

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**YEAR/SEM:II/IV MAX MARKS:75 MARKS**

**SUBJECT CODE:CSE61 SUBJECT NAME:OOAD**

**DATE:14.03.2024 DURATION:3.00Hrs**

**SECTION-A(20 MARKS)**

**PART-1(10\*2=20 MARKS)**

1. **System Development**: System development refers to the process of creating or enhancing software systems, applications, or solutions to meet specific requirements or address business needs. It involves stages such as analysis, design, implementation, testing, deployment, and maintenance.
2. **Seamless Process in Object-Oriented Approach**: In object-oriented approach, a seamless process refers to the smooth transition and integration of different stages of software development, including analysis, design, and implementation. It emphasizes the cohesive representation of real-world entities as objects with well-defined behaviors and interactions.
3. **Functional Model**: A functional model is a representation of a system or application that focuses on its functional aspects, describing the behavior and functionality of the system without detailing its internal structure or implementation. It typically includes use cases, scenarios, and functional requirements.
4. **Activities in System Development**: The activities in system development typically include:
   * Requirements gathering and analysis
   * System design
   * Implementation or coding
   * Testing
   * Deployment
   * Maintenance and support
5. **RUP (Rational Unified Process)**: RUP is a software development methodology that provides a disciplined approach to assigning tasks and responsibilities within a development organization. It emphasizes iterative development, risk management, and continuous improvement.
6. **Graphical Diagrams Defined by UML (Unified Modeling Language)**:
   * Class Diagram (static)
   * Use Case Diagram
   * Sequence Diagram
   * Collaboration Diagram
   * State Diagram
   * Activity Diagram
   * Component Diagram
   * Deployment Diagram
7. **Deployment Diagrams**: Deployment diagrams in UML represent the physical deployment of components and artifacts to nodes in a distributed system. They illustrate how software components are distributed across hardware nodes in a networked environment.
8. **Use Case**: A use case is a representation of a discrete unit of functionality or interaction within a system, typically described from the perspective of an actor interacting with the system. It defines the behavior of the system under specific conditions or scenarios.
9. **Component Diagrams**: Component diagrams in UML depict the organization and dependencies of components within a system. They illustrate the high-level structure of software components and their interactions, focusing on the modular design of the system.
10. **Actor and Scenario**:

* **Actor**: In the context of use case modeling, an actor represents a role played by a user, system, or external entity that interacts with the system to achieve a specific goal.
* **Scenario**: A scenario describes a sequence of steps or interactions between actors and the system to accomplish a particular task or achieve a desired outcome.

**SECTION-B (55 MARKS)**

**PART-II(5\*11=55 MARKS)**

**11.Rambaugh object modeling techniques:**

**1. Object Model**

Object Model encompasses the principles of abstraction, encapsulation, modularity, hierarchy, typing, concurrency and persistence. Object Model basically emphasizes on the *object* and *class*.

**2. Dynamic Model**

Dynamic Model involves states, events and state diagram (transition diagram) on the model. Main concepts related with Dynamic Model are states, transition between states and events to trigger the transitions.

**3. Functional Model**

Functional Model focuses on how data is flowing, where data is stored and different processes. Main concepts involved in Functional Model are data, data flow, data store, process and actors.

## Phases of Object Modeling Technique

OMT has the following phases:

### 1. Analysis

This is the first phase of the object modeling technique. This phase involves the preparation of precise and correct modelling of the real world problems. Analysis phase starts with setting a goal i.e. finding the problem statement. Problem statement is further divided into above discussed three models i.e. object, dynamic and functional model.

### 2. System Design

This is the second phase of the object modeling technique and it comes after the analysis phase. It determines all system architecture, concurrent tasks and data storage. High level architecture of the system is designed during this phase.

### 3. Object Design

Object design is the third phase of the object modelling technique and after system design is over, this phase comes. Object design phase is concerned with classification of objects into different classes and about attributes and necessary operations needed. Different issues related with generalization and aggregation are checked.

### 4. Implementation

This is the last phase of the object modeling technique. It is all about converting prepared design into the software. Design phase is translated into the Implementation phase.

**OR**

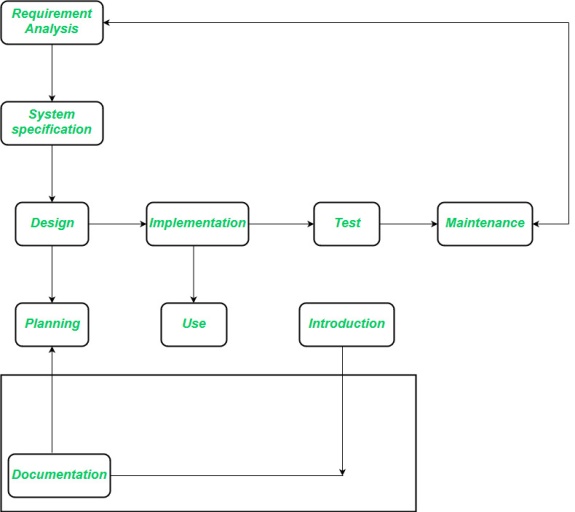
**12.System Life Cycle used in object oriented approach:**

1. **Object-oriented analysis:** Object-oriented analysis develops an object-oriented model of the application domain.
2. Object-oriented design: Object-oriented design develops an object-oriented model of the software system.
3. **Object-oriented programming:**Object-oriented programming realizes the software design with an object-oriented programming language that supports the direct implementation of objects, classes, and inheritance.

There are a variety of object-oriented methodologies such as:

1. **Object Identification:** System objects and their characteristics and events.
2. **Object Organization:** Shows how objects are related via “part-of” relationships.
3. **Object Interfaces:** Shows how objects interact with other objects.

These activities tend to overlap and in general parallel.



*Object-oriented Life Cycle Model*

The **requirements analysis** stage strives to achieve an understanding of the client’s application domain. The task that a software solution must address emerge in the course of requirement analysis. The requirement analysis phase remains completely independent of any implementation technique that might be applied later.

**13.** **BOOCH Methodology:**

The Booch methodology, developed by Grady Booch, is an object-oriented analysis and design (OOAD) methodology that provides a comprehensive approach to software development. Here's an overview of the key aspects of the Booch methodology:

1. **Object-Oriented Concepts**: The Booch methodology emphasizes modeling real-world entities as objects with attributes and behaviors. It focuses on encapsulating data and functionality within classes, promoting concepts such as inheritance, polymorphism, and abstraction.
2. **Use of Diagrams**: Booch methodology utilizes various diagrams and modeling techniques to visualize and represent different aspects of software systems. These diagrams may include class diagrams, object diagrams, state diagrams, and collaboration diagrams, among others.
3. **Iterative Development**: The Booch methodology advocates for iterative and incremental development processes, allowing for continuous refinement and improvement of system designs. It encourages feedback-driven development, where designs are iteratively refined based on stakeholder feedback and evolving requirements.
4. **Reusability and Modularity**: The Booch methodology promotes the reuse of existing components and the creation of modular, maintainable designs. It encourages the identification of common patterns, components, and frameworks that can be reused across multiple projects, fostering scalability and flexibility.
5. **Collaboration and Communication**: The Booch methodology emphasizes collaboration and communication among stakeholders, developers, and designers throughout the development lifecycle. It encourages the use of modeling techniques and visual representations to facilitate understanding and alignment among team members.
6. **Model-Driven Development**: Booch methodology emphasizes the use of models as central artifacts in the development process. Models capture the essential aspects of the system, allowing stakeholders to visualize, analyze, and communicate complex system designs effectively.
7. **Analysis and Design**: The Booch methodology provides guidelines and techniques for conducting object-oriented analysis and design activities. It emphasizes the identification of requirements, the creation of conceptual models, and the transformation of these models into concrete designs.

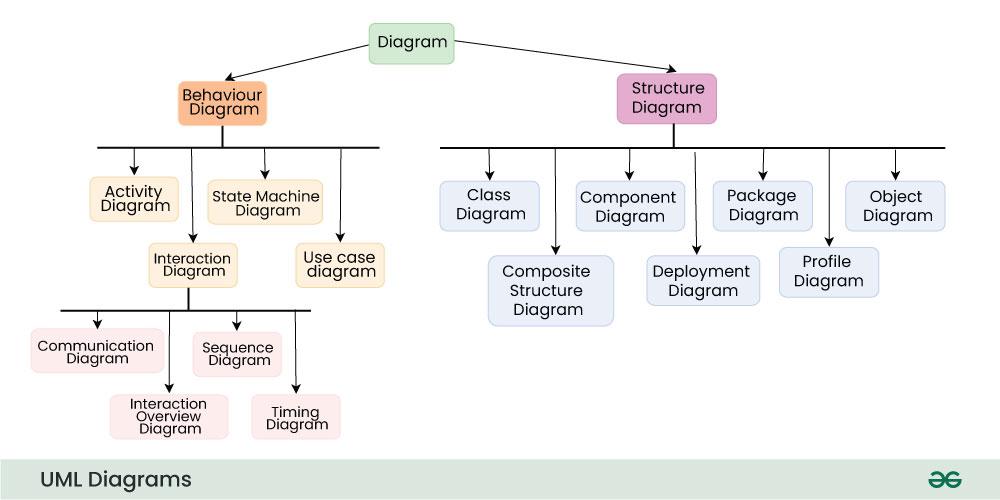
**OR**

**14.** **Traditional Life Cycle models used in OOAD:**

In Object-Oriented Analysis and Design (OOAD), traditional life cycle models provide a structured approach to software development, emphasizing different phases and activities throughout the development process. Some of the traditional life cycle models commonly used in OOAD include:

1. **Waterfall Model**: The waterfall model follows a linear sequential flow, where each phase (requirements, design, implementation, testing, deployment, and maintenance) is completed before moving on to the next. It lacks flexibility for changes later in the development process.
2. **V-Model**: The V-Model is an extension of the waterfall model, where each phase of the development process is paired with a corresponding testing phase. It emphasizes the importance of testing throughout the development life cycle.
3. **Spiral Model**: The spiral model combines iterative development with elements of the waterfall model. It involves cyclic iterations through the stages of risk analysis, planning, implementation, and evaluation. Each cycle represents a prototype iteration, allowing for progressive refinement and risk mitigation.
4. **Incremental Model**: The incremental model divides the development process into smaller, manageable increments or modules. Each increment delivers a portion of the system's functionality, allowing for incremental development and delivery of the software.
5. **Prototype Model**: The prototype model involves the creation of an initial, partial implementation of the system to gather feedback and refine requirements. It focuses on rapid prototyping and iteration to explore and validate design concepts.
6. **Unified Process (UP)**: The Unified Process is an iterative and incremental development methodology that combines elements of the waterfall model with iterative development. It divides the development process into phases (inception, elaboration, construction, and transition) and iterations within each phase.
7. **Rational Unified Process (RUP)**: RUP is a specific instantiation of the Unified Process, developed by Rational Software (now IBM). It provides a structured approach to software development, emphasizing iterative development, use case-driven analysis and design, and architecture-centric development.
8. **Evolutionary Model**: The evolutionary model involves the incremental development of a system over time, allowing for continuous refinement and enhancement based on evolving requirements and feedback from users.

**15.**  **UML Models:**



Unified Modeling Language (UML) provides various diagrams and models to represent different aspects of software systems. Here are the main UML models:

1. **Class Diagram**:
   * Represents the static structure of classes, their attributes, operations, and relationships.
   * Used to model the conceptual classes and their associations in the system.
2. **Use Case Diagram**:
   * Illustrates the interactions between actors (users) and the system to achieve specific goals or use cases.
   * Used to identify and define the functional requirements of the system from the user's perspective.
3. **Sequence Diagram**:
   * Represents the sequence of interactions between objects over time to accomplish a specific task or scenario.
   * Used to visualize the dynamic behavior of objects and their interactions in response to events.
4. **State Diagram (or State Machine Diagram)**:
   * Models the behavior of objects over time in response to events and state transitions.
   * Used to represent the lifecycle of an object and its state transitions based on external stimuli.
5. **Activity Diagram**:
   * Depicts the flow of control and actions within a system or process, representing workflows, algorithms, or business processes.
   * Used to model the sequential and concurrent activities in a system.
6. **Component Diagram**:
   * Illustrates the organization and dependencies of components within a system.
   * Used to represent the physical structure of the system and the relationships between components.
7. **Deployment Diagram**:
   * Represents the physical deployment of components and artifacts to nodes in a distributed system.
   * Used to illustrate how software components are distributed across hardware nodes in a networked environment.
8. **Package Diagram**:
   * Shows the organization and dependencies of packages or modules within a system.
   * Used to depict the structure of the system at a higher level of abstraction, focusing on the grouping and organization of related elements.
9. **Object Diagram**:
   * Represents a snapshot of the objects and their relationships at a specific point in time.
   * Used to illustrate instances of classes and their relationships in a particular scenario or context.
10. **Communication Diagram (formerly Collaboration Diagram)**:
    * Shows how objects collaborate to fulfill a particular task or scenario.
    * Used to emphasize communication and interaction between objects, including the messages exchanged between them.

**OR**

**16.** **UML Class Diagram and its notation:**

Class diagrams are a type of UML (Unified Modeling Language) diagram used in software engineering to visually represent the structure and relationships of classes within a system

## UML Class Notation

1. **Class Name:**
   * The name of the class is typically written in the top compartment of the class box and is centered and bold.
2. **Attributes:**
   * Attributes, also known as properties or fields, represent the data members of the class. They are listed in the second compartment of the class box and often include the visibility (e.g., public, private) and the data type of each attribute.
3. **Methods:**
   * Methods, also known as functions or operations, represent the behavior or functionality of the class. They are listed in the third compartment of the class box and include the visibility (e.g., public, private), return type, and parameters of each method.
4. **Visibility Notation:**
   * Visibility notations indicate the access level of attributes and methods. Common visibility notations include:
     + + for public (visible to all classes)
     + - for private (visible only within the class)
     + # for protected (visible to subclasses)
     + ~ for package or default visibility (visible to classes in the same package

**17.** **a. Sequence diagram:**

In Object-Oriented Analysis and Design (OOAD), a Sequence Diagram is a behavioral diagram that depicts the interactions between objects or components within a system over time to accomplish a specific task or scenario. Sequence diagrams are particularly useful for visualizing the flow of messages and the sequence of method invocations during runtime. Here's how Sequence Diagrams are used in OOAD:

1. **Identifying Objects and Participants**:
   * Begin by identifying the objects or participants involved in the scenario. These objects represent instances of classes or components within the system.
2. **Creating Lifelines**:
   * Represent each object as a vertical lifeline or box along the top of the diagram. The lifeline represents the lifespan or existence of the object during the scenario.
3. **Adding Messages**:
   * Depict the messages exchanged between objects to accomplish the desired behavior. Messages can be synchronous, asynchronous, or return messages.
   * Use arrows to indicate the flow of messages from the sender to the receiver.
4. **Specifying Activation and Deactivation**:
   * Show the period of time during which an object is active or executing a particular operation. This is represented by the activation bar or thickened portion of the lifeline.
   * Use activation and deactivation symbols to indicate when an object starts and ends its activity.
5. **Including Control Constructs**:
   * Integrate control constructs such as loops, conditions, and branches into the sequence diagram as necessary. Use combined fragments (e.g., alt, opt, loop) to represent these control structures.
6. **Adding Execution Occurrences**:
   * Specify the points in time when methods or operations are executed by objects. Execution occurrences are represented by small rectangles on the lifeline.
7. **Documenting Scenarios**:
   * Sequence diagrams are often used to document specific scenarios or use cases within the system. They provide a detailed view of the interactions between objects, helping to clarify the behavior of the system.
8. **Validating Design**:
   * Sequence diagrams can also be used to validate the design of the system by identifying potential issues, such as missing or redundant interactions, and refining the design accordingly.

**b. Collaboration diagram:**

In Object-Oriented Analysis and Design (OOAD), a Collaboration Diagram (also known as a Communication Diagram) is a behavioral diagram that illustrates the interactions between objects or components within a system to achieve a specific task or scenario. Collaboration diagrams focus on showing the structural organization of objects and the messages exchanged between them during runtime. Here's how Collaboration Diagrams are used in OOAD:

1. **Identifying Objects and Participants**:
   * Begin by identifying the objects or participants involved in the scenario. These objects represent instances of classes or components within the system.
2. **Creating Objects**:
   * Represent each object as a labeled rectangle or oval on the diagram. The label typically includes the name of the object and optionally its class or type.
3. **Adding Links and Messages**:
   * Depict the interactions between objects using links or connectors. Links represent relationships or associations between objects.
   * Use arrows to indicate the flow of messages from the sender to the receiver. Messages can be synchronous, asynchronous, or return messages.
4. **Specifying Message Flow**:
   * Show the sequence of messages exchanged between objects to accomplish the desired behavior. Messages flow along the links between objects, indicating the direction of communication.
5. **Documenting Scenarios**:
   * Collaboration diagrams are often used to document specific scenarios or use cases within the system. They provide a visual representation of the interactions between objects, helping to clarify the behavior of the system.
6. **Validating Design**:
   * Collaboration diagrams can also be used to validate the design of the system by identifying potential issues, such as missing or redundant interactions, and refining the design accordingly.
7. **Including Control Constructs**:
   * Integrate control constructs such as loops, conditions, and branches into the collaboration diagram as necessary. Use combined fragments (e.g., alt, opt, loop) to represent these control structures.
8. **Adding Execution Occurrences**:
   * Specify the points in time when methods or operations are executed by objects. Execution occurrences are represented by small rectangles or symbols on the diagram, similar to sequence diagrams.

**OR**

**18.** **Use case Diagram and Identifying Use Cases from the Actors:**

A Use Case Diagram is a type of Unified Modeling Language (UML) diagram that represents the interaction between actors (users or external systems) and a system under consideration to accomplish specific goals.

### Actors

Actors are external entities that interact with the system. These can include users, other systems, or hardware devices. In the context of a Use Case Diagram, actors initiate use cases and receive the outcomes.

### Use Cases

Use cases are like scenes in the play. They represent specific things your system can do. In the online shopping system, examples of use cases could be “Place Order,” “Track Delivery,” or “Update Product Information”.

### System Boundary

The system boundary is a visual representation of the scope or limits of the system you are modeling. It defines what is inside the system and what is outside

## Use Case Diagram Relationships

### Association Relationship

TheAssociation Relationship represents a communication or interaction between an actor and a use case. It is depicted by a line connecting the actor to the use case. This relationship signifies that the actor is involved in the functionality described by the use case.

**Example: Online Banking System**

* **Actor:** Customer
* **Use Case:** Transfer Funds
* **Association:** A line connecting the “Customer” actor to the “Transfer Funds” use case, indicating the customer’s involvement in the funds transfer process.

### Include Relationship

The Include Relationship indicates that a use case includes the functionality of another use case. It is denoted by a dashed arrow pointing from the including use case to the included use case. This relationship promotes modular and reusable design.

**Example: Social Media Posting**

* **Use Cases:** Compose Post, Add Image
* **Include Relationship:** The “Compose Post” use case includes the functionality of “Add Image.” Therefore, composing a post includes the action of adding an image.

### Extend Relationship

The Extend Relationship illustrates that a use case can be extended by another use case under specific conditions. It is represented by a dashed arrow with the keyword “extend.” This relationship is useful for handling optional or exceptional behavior.

**Example: Flight Booking System**

* **Use Cases:** Book Flight, Select Seat
* **Extend Relationship:** The “Select Seat” use case may extend the “Book Flight” use case when the user wants to choose a specific seat, but it is an optional step.

### Generalization Relationship

The Generalization Relationship establishes an “is-a” connection between two use cases, indicating that one use case is a specialized version of another. It is represented by an arrow pointing from the specialized use case to the general use case.

**Example: Vehicle Rental System**

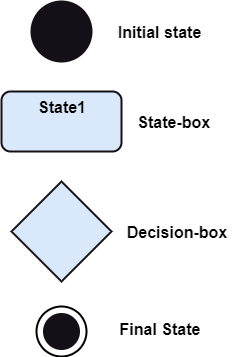
* **Use Cases:** Rent Car, Rent Bike
* **Generalization Relationship:** Both “Rent Car” and “Rent Bike” are specialized versions of the general use case “Rent Vehicle.”

**OR**

**19.** **State Diagram and its functions:**

## Notation of a State Machine Diagram

Following are the notations of a state machine diagram enlisted below:



1. **Initial state:** It defines the initial state (beginning) of a system, and it is represented by a black filled circle.
2. **Final state:** It represents the final state (end) of a system. It is denoted by a filled circle present within a circle.
3. **Decision box:** It is of diamond shape that represents the decisions to be made on the basis of an evaluated guard.
4. **Transition:** A change of control from one state to another due to the occurrence of some event is termed as a transition. It is represented by an arrow labeled with an event due to which the change has ensued.
5. **State box:** It depicts the conditions or circumstances of a particular object of a class at a specific point of time. A rectangle with round corners is used to represent the state box.

## Types of State

The UML consist of three states:

1. **Simple state:** It does not constitute any substructure.
2. **Composite state:** It consists of nested states (substates), such that it does not contain more than one initial state and one final state. It can be nested to any level.
3. **Submachine state:** The submachine state is semantically identical to the composite state, but it can be reused.

## use a State Machine Diagram

State machine diagram is used for:

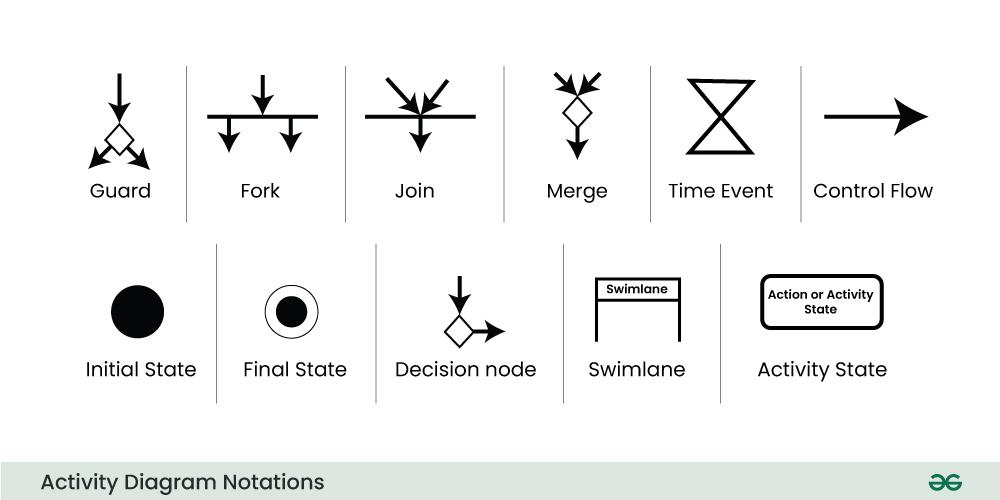
1. For modeling the object states of a system.
2. For modeling the reactive system as it consists of reactive objects.
3. For pinpointing the events responsible for state transitions.
4. For implementing forward and reverse engineering.

**OR**

**20.** **Activity Diagram:**

Activity Diagrams are used to illustrate the flow of control in a system and refer to the steps involved in the execution of a use case

## Activity Diagram Notations



### ****Initial State****

The starting state before an activity takes place is depicted using the initial state.

### ****Action or Activity State****

An activity represents execution of an action on objects or by objects. We represent an activity using a rectangle with rounded corners.

### ****Action Flow or Control flows****

Action flows or Control flows are also referred to as paths and edges. They are used to show the transition from one activity state to another activity state.

### ****Decision node and Branching****

When we need to make a decision before deciding the flow of control, we use the decision node.

### ****Guard****

A Guard refers to a statement written next to a decision node on an arrow sometimes within square brackets.

### ****Fork****

Fork nodes are used to support concurrent activities. When we use a fork node when both the activities get executed concurrently

### ****Join****

Join nodes are used to support concurrent activities converging into one. For join notations we have two or more incoming edges and one outgoing edge.

### ****Merge or Merge Event****

Scenarios arise when activities which are not being executed concurrently have to be merged. We use the merge notation for such scenarios.

### ****Swimlanes****

We use Swimlanes for grouping related activities in one column. Swimlanes group related activities into one column or one row. Swimlanes can be vertical and horizontal.

### ****Time Event****

This refers to an event that stops the flow for a time; an hourglass depicts it.

### ****Final State or End State****

The state which the system reaches when a particular process or activity ends is known as a Final State or End State.