OOAD(Object Oriented Analysis and Design)

2017 1. What is observer pattern? 2. What is class diagram? 3. List the characteristics of unified process. 4. Define actor in use case diagram. 5. What are the types of interaction diagram? 6. What is dynamic model? 7. Define a conceptual class. 8. When an aggregation is said to be composite aggregation? 9. What is activity diagram? 10. List any two types of common association.	**2018** 11. What does actor represent in use case diagram? 12. What is layered architecture? 13. Define 100% rule. 14. What is abstract class? 15. When "includes" is used in use case diagram? 16. What is the use of activity diagram? 17. List three kinds of actor. 18. What is domain model? 19. What is meant by low 20. What is component diagram?
2019 (make up) 21. List the relationships that can exist between use cases. 22. What is the role of creator 23. Why is high cohesion important for software? 24. Define conceptual class. 25. What does role specify while assigning relationship between classes? 26. Which UML diagram is used to describe functional requirement? 27. Which symbols are used to represent private and protected members in class diagram? 28. Differentiate between abstract class and concrete class. 29. List the possible relationships between classes. 30. What is the task performed in elaboration phase?	**2019 TU** 31. What do you mean by low coupling? 32. What is 'includes' in use case? 33. What are the strategies to find conceptual classes? 34. When are Specification Conceptual Classes Required? 35. Why an activity diagram is needed in behavioral modeling? 36. What is collaboration diagram? 37. Define adapter design pattern. 38. List any five GRASP patterns. 39. Why high cohesive design is needed? 40. What do you mean by iterative development method?
2022 41. Why do we need interface? 42. What is the importance of Use-Case Generalization? 43. List any two symbols and their task used in activity diagram. 44. Why should we prepare deployment diagram? 45. Give an example of association and composition. 46. What is an iteration? 47. How can we achieve high cohesion? 48. How do we represent abstract class and interface in the descriptive class diagram? 49. Why does OOAD prefer low coupling? Notes 50. How can State Diagram be used for modeling dynamic behavior of the system?	**2023** 51. Why is UML regarded both as a tool and a language? 52. How do we represent abstract class and interface in a class diagram? 53. Define description class. 54. List the components of use case diagram. 55. Why should inception phase be of short duration? 56. Define UML state diagram. 57. What are the input and output artifacts of elaboration phase? 58. How can we avoid a direct coupling between two or more elements? 59. Define 100% rule. 60. What is domain model?

Here are answers to the questions you listed:

2017

1. Observer Pattern: This behavioral design pattern defines a one-to-many dependency between objects. When one object (the subject) changes state, all dependent objects (observers) are notified and updated automatically.

2. Class Diagram: A class diagram is a type of UML diagram that describes the static structure of a system by showing its classes, their attributes, methods, and the relationships between the classes.

3. Characteristics of Unified Process:
- **Iterative and incremental development**
- **Phases: Inception, Elaboration, Construction, and Transition**
- **Focus on risk management**
- **Use-case driven**
- **Architecture-centric**

4. Actor in Use Case Diagram: An actor represents a user or another system that interacts with the system being modeled. Actors initiate use cases and interact with the system's functionality.

5. Types of Interaction Diagrams:
- **Sequence Diagrams**
- **Collaboration Diagrams (also known as Communication Diagrams)**

6. Dynamic Model: A dynamic model describes the behavior of a system over time, focusing on how the system responds to events and changes state. It includes models like sequence diagrams and state diagrams.

7. Conceptual Class: A conceptual class is a class that represents a key concept in the problem domain, usually reflecting real-world entities or abstractions. It's used in the conceptual model to define what should be modeled.

8. Composite Aggregation: Aggregation is considered composite when it represents a whole-part relationship with a strong lifecycle dependency, meaning the lifetime of the part is dependent on the lifetime of the whole. If the whole is destroyed, the parts are too.

9. Activity Diagram: An activity diagram represents the workflow or activities of a system or process. It shows the sequence of actions, decisions, and parallel processes.

10. Two Types of Common Association:
- **Unidirectional Association: One class is aware of another, but not vice versa.**
- **Bidirectional Association: Both classes are aware of each other.**

2018

11. Actor in Use Case Diagram: An actor represents a role that interacts with the system. It can be a user, another system, or any external entity that interacts with the system's functionalities.

12. Layered Architecture: Layered architecture is a software design pattern where the system is divided into layers with each layer having specific responsibilities. Typically, layers include presentation, business logic, and data access.

13. 100% Rule: The 100% rule states that all the requirements of a system should be represented in the use case diagram. It ensures that all functional requirements are covered by use cases.

14. Abstract Class: An abstract class is a class that cannot be instantiated on its own and is intended to be subclassed. It may include abstract methods that must be implemented by its subclasses.

15. Includes in Use Case Diagram: "Includes" represents a relationship where one use case (the base use case) includes the behavior of another use case (the included use case). It's used to reuse common functionality.

16. Use of Activity Diagram: Activity diagrams are used to model the flow of control or data within a system. They help visualize and understand workflows, process sequences, and decision points.

17. Three Kinds of Actor:
- **Primary Actor: Initiates a use case to achieve a goal.**
- **Secondary Actor: Provides a service or input to the system.**
- **Supporting Actor: Helps with the execution of a use case but isn't directly involved.**

18. Domain Model: A domain model represents the concepts and relationships within a specific problem domain. It's a conceptual model that provides a high-level view of the system's structure.

19. Low Coupling: Low coupling refers to a design principle where classes or components have minimal dependencies on each other. This enhances modularity and reduces the impact of changes in one part of the system on others.

20. Component Diagram: A component diagram describes the physical components of a system and their dependencies. It shows how different components (like executables, libraries, or files) are organized and interact.

2019 (Make-Up)

21. Relationships Between Use Cases:
- **Includes: Represents a relationship where one use case includes the functionality of another.**
- **Extends: Represents an optional behavior that extends the base use case.**
- **Generalization: Represents a relationship where a use case inherits behavior from another use case.**

22. Role of Creator: The creator role in design patterns specifies which class is responsible for creating instances of another class. It helps in delegating object creation responsibilities to appropriate classes.

23. Importance of High Cohesion: High cohesion means that a class or module is focused on a single task or responsibility. It makes the system more understandable, maintainable, and reliable, as changes in one part have minimal impact on others.

24. Conceptual Class: (Same as previously defined) A conceptual class represents key concepts in the problem domain, often corresponding to real-world entities or abstractions.

25. Role in Class Relationships: A role specifies the function or responsibility of a class in a relationship. It defines how the class participates in the association, aggregation, or composition with other classes.

26. UML Diagram for Functional Requirement: Use Case Diagram is used to describe functional requirements by depicting the interactions between actors and the system's use cases.

27. Symbols for Private and Protected Members:
- **Private Members: Represented with a minus sign (-).**
- **Protected Members: Represented with a hash sign (#).**

28. Abstract Class vs. Concrete Class:
- **Abstract Class: Cannot be instantiated; contains abstract methods that must be implemented by subclasses.**
- **Concrete Class: Can be instantiated and provides implementations for all its methods.**

29. Possible Relationships Between Classes:
- **Association**
- **Aggregation**
- **Composition**
- **Inheritance (Generalization)**

- Dependency

30. Task in Elaboration Phase: The elaboration phase involves refining the system's architecture, addressing high-risk elements, and establishing a detailed project plan. It includes developing detailed use cases and system requirements.

2019 TU

31. Low Coupling: (Same as previously defined) Low coupling means that components or classes have minimal dependencies on each other, which improves modularity and makes the system easier to maintain and understand.

32. Includes in Use Case: (Same as previously defined) The "includes" relationship is used when one use case's behavior is included in the execution of another use case.

33. Strategies to Find Conceptual Classes:
   - Identify Nouns: Extract nouns from requirements and domain descriptions.
   - Identify Verbs: Look for verbs that imply actions or processes.
   - Analyze Use Cases: Derive classes from the main activities and interactions described in use cases.

34. Specification of Conceptual Classes Required: Specification is needed when defining the detailed attributes, operations, and constraints of conceptual classes, which is crucial for accurate system modeling and design.

35. Need for Activity Diagram in Behavioral Modeling: Activity diagrams help visualize workflows, identify potential bottlenecks, and understand the sequence of activities and decision points within a process.

36. Collaboration Diagram: A collaboration diagram (or communication diagram) shows how objects interact and collaborate to achieve a goal. It emphasizes the relationships and communication between objects.

37. Adapter Design Pattern: The adapter pattern allows incompatible interfaces to work together. It acts as a bridge by converting the interface of a class into another interface that clients expect.

38. Five GRASP Patterns:
   - Information Expert
   - Creator
   - Controller
   - Low Coupling
   - High Cohesion

39. Need for High Cohesion: High cohesion ensures that a class or module performs a single, well-defined task, making the system easier to understand, maintain, and modify. It leads to better organization and reduced interdependence.

40. Iterative Development Method: An iterative development method involves repeating cycles of development where each iteration refines and builds upon the previous one. It allows for incremental progress and continuous improvement.

2022

41. Need for Interface: Interfaces provide a way to define a contract that classes must adhere to, promoting flexibility, reusability, and decoupling. They allow for different implementations to be used interchangeably.

42. Importance of Use-Case Generalization: Use-case generalization allows for the reuse of common functionality across multiple use cases by defining a general use case and extending it with specific use cases.

43. Two Symbols in Activity Diagram:

- Action/Activity Node: Represented by a rounded rectangle, it denotes a task or operation.
- Decision Node: Represented by a diamond shape, it shows a decision point with multiple outcomes.

44. Importance of Deployment Diagram: Deployment diagrams show the physical arrangement of hardware and software components, helping in understanding system deployment, configuration, and interactions between components.

45. Example of Association and Composition:
   - Association: A teacher and a student have an association where a teacher can teach multiple students.
   - Composition: A house and its rooms, where rooms are parts of the house and cannot exist independently.

46. Iteration: An iteration refers to a single cycle of the development process where a specific set of tasks or features is developed, tested, and reviewed. It is a part of the iterative development methodology.

47. Achieving High Cohesion: High

cohesion can be achieved by ensuring that each class or module has a single, well-defined responsibility and by organizing related functions and data together.

48. Representation of Abstract Class and Interface:
   - Abstract Class: Represented with italics or with a label indicating "abstract" in a class diagram.
   - Interface: Represented with a stereotype «interface» or with a label indicating "interface."

49. OOAD Preference for Low Coupling: Low coupling is preferred in Object-Oriented Analysis and Design (OOAD) because it minimizes dependencies between classes, making the system more modular, easier to maintain, and more flexible to changes.

50. State Diagram for Dynamic Behavior: A state diagram models the different states an object can be in and how it transitions between states based on events. It helps in understanding and managing the dynamic behavior of the system.

2023

51. UML as Tool and Language: UML is regarded as both a tool and a language because it provides a standardized set of visual notations (language) for modeling systems and also offers tools for creating and analyzing these models.

52. Representation of Abstract Class and Interface:
   - Abstract Class: Represented with italicized class names or a specific notation for abstract classes.
   - Interface: Represented with the «interface» stereotype or a specific notation indicating an interface.

53. Description Class: A description class provides additional information about the roles, responsibilities, and behavior of a class. It’s often used to provide context or documentation for classes in a system.

54. Components of Use Case Diagram:
   - Actors: Represent entities that interact with the system.
   - Use Cases: Represent the functionalities or services provided by the system.
   - Relationships: Include associations, includes, extends, and generalizations.

55. Short Duration of Inception Phase: The inception phase should be short to quickly validate the project’s feasibility, gather initial requirements, and create a basic project plan, avoiding prolonged planning that could delay development.

56. UML State Diagram: A state diagram represents the states of an object and the transitions between these states based on events. It helps model the dynamic behavior and lifecycle of an object.

57. Input and Output Artifacts of Elaboration Phase:
- **Input: Initial requirements, project vision, use cases.**
- **Output: Refined architecture, risk assessment, detailed project plan.**

58. Avoiding Direct Coupling: To avoid direct coupling, use design principles such as dependency injection, interfaces, or abstract classes to manage dependencies and promote loose coupling between components.

59. 100% Rule: (Same as previously defined) The 100% rule ensures that all functional requirements are represented by use cases in the use case diagram, providing complete coverage of the system's functionality.

60. Domain Model: (Same as previously defined) A domain model represents the key concepts, entities, and relationships within a problem domain, providing a high-level view of the system's structure and behavior.