:
Activity diagrams help model the workflow of a system, process, or use case. They are useful in documenting business processes, representing complex algorithms, and visualizing parallel or conditional workflows.

Q3: What are UML state machine diagrams?

Answer:
UML State Machine Diagrams represent the behavior of a system or object in terms of states and transitions. They show how an object responds to events by transitioning between different states.

Q4: Define Event, State, and Transition.

Answer:

- **Event:** An occurrence that triggers a change in the state of an object.
- **State:** A specific condition in which an object exists at a given time.
- **Transition:** The movement from one state to another, triggered by an event.

Q5: What are deployment diagrams?

Answer:
Deployment diagrams are UML diagrams used to model the physical deployment of artifacts

(software components, files) onto hardware nodes such as servers, databases, or client machines.

Q6: What is a component diagram?

Answer:
A component diagram is a UML diagram that illustrates the organization and dependencies between different software components in a system, showing how components interact to form the system.

Q7: What is a UML Class Diagram?

Answer:
A UML Class Diagram is a structural diagram that shows the classes, their attributes, methods, and relationships between objects in the system. It models the static structure of a system.

Q8: Define UML Dynamic Modelling.

Answer:
UML Dynamic Modelling refers to the representation of the dynamic behavior of a system, such as interactions between objects and changes in states, using diagrams like sequence diagrams, state machine diagrams, and activity diagrams.

Q9: Explain briefly Implementation Diagrams.

Answer:
Implementation diagrams in UML represent the implementation architecture of the system. These include **component diagrams** and **deployment diagrams**, which show how software components are deployed on hardware nodes.

Q10: What is a Use-Case Diagram?

Answer:
A Use-Case Diagram is a UML diagram that represents the functional requirements of a system

by showing interactions between actors (users) and the system, and identifying the use cases the system must perform.

Q11: What are Static and Dynamic Models?

Answer:

- **Static Model:** Represents the structural aspects of a system (e.g., Class Diagrams, Object Diagrams).
- **Dynamic Model:** Represents the behavioral aspects of a system (e.g., Sequence Diagrams, Activity Diagrams).

Q12: Explain in brief UML Extensibility.

Answer:
UML Extensibility allows extending UML through **stereotypes**, **tagged values**, and **constraints**. These mechanisms enable developers to add domain-specific elements and adapt UML for different needs.

Q13: Define UML Meta-Model.

Answer:
A UML Meta-Model defines the abstract syntax and semantics of UML itself, describing the rules and structures that govern how UML models are built and understood.

Q14: How are associations used in UML?

Answer:
In UML, associations represent relationships between two or more classes or objects. They show how objects interact with each other and can be further classified as **unidirectional** or **bidirectional**.

Q15: What are the primary goals in the design of UML?

Answer:
The primary goals of UML design are to:

- Provide a standardized way to visualize system design.
- Foster communication between stakeholders.
- Ensure scalability and reusability in system architecture.

Q16: List various UML diagrams.

Answer:
The various UML diagrams are:

1. Class Diagram
2. Object Diagram
3. Use-Case Diagram
4. Sequence Diagram
5. Activity Diagram
6. State Machine Diagram
7. Component Diagram
8. Deployment Diagram

Q17: What is a UML Activity Diagram?

Answer:
A UML Activity Diagram is a type of flowchart that depicts the sequence of activities or actions that occur in a system, showing control flow from one activity to another.

Q18: Discuss briefly the advantages of using UML.

Answer:
Advantages of using UML include:

- **Standardization:** Provides a common language for system design.
- **Visualization:** Makes complex system designs easier to understand.
- **Communication:** Enhances communication among stakeholders.

- **Reusability:** Helps in building reusable components.
- **Consistency:** Ensures consistency in design and development.

Q19: In what sense is UML unified? How does modeling help?

Answer:
UML is unified because it integrates various methods and notations from different modeling techniques into one standardized approach. Modeling helps by providing a blueprint for system development, improving understanding, and reducing complexity.

Q20: What are the different types of relationships supported in UML?

Answer:
UML supports the following types of relationships:

1. **Association:** Represents relationships between objects.
2. **Aggregation:** Represents a "whole-part" relationship.
3. **Composition:** A stronger form of aggregation where parts cannot exist independently.
4. **Inheritance:** Indicates an "is-a" relationship between classes.
5. **Dependency:** Represents a "use" relationship between components or classes.

Level B

Q21: Write the uses of UML Component Diagram.

Answer: UML Component Diagrams represent the physical architecture of a system and its software components. They are used to:

1. **Visualize Software Architecture:** Display how components like executables, libraries, or databases interact.
2. **Support Deployment Planning:** Help in planning the distribution of software components across nodes or servers.

3. **Model Component Relationships:** Show dependencies and interfaces between various system components, enabling better communication between development teams.
4. **Ensure Reusability:** Highlight reusable components, making it easier to maintain or extend the system.
5. **Facilitate System Development:** Guide developers in structuring and organizing components during implementation and deployment phases.

Q22: Explain the term design class diagram (DCD) with the help of an example.

Answer: A Design Class Diagram (DCD) focuses on the detailed design of classes in an object-oriented system. It includes attributes, methods, and relationships like inheritance or associations. DCD is more implementation-focused than conceptual class diagrams and serves as a blueprint for coding.

Example:
Consider an e-commerce system with two classes:

- Product class has attributes like productId: int, name: string, price: double.
- Order class has attributes orderId: int, quantity: int, and method calculateTotalPrice(): double.

Here, DCD would specify the data types of attributes and the parameters for methods, guiding the actual coding process.

Q23: Demonstrate how to show methods in class diagrams.

Answer: In UML class diagrams, methods are shown in the second compartment of the class rectangle, below the attributes. The format includes the visibility symbol (e.g., + for public, - for private), method name, parameters, and return type.

Example:

```
+ calculateTotal(price: double, quantity: int): double
```

This shows a public method calculateTotal, which accepts price and quantity as parameters and returns a double. Methods in a class diagram give a clear representation of the behaviors that a class can perform.

Q24: Write the attribute presentation suggested by UML.

Answer: In UML, attributes are displayed in the middle section of a class diagram. Each attribute is represented by its visibility, name, and type. The typical format is:

`visibility attributeName : dataType [multiplicity] = defaultValue`

For example:

`- name: string = "default"`

Attributes can have visibility like + (public), - (private), or # (protected), and may include multiplicity (e.g., [0..1]) or default values. This standard notation helps to clearly define the structure and constraints of the class.

Q25: Explain the use of OCL in UML.

Answer: The Object Constraint Language (OCL) in UML is used to specify detailed rules, conditions, and constraints that cannot be expressed graphically in diagrams. OCL adds precision to the model by:

1. **Defining Invariants:** Ensuring that certain conditions always hold true for a class (e.g., "the price of a product must always be positive").
2. **Expressing Pre- and Post-conditions:** Used to describe what must be true before (pre-condition) and after (post-condition) a method is executed.
3. **Navigating Associations:** OCL can navigate relationships between classes and define constraints for those relationships. By providing a formal specification of conditions, OCL helps validate and verify system behavior.

Q26: Describe the basic elements of a UML Deployment Diagram.

Answer: A UML Deployment Diagram models the physical deployment of an application on hardware nodes. Key elements include:

1. **Nodes:** Physical or virtual devices (e.g., servers, computers) where software is deployed.
2. **Artifacts:** Software components or executables that are deployed on the nodes.
3. **Communication Paths:** Represent the interactions or communication between nodes, often depicted as lines connecting the nodes.

4. **Devices:** Physical hardware units that can be connected through communication paths. These diagrams are essential for planning and visualizing the system's infrastructure and deployment architecture.

Q27: Differentiate between links and associations.

Answer:

1. **Associations**: In UML, associations represent a relationship between two or more classes at the type level. They describe how objects of one class are related to objects of another class, typically shown as a line connecting the two classes.
    - Example: An association between Student and Course classes indicating that a student enrolls in a course.
2. **Links**: Links are instances of associations, representing relationships between individual objects at runtime. While associations connect classes, links connect objects.
    - Example: A link between John (Student) and Math 101 (Course) shows that a specific student is enrolled in a specific course.

Q28: Describe briefly about association classes and association roles.

Answer:

1. **Association Classes**: An association class in UML represents an association between two or more classes but also carries additional properties or methods. It combines the features of both a class and an association, allowing the association itself to have attributes.
    - Example: In a Student and Course association, an Enrollment association class may store enrollmentDate and grade attributes.
2. **Association Roles**: These define the specific roles played by objects in an association. For example, in the association between Student and Course, a student's role could be "enrolls in," and a course's role could be "offered by."
    - Example: The roles might be labeled on either side of the association line.

Q29: Differentiate between Aggregation and Composition.

Answer:

1. **Aggregation**: A "has-a" relationship where one class contains a reference to another, but the contained object can exist independently. It is a weak association.
    - Example: A Team class aggregates Player objects. The Player can exist without the Team.
2. **Composition**: A stronger "part-of" relationship where one class cannot exist independently of the whole. If the whole is destroyed, the parts are too.
    - Example: A House class is composed of Room objects. If the House is destroyed, its Rooms are also destroyed.

Q30: Describe briefly the basic elements of a Deployment Diagram.

Answer: Deployment Diagrams in UML represent the physical deployment of software components on hardware devices. Basic elements include:

1. **Nodes**: Represent physical hardware or virtual servers where components are deployed.
2. **Artifacts**: Executable files or software modules that are deployed on nodes.
3. **Communication Paths**: Lines that connect nodes, indicating communication between them, like a network connection.
4. **Devices**: Hardware components like servers or routers.
5. **Execution Environments**: Virtual machines or runtime environments where components run.

These elements are essential for visualizing the distribution of system components and their interaction in a real-world environment.

Level C

Q31: Draw a Deployment Diagram for Book Bank System

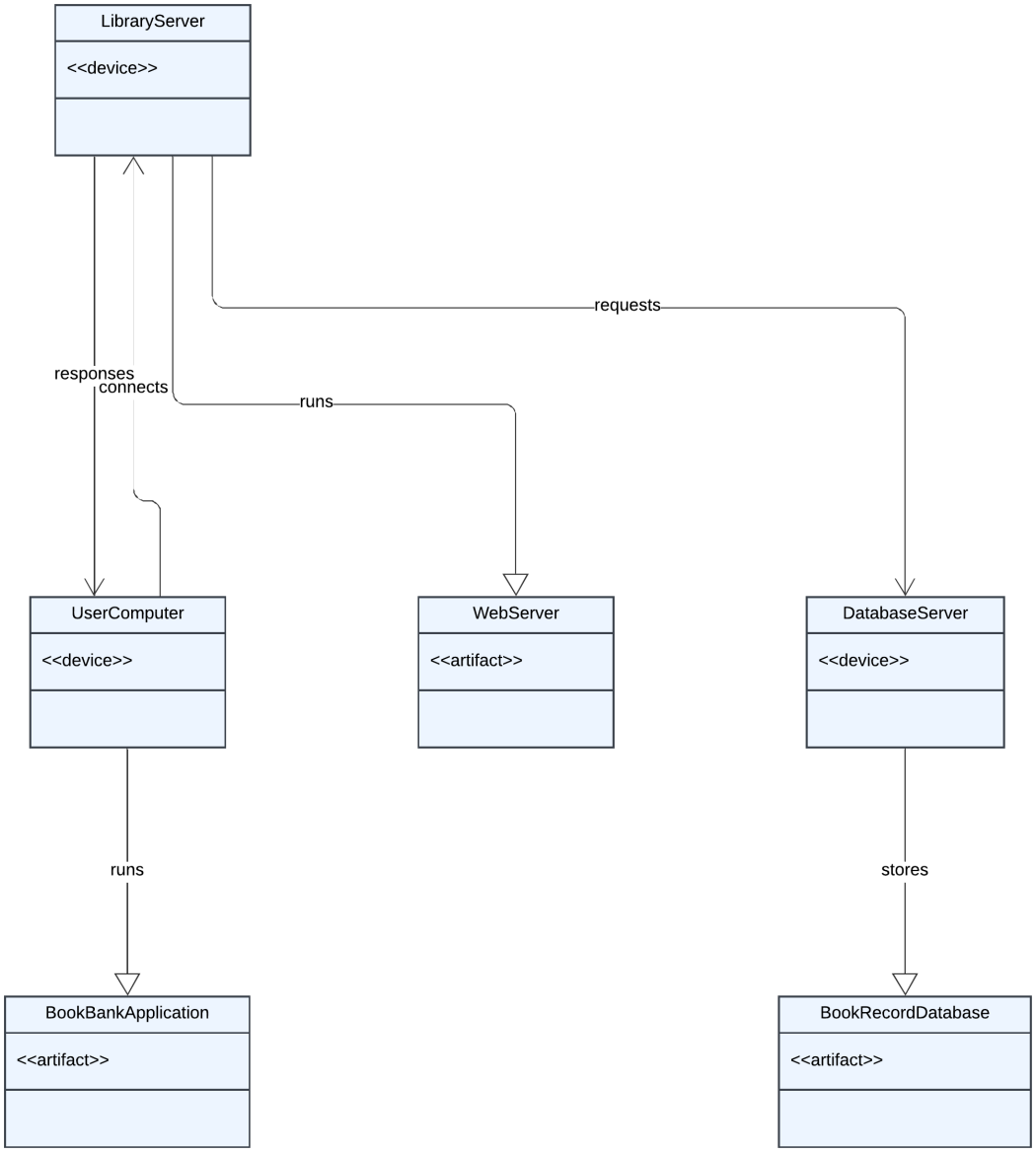
Introduction
A deployment diagram in UML shows the physical deployment of artifacts (software) on hardware nodes. For the Book Bank System, it includes the hardware devices and software components used for managing book borrowing, returns, and inventory.

Nodes and Components

In the Book Bank System, the deployment diagram would include the following elements:

- **Nodes**: These represent physical or virtual devices such as the user's computer, the library server, and the database server.
- **Artifacts**: These represent the software components like the Book Bank System application, a database containing book records, and a web server handling requests.

Diagram



Q32: Describe the UML Notation for Class Diagram with Example

Introduction

A class diagram in UML is a static structure diagram that describes the system's classes, attributes, methods, and relationships between classes.

UML Class Diagram Elements

1. **Classes**: Represented by rectangles divided into three compartments: class name, attributes, and methods.
2. **Attributes**: Displayed as visibility name: type. For example, +name: string means a public string attribute called name.
3. **Methods**: Displayed in the format visibility name(parameters): returnType. For example, +calculateTotal(price: double): double means a public method named calculateTotal returning a double.
4. **Relationships**:
    - **Association**: A solid line connecting two classes, showing their interaction.
    - **Inheritance**: Represented by a solid line with a hollow arrowhead pointing to the parent class.
    - **Aggregation**: Represented by a line with an open diamond at the end.
    - **Composition**: Represented by a line with a filled diamond at the end.

Example

In a Student class diagram for a university system:

- Student class has attributes like studentId: int, name: string.
- Methods include +enroll(course: Course): void.
- The diagram would show associations with classes like Course and Department.

Q33: Write Briefly about Elaboration. Describe the Difference Between Elaboration and Inception

Introduction to Elaboration

Elaboration is the second phase of the Unified Process (UP) in software development. It focuses on refining the project's vision, validating the core architecture, and mitigating high-risk elements. It is about turning the vague, high-level requirements into detailed system specifications.

Key Features of Elaboration

- **Detailed Requirements**: Develop use cases in more detail and define system features.
- **Architectural Baseline**: Establish an executable architecture to guide future development.
- **Risk Mitigation**: Identify and resolve critical technical and design risks before moving forward.

Difference Between Elaboration and Inception

- **Inception**: Inception is the initial phase where the project vision is set, stakeholders are identified, and feasibility is assessed. It's about gathering broad, high-level requirements.
- **Elaboration**: In contrast, elaboration focuses on refining requirements, developing a solid architecture, and addressing risks. It's a deep dive into the specifics of the system and prepares the project for full-scale development.

Q34: Design the Class Diagram for Airline Reservation System

Introduction

The Airline Reservation System manages booking, flight schedules, passenger details, and ticket generation. The class diagram visualizes the system's core classes and their relationships.

Key Classes

1. **Flight**: Stores information about flights, including flightNumber: string, destination: string, departureTime: DateTime.
    - Methods: +checkAvailability(): bool, +bookSeat(): void.
2. **Passenger**: Holds passenger details such as passengerId: int, name: string, passportNumber: string.
    - Methods: +bookFlight(Flight): Ticket, +cancelBooking(Booking): void.
3. **Booking**: Represents a booking made by a passenger, including bookingId: int, status: string.
    - Methods: +confirmBooking(): void, +cancel(): void.
4. **Ticket**: Represents the ticket issued after booking, with details like ticketId: int, seatNumber: string.

Relationships

- **Association**: Passenger has a many-to-one relationship with Booking (a passenger can make multiple bookings).
- **Aggregation**: A Flight can aggregate multiple Ticket objects.
- **Inheritance**: DomesticFlight and InternationalFlight can inherit from the Flight class.

Q35: Illustrate About Aggregation and Composition with Example

Aggregation
Aggregation represents a weak relationship between two classes, where one class (the whole) contains a reference to another (the part), but the part can exist independently of the whole.

Example:
A Team class can aggregate Player objects. The team can exist even if no players are assigned to it, and players can exist independently of a team.

Team -o-> Player

Composition
Composition is a stronger relationship where one class (the whole) cannot exist without the other (the part). If the whole is destroyed, its parts are also destroyed.

Example:
A House class is composed of Room objects. If the house is demolished, the rooms cease to exist.

House <>-> Room

Key Difference
In aggregation, the part can exist independently, whereas in composition, the part is tightly bound to the whole.

Q36: Discuss About Use Case Diagram with Example

Introduction
A use case diagram is a behavioral diagram that shows the interaction between the user (actor) and the system through use cases, representing the system's functional requirements.

Elements of Use Case Diagram

1. **Actors**: Represent users or external systems that interact with the system. Actors can be human or other systems.
2. **Use Cases**: Represent specific functionalities that the system provides to its users.
3. **Relationships**: Use cases are connected to actors by solid lines, showing the interaction between them. Use cases can also have relationships like include (mandatory use) or extend (optional use).

Example:
In an ATM system:

- Actors: Customer, Bank Server.
- Use cases: Withdraw Money, Check Balance, Transfer Funds. The Withdraw Money use case might include the Authenticate User use case, indicating that authentication is a prerequisite for withdrawal.

Q37: Illustrate with an Example the Relationship Between Sequence Diagram and Use Case Diagram

Introduction

A use case diagram outlines the high-level functionality of a system, while a sequence diagram details how that functionality is executed through object interactions over time. Both diagrams work together to provide a complete picture of system behavior.

Use Case and Sequence Diagram Relationship

1. **Use Case**: A use case diagram shows system functionality. For example, in a library system, a use case might be Borrow Book.
2. **Sequence Diagram**: For the Borrow Book use case, a sequence diagram would show the detailed interaction between objects like User, Librarian, and Database in carrying out the task.
    - The sequence diagram depicts how the user sends a Borrow Request to the librarian, the librarian checks availability with the Database, and the Database confirms the book loan.

Example

For the Borrow Book use case:

- The sequence diagram shows how the User object interacts with Librarian, who then interacts with Database to complete the request. This demonstrates how the sequence diagram provides deeper insight into the behavior described by a use case.