CS6502 – OBJECT ORIENTED ANALYSIS AND DESIGN
2 Marks with Answer

Unit-I
UML DIAGRAMS

1. What is an object?
An object is a combination of data and logic; the representation of some real-world entity.

2. What are the main advantages of object oriented development?
- High level of abstraction
- Seamless transition among different phases of software development
- Encouragement of good programming techniques
- Promotion of reusability

3. What is object oriented system development methodology?
Object oriented system development methodology is a way to develop software by building self-contained modules or objects that can be easily replaced, modified and reused.

4. Distinguish between method and message in object.

Method	Message
Methods are similar to functions, procedures or subroutines in more traditional programming languages	Message essentially are non specific function Calls.
Method is an implementation	Message is an instruction
In an object oriented system, a method is invoked by sending an object a message.	An object understands a message when it can match the message to a method that has the same name as the message.

5. What is Object-Oriented Analysis?
During object-oriented analysis there is an emphasis on finding and describing the objects or concepts in the problem domain. For example, in the case of the flight information system, some of the concepts include Plane, Flight, and Pilot.

6. What is Object-Oriented Design?
During object-oriented design (or simply, object design) there is an emphasis on defining software objects and how they collaborate to fulfill the requirements.

7. What is Object-Oriented Analysis and Design?

During object-oriented analysis there is an emphasis on finding and describing the objects or concepts in the problem domain. For example, in the case of the flight information system, some of the concepts include Plane, Flight, and Pilot.

During object-oriented design (or simply, object design) there is an emphasis on defining software objects and how they collaborate to fulfill the requirements. The combination of these two concepts shortly known as object oriented analysis and design.

8. What is Analysis and Design?

Analysis emphasizes an investigation of the problem and requirements, rather than a solution. Design emphasizes a conceptual solution (in software and hardware) that fulfills the requirements, rather than its implementation. For example, a description of a database schema and software objects.

9. Define Design Class Diagrams

A static view of the class definitions is usefully shown with a design class diagram. This illustrates the attributes and methods of the classes.

10. What is the UML?

The Unified Modeling Language is a visual language for specifying, constructing and documenting the artifacts of systems.

11. What are the three ways and perspectives to Apply UML?

- Ways - UML as sketch, UML as blueprint, UML as programming language
- Perspectives-Conceptual perspective, Specification (software) perspective, Implementation (Software) perspective.

12. What is Inception?

Inception is the initial short step to establish a common vision and basic scope for the Project. It will include analysis of perhaps 10% of the use cases, analysis of the critical non-Functional requirement, creation of a business case, and preparation of the development Environment so that programming can start in the elaboration phase. Inception in one Sentence: Envision the product scope, vision, and business case.

13. What Artifacts May Start in Inception?

Some sample artifacts are Vision and Business Case, Use-Case Model, Supplementary Specification, Glossary, Risk List & Risk Management Plan, Prototypes and proof-of-concepts etc.

14. Define Requirements and mention its types.

Requirements are capabilities and conditions to which the system and more broadly, the project must conform.

1. Functional
2. Reliability

3. Performance
4. Supportability

15. What are Actors?
An actor is something with behavior, such as a person (identified by role), computer system, or organization; for example, a cashier.

16. What is a scenario?
A scenario is a specific sequence of actions and interactions between actors and the system; it is also called a use case instance. It is one particular story of using a system, or one path through the use case; for example, the scenario of successfully purchasing items with cash, or the scenario of failing to purchase items because of a credit payment denial.

17. Define Use case.
A use case is a collection of related success and failure scenarios that describe an actor using a system to support a goal. Use cases are text documents, not diagrams, and use-case modeling is primarily an act of writing text, not drawing diagrams.

18. What are Three Kinds of Actors?
- Primary actor,
- Supporting actor
- offstage actor

19. What Tests Can Help Find Useful Use Cases?
1. The Boss Test
2. The EBP Test
3. The Size Test

20. What are Use Case Diagrams?
A use case diagram is an excellent picture of the system context; it makes a good context diagram that is, showing the boundary of a system, what lies outside of it, and how it gets used. It serves as a communication tool that summarizes the behavior of a system and its actors.

21. What are Activity Diagrams?
A diagram which is useful to visualize workflows and business processes. These can be a useful alternative or adjunct to writing the use case text, especially for business use cases that describe complex workflows involving many parties and concurrent actions.

22. List the relationships used in use cases?
1.Include
2.Extend
3.Generalize

SAEC

23. What are the primary goals in the design of UML?

- Provide users a ready to use expressive visual modeling language so they can develop and exchange meaningful models.
- Provide extensibility and specialization mechanism to extend the core concepts.
- Encourage the growth of oo tools market.
- Support higher level development concepts.

24. Define class diagram.

The main static structure analysis diagram for the system,it represents the class structure of a system including the relationships between class and inheritance structure.

25. What is interaction diagram? Mention its types.

Interaction diagrams are diagrams that describe how groups of objects collaborate to get the job done interaction diagrams capture the behavior of the single use case, showing the pattern of interaction among objects.

Types:

- Sequence Diagram
- Collaboration or Communication Diagram

26. What is sequence diagram?

It is an easy and intuitive way of describing the behavior of a system by viewing the interaction between the system and its environment.

27. What is collaboration diagram?

It represents a collaboration, which is a set of objects related in a particular context, and interaction, which is a set of messages exchanged among the objects with in collaboration to achieve a desired outcome.

28. Define start chart diagram?

It shows a sequence of states that an object goes through during its life in response to events. A state is represented as a round box, which may contain one or more compartments. The compartments are all optional.

29. What is meant by implementation diagram?

Implementation diagrams show the implementation phase of systems development such as the source code structure and the run time implementation structure.

There are two types of Implementation Diagram

Component diagram and Development diagram

30. Define component diagram.

A component diagram shows the organization and dependencies among a set of components. A component diagram is used to model the static implementation view of a system. This involves modeling the physical thing that reside on a mode, such as executable, libraries, files and documents.

31. Define deployment diagram,

Deployment diagram shows the configuration of run time processing elements and the software components, processes and objects that live in them.

Deployment diagrams are used to model the static deployment view of a system. A deployment diagram is a graph of nodes connected by communication association.

32. What is UP?

A software development process describes an approach to building, deploying and possibly maintaining software. The unified process has emerged as a popular iterative software development process for building object oriented system.

33. What is iteration?

A key practice in both the UP and most other modern methods is iterative development. In this lifecycle approach, development is organized into a series of short, fixed lengthmini projects called iterations.

34. What is iterative evolutionary development?

The iterative lifecycle is based on the successive enlargement and refinement of a system through multiple iterations, with cyclic feedback and adaptation as core drivers to converge upon a suitable system. The system grows incrementally over time, iteration by iteration and thus this approach is also known as iterative and incremental development.

35. What are the phases of unified process?

a. Inception
b. Elaboration
c. Construction
d. Transition

36. What is inception?

Inception is the initial short step to establish a common vision and basic scope for the project. It will include analysis of perhaps 10% of the use cases, analysis of the critical non-functional requirement, creation of a business case and preparation of the development environment.

37. Define use case modeling.

Use case modeling is a form of requirement engineering. How to create an SRS in what we might call the traditional way. Use case modeling is a different and complementary way of eliciting and documenting requirements.

38. Define use case generalization?

Use case generalization is used when you have one or more use cases that are rally specializations of more general case.

SAEC

39. What is an activity diagram?

A UML activity diagram shows sequential and parallel activities in a process. They are useful for modeling business processes, workflows, data flows, and complex algorithms. Basic UML activity diagram notation illustrates an action, partition, fork, join, and object node. In essence, this diagram shows a sequence of actions, some of which may be parallel. Most of the notation is self-explanatory; two subtle points:

- once an action is finished, there is an automatic outgoing transition
- the diagram can show both control flow and data flow

40. What are the Benefits of Iterative Development?

- Early rather than late mitigation of high risks (technical, requirements, objectives, usability, and so forth)
- Early visible progress
- Early feedback, user engagement, and adaptation, leading to a refined system that more closely meets the real needs of the stakeholders managed complexity; the team is not overwhelmed by "analysis paralysis" or very long and complex steps
- The learning within iteration can be methodically used to improve the development process itself, iteration by iteration

SAEC

UNIT-II
DESIGN PATTERNS

1. How to Choose the Initial Domain Object?

Choose as an initial domain object a class at or near the root of the containment or aggregation hierarchy of domain objects. This may be a facade controller, such as *Register,* or some other object considered to contain all or most other objects, such as a *Store.*

2. Define patterns.

A pattern is a named problem/solution pair that can be applied in new context, with advice on how to apply it in novel situations and discussion of its trade-offs.

3. How to Connect the UI Layer to the Domain Layer?

An initializing routine (for example, a Java *main* method) creates both a UI and a domain object, and passes the domain object to the UI.

A UI object retrieves the domain object from a well-known source, such as a factory object that is responsible for creating domain objects.

4. Mention the Interface and Domain Layer Responsibilities.

The UI layer should not have any domain logic responsibilities. It should only be responsible for user interface tasks, such as updating widgets. The UI layer should forward requests for all domain-oriented tasks on to the domain layer, which is responsible for handling them.

5. How to Apply the GRASP Patterns?

The following sections present the first five GRASP patterns:

- Information Expert
- Creator
- High Cohesion
- Low Coupling
- Controller

6. Define Responsibilities and Methods.

The UML defines a responsibility as "a contract or obligation of a classifier". Responsibilities are related to the obligations of an object in terms of its behavior. Basically, these responsibilities are of the following two types:

- knowing
-doing

7. List out some scenarios that illustrate varying degrees of functional cohesion.

-Very low cohesion
-low cohesion
-High cohesion
-Moderate cohesion

SAEC

8. Define Modular Design.

Coupling and cohesion are old principles in software design; designing with objects does not imply ignoring well-established fundamentals. Another of these, which is strongly related to coupling and cohesion is to promote modular design.

9. What are the advantages of Factory objects?

- Separate the responsibility of complex creation into cohesive helper objects.
- Hide potentially complex creation logic.
- Allow introduction of performance-enhancing memory management strategies, such as object caching or recycling.

10. What is meant by Abstract Class Abstract Factory?

A common variation on Abstract Factory is to create an abstract class factory that is accessed using the Singleton pattern, reads from a system property to decide which of its subclass factories to create, and then returns the appropriate subclass instance. This is used, for example, in the Java libraries with the *java.awt.Toolkit* class, which is an abstract class abstract factory for creating families of GUI widgets for different operating system and GUI subsystems.

11. Differentiate coupling and cohesion.

Coupling deals with interactions between objects or software components while cohesion deals with the interactions within a single object or software component.

12. What is meant by Fine-Grained Classes?

Consider the creation of the *Credit Card, Drivers License,* and *Check* software objects. Our first impulse might be to record the data they hold simply in their related payment classes, and eliminate such fine-grained classes. However, it is usually a more profitable strategy to use them; they often end up providing useful behavior and being reusable. For example, the *Credit Card* is a natural Expert on telling you its credit company type (Visa, MasterCard, and so on).

13. Define coupling.

The degree to which components depend on one another. There are two types of coupling, "tight" and "loose". Loose coupling is desirable for good software engineering but tight coupling may be necessary for maximum performance. Coupling is increased when the data exchanged between components becomes larger or more complex.

14. What do you mean by degree of coupling?

` The degree of coupling is a function of How complicated the connection is.

Whether the connection refers to the object itself or something inside it. What is being sent or received.

The degree or strength of coupling between two components is measured by the amount and complexity of information transmitted between them. Coupling increases with increasing

complexity and decreases when the connection is to the component interface rather than to an internal component. Coupling is also lower for data connections than for control connections.

15. What do you mean by cohesion? Give the types of cohesion.

Cohesion can be defined as the interactions within a single object or software component. Cohesion reflects the single purposeness of an object. Cohesion helps in designing classes that have very specific goals and clearly defined purposes.

- Method cohesion
- Class cohesion
- Inheritance cohesion

16. What do you mean by design patterns?

Design patterns are devices that allow systems to share knowledge about their design, by describing commonly recurring structures of communicating components that solve a general design problem within a particular context. A design pattern provides a scheme for refining the subsystems or components of a software system or the relationships among them. Design patterns are documented by writing essays in a fairly well-defined form.

17. What are the three basic types of attributes?

The three basic types of attributes are

Single-value attributes.

The single-valued attribute has only one value or state.

Multiplicity or multi value attributes.

The multiplicity or multi valued attribute can have a collection of many values at any point in time.

Reference to another object, or instance connection.

These attributes are required to provide the mapping needed by an object to fulfill its responsibilities, in other words, instance connection model association.

18. What is a Metaphor?

It is an analogy that relates two unrelated things by using one to denote the other.

19. Give the three UI design rules.

UI design rule 1: Making the interface simple.
UI design rule 2: Making the interface transparent and natural.
UI design rule 3: Allowing users to be in control of the software.

20. Define Package.

A package groups and manages the modeling elements, such as classes, their associations, and their structures. Packages themselves may be nested within other packages. A package may contain both other packages and ordinary model elements. The entire system description can be thought of as a single high-level sub-system package with everything else init. All kinds of UML model elements and diagrams can be organized into packages.

21. What is concurrency policy?

A concurrency control policy dictates what happens when conflicts arise between transactions that attempt access to the same object and how these conflicts are to be resolved There are two policies:

Conservative or pessimistic policy

Allows a user to lock all objects or records when they are accessed and to release the locks only after a transaction commits.

Optimistic policy

Two conflicting transactions are compared in their entirety and then their serial ordering is determined.

22. What is meant by Low Coupling?

Coupling is a measure of how strongly one element is connected to, has knowledge of, or relies on other elements. An element with low (or weak) coupling is not dependent on too many other elements; "too many" is context-dependent, but will be examined. These elements include classes, subsystems, systems, and so on.

23. What is meant by High cohesion?

Cohesion (or more specifically, functional cohesion) is a measure of how strongly related and focused the responsibilities of an element are. An element with highly related responsibilities, and which does not do a tremendous amount of work, has high cohesion. These elements include classes, subsystems, and so on.

24. Define Controller.

Assign the responsibility for receiving or handling a system event message to a class representing one of the following choices:

- Represents the overall system, device, or subsystem *(facade controller).*
- Represents a use case scenario within which the system event occurs, often named <UseCaseName>Handler, <UseCaseName>Coordinator, or <Use-CaseName>Session *(use-case or session controller).*
- Use the same controller class for all system events in the same use case scenario.
-Informally, a session is an instance of a conversation with an actor.
-Sessions can be of any length, but are often organized in terms of use cases.

25. What is meant by CRC card?

CRC cards are index cards, one for each class, upon which the responsibilities of the class are briefly written, and a list of collaborator objects to fulfill those responsibilities. They are usually developed in a small group session. The GRASP patterns may be applied when considering the design while using CRC cards.

26. What is meant by Pure Fabrication?

This is another GRASP pattern. A Pure Fabrication is an arbitrary creation of the designer, not a software class whose name is inspired by the Domain Model. A use-case controller is a kind of Pure Fabrication.

27. Define Responsibilities and Methods.

The UML defines a responsibility as "a contract or obligation of a classifier". Responsibilities are related to the obligations of an object in terms of its behavior.

Basically, these responsibilities are of the following two types:

- knowing
-doing

Doing responsibilities of an object include:

- doing something itself, such as creating an object or doing a calculation
- initiating action in other objects
-controlling and coordinating activities in other objects

Knowing responsibilities of an object include:

- knowing about private encapsulated data
- knowing about related objects
- knowing about things it can derive or calculate

28. Who is creator?

Solution Assign class B the responsibility to create an instance of class A if one or more of the following is true:

. B *aggregates* an object.
. B *contains* an object.
. B *records* instances of objects.
. B *closely uses* objects.
. B *has the initializing data* that will be passed to A when it is created.

B is a *creator* of an object.
If more than one option applies, prefer a class B which *aggregates* or *contains* class A.

SAEC

UNIT-III
CASE STUDY

1. What is Elaboration?
Elaboration is the initial series of iterations during which the team does serious investigation, implements (programs and tests) the core architecture, clarifies most requirements, and tackles the high-risk issues. In the UP, "risk" includes business value. Therefore, early work may include implementing scenarios that are deemed important, but are not especially technically risky.

2. What are the tasks performed in elaboration?
- the core, risky software architecture is programmed and tested
- the majority of requirements are discovered and stabilized
- the major risks are mitigated or retired

3. What are the key ideas and best practices that will manifest in elaboration?
- do short time boxed risk-driven iterations
- start programming early
- adaptively design, implement, and test the core and risky parts of the architecture
- test early, often, realistically
- adapt based on feedback from tests, users, developers
- write most of the use cases and other requirements in detail, through a series of workshops, once per elaboration iteration.

4. What are the key ideas for Planning the Next Iteration?
Organize requirements and iterations by risk, coverage, and criticality.

5. What is a Domain Model?
A domain model is a visual representation of conceptual classes or real-situation objects in a domain. The term "Domain Model" means a representation of real-situation conceptual classes, not of software objects. The term does not mean a set of diagrams describing software classes, the domain layer of a software architecture, or software objects with responsibilities.

6. **How the domain model is illustrated?**
Applying UML notation, a domain model is illustrated with a set of class diagrams in which no operations (method signatures) are defined. It provides a conceptual perspective.
It may show:
- domain objects or conceptual classes
- associations between conceptual classes
- attributes of conceptual classes

7. Why Call a Domain Model a "Visual Dictionary"?
The information it illustrates could alternatively have been expressed in plain text. But it's easy to understand the terms and especially their relationships in a visual language, since our brains are good at understanding visual elements and line connections. Therefore, the domain model is

a visual dictionary of the noteworthy abstractions, domain vocabulary, and information content of the domain.

8. What are the elements not suitable in a domain model?

The following elements are not suitable in a domain model

- Software artifacts, such as a window or a database, unless the domain being modeled is of software concepts, such as a model of graphical user interfaces.
- Responsibilities or methods

9. What are Conceptual Classes?

The domain model illustrates conceptual classes or vocabulary in the domain. Informally, a conceptual class is an idea, thing, or object. More formally, a conceptual class may be considered in terms of its symbol, intension, and extension

- Symbol words or images representing a conceptual class.
- Intension the definition of a conceptual class.
- Extension the set of examples to which the conceptual class applies

10. How to Create a Domain Model?

The current iteration requirements under design:

1. Find the conceptual classes (see a following guideline).
2. Draw them as classes in a UML class diagram.
3. Add associations and attributes.

11. How to Find Conceptual Classes?

1. Reuse or modify existing models.
2. Use a category list.
3. Identify noun phrases.

12. List the relationships used in class diagram?

- Generalization(class to class)
- Association (object to object)
- Aggregation (object to object)
- Composition (object to object)

13. Define Association.

An **association** is a relationship between classes (more precisely, instances of those classes) that indicates some meaningful and interesting connection.

14**. How to Name an Association in UML?**

Name an association based on a Class Name-Verb Phrase-Class Name format where the verb phrase creates a sequence that is readable and meaningful.

SAEC

15. What is Aggregation?

Aggregation is a vague kind of association in the UML that loosely suggests whole-part relationships (as do many ordinary associations). It has no meaningful distinct semantics in the UML versus a plain association, but the term is defined in the UML.

16. What is composition?

Composition, also known as composite aggregation, is a strong kind of whole-part aggregation and is useful to show in some models. A composition relationship implies that

1) an instance of the part (such as a Square) belongs to only one composite instance (such as one Board) at a time,
2) the part must always belong to a composite (no free-floating Fingers), and
3) the composite is responsible for the creation and deletion of its parts either by itself creating/deleting the parts, or by collaborating with other objects.

17. Mention the guidelines that suggest when to show aggregation.

- The lifetime of the part is bound within the lifetime of the composite there is a create-delete dependency of the part on the whole.
- There is an obvious whole-part physical or logical assembly.
- Some properties of the composite propagate to the parts, such as the location.
- Operations applied to the composite propagate to the parts, such as destruction, movement, and recording.

18. Define Attributes.

An attribute is a logical data value of an object.

19. When Are Contracts Useful?

The use cases are the main repository of requirements for the project. They may provide most or all of the detail necessary to know what to do in the design, in which case, contracts are not helpful. However, there are situations where the details and complexity of required state changes are awkward to capture in use cases.

20. Mention the Guidelines for Contracts.

To make contracts:

1. Identify system operations from the SSDs.
2. For system operations that are complex and perhaps subtle in their results, or which are not clear in the use case, construct a contract.
3. To describe the post conditions, use the following categories:
   - Instance creation and deletion
   - attribute modification
   - Associations formed and broken

SAEC

Unit-IV
APPLYING DESIGN PATTERNS

1. What is meant by System Sequence Diagrams?
A system sequence diagram (SSD) is a picture that shows, for a particular scenario of a use case, the events that external actors generate their order, and inter-system events. All systems are treated as a black box; the emphasis of the diagram is events that cross the system boundary from actors to systems.

2. What is meant by System Behavior?
System behavior is a description *of what* a system does, without explaining how it does it. One Part of that description is a system sequence diagram. Other parts include the Use cases, and system contracts.

3. What is meant by Inter-System SSDs?
SSDs can also be used to illustrate collaborations between systems, such as between the Next Gen POS and the external credit payment authorizer. However, this is deferred until a later iteration in the case study, since this iteration does not include remote systems collaboration.

4. Define System Events and the System Boundary.
To identify system events, it is necessary to be clear on the choice of system boundary, as discussed in the prior chapter on use cases. For the purposes of software development, the system boundary is usually chosen to be the software system itself; in this context, a system event is an external event that directly stimulates the software.

5. How to Naming System Events and Operations?
System events (and their associated system operations) should be expressed at the level of intent rather than in terms of the physical input medium or interface widget level. It also improves clarity to start the name of a system event with a verb Thus "enter item" is better than "scan" (that is, laser scan) because it captures the intent of the operation while remaining abstract and noncommittal with respect to design choices about what interface is used to capture the system event.

6. What is meant by interaction diagram?
The term *interaction diagram* is a generalization of two more specialized UML diagram types; both can be used to express similar message interactions:

. Collaboration diagrams
. Sequence diagrams

7. What is meant by link?
A link is a connection path between two objects; it indicates some form of navigation and visibility between the objects is possible . More formally, a link is an instance of an association. For example, there is a link or path of navigation from a *Register* to a *Sale*, along which messages may flow, such as the *make 2 Payment* message.

8. What is meant by Messages?

Each message between objects is represented with a message expression and small arrow indicating the direction of the message. Many messages may flow along this link. A sequence number is added to show the sequential order of messages in the current thread of control.

9. How to create an instance?

Any message can be used to create an instance, but there is a convention in the UML to use a message named *create* for this purpose. If another (perhaps less obvious) message name is used, the message may be annotated with a special feature called a UML stereotype, like so: *«create»*.
The *create* message may include parameters, indicating the passing of initial values. This indicates, for example, a constructor call with parameters in Java.

10. How to Choosing the Initial Domain Object?

Choose as an initial domain object a class at or near the root of the containment or aggregation hierarchy of domain objects. This may be a facade controller, such as *Register*, or some other object considered to contain all or most other objects, such as a *Store*.

11. How to Connecting the UI Layer to the Domain Layer?

• An initializing routine (for example, a Java *main* method) creates both a UI and a domain object, and passes the domain object to the UI.

• A UI object retrieves the domain object from a well-known source, such as a factory object that is responsible for creating domain objects.

12. Mention the Interface and Domain Layer Responsibilities.

The UI layer should not have any domain logic responsibilities. It should only be responsible for user interface tasks, such as updating widgets. The UI layer should forward requests for all domain-oriented tasks on to the domain layer, which is responsible for handling them.

13. List out some scenarios that illustrate varying degrees of functional cohesion.

-Very low cohesion
-low cohesion
-High cohesion
-Moderate cohesion

14. Define Modular Design.

Coupling and cohesion are old principles in software design; designing with objects does not imply ignoring well-established fundamentals.

15. What are the advantages of Factory objects?

• Separate the responsibility of complex creation into cohesive helper objects.

• Hide potentially complex creation logic.

• Allow introduction of performance-enhancing memory management strategies, such as object caching or recycling.

SAEC

16. Designing for Non-Functional or Quality Requirements.
Interestingly—and this a key point in software architecture—it is common that the large-scale themes, patterns, and structures of the software architecture are shaped by the designs to resolve the non-functional or quality requirements, rather than the basic business logic.

17. Abstract for Factory (GoF) for Families of Related Objects.
The Java POS implementations will be purchased from manufacturers.
For example:
// IBM's drivers
com.ibm.pos.jpos.CashDrawer (implements jpos.CashDrawer)
com.ibm.pos.jpos.CoinDispenser (implements jpos.CoinDispenser)
// NCR's drivers
com.ncr.posdrivers.CashDrawer (implements jpos.CashDrawer)
com.ncr.posdrivers.CoinDispenser (implements jpos.CoinDispenser)

18. What is meant by Abstract Class Abstract Factory?
A common variation on Abstract Factory is to create an abstract class factory that is accessed using the Singleton pattern, reads from a system property to decide which of its subclass factories to create, and then returns the appropriate subclass instance. This is used, for example, in the Java libraries with the *java.awt.Toolkit* class, which is an abstract class abstract factory for creating families of GUI widgets for different operating system and GUI subsystems.

19. What is meant by Fine-Grained Classes?
Consider the creation of the *Credit Card, Drivers License,* and *Check* software objects. Our first impulse might be to record the data they hold simply in their related payment classes, and eliminate such fine-grained classes. However, it is usually a more profitable strategy to use them; they often end up providing useful behavior and being reusable.

20. **How will you create Class Definitions from DCDs?**
At the very least, DCDs depict the class or interface name, super classes, method signatures, and simple attributes of a class. This is sufficient to create a basic class definition in an object-oriented programming language. Later discussion will explore the addition of interface and namespace (or package) information, among other details.

21. Creating Class Definitions from DCDs.
At the very least, DCDs depict the class or interface name, super classes, method signatures, and simple attributes of a class. This is sufficient to create a basic class definition in an object oriented programming language. Later discussion will explore the addition of interface and namespace (or package) information, among other details.

22. What are the Benefits of Iterative Development?
- Early rather than late mitigation of high risks (technical, requirements, objectives, usability, and so forth)

• Early visible progress
• Early feedback, user engagement, and adaptation, leading to a refined system that more closely meets the real needs of the stakeholders
• managed complexity; the team is not overwhelmed by "analysis paralysis" or very long and complex steps
• The learning within iteration can be methodically used to improve the development process itself, iteration by iteration

23. Define Events, States, and Transitions.

An event is a significant or noteworthy occurrence.
For example:
A telephone receiver is taken off the hook.
A state is the condition of an object at a moment in time—the time between events.
For example:
• A telephone is in the state of being "idle" after the receiver is placed on the hook and until it is taken off the hook.
A transition is a relationship between two states that indicates that when an event occurs, the Object moves from the prior state to the subsequent state.
For example:
• When the event "off hook" occurs, transition the telephone from the "idle" to "active" state.

24. What is meant by State chart Diagrams?

A UML state chart diagram, illustrates the interesting events and states of an object, and the behavior of an object in reaction to an event. Transitions are shown as arrows, labeled with their event. States are shown in rounded rectangles. It is common to include an initial pseudo-state, which automatically transitions to another state when the instance is created.

25. State chart Diagrams in the UP?

There is not one model in the UP called the "state model." Rather, any element in any model (Design Model, Domain Model, and so forth) may have a state chart to better understand or communicate its dynamic behavior in response to events. For example, a state chart associated with the *Sale* design class of the Design Model is itself part of the Design Model.

26. Utility of Use Case State chart Diagrams.

- Hard-coded conditional tests for out-of-order events
- Use of the *State* pattern (discussed in a subsequent chapter)
- disabling widgets in active windows to disallow illegal events (a desirable approach)
- A state machine interpreter that runs a state table representing *a* use case state chart diagram.

27. Define External event.

External event—also known as a system event, is caused by something (for example, an actor) outside our system boundary. SSDs illustrate external events. Noteworthy external events precipitate the invocation of system operations to respond to them.

28. Define internal event.

Internal event—caused by something inside our system boundary. In terms of software, an internal event arises when a method is invoked via a message or signal that was sent from another internal object. Messages in interaction diagrams suggest internal events.

29. Define temporal event.

Temporal event—caused by the occurrence of a specific date and time or passage of time. In terms of software, a temporal event is driven by a real time or simulated-time clock.

SAEC

Unit-V
CODING AND TESTING

1. What are Steps for Mapping Designs to Code?

Implementation in an object-oriented programming language requires writing source code for:

- Class and interface definitions
- Method definitions

2. List the levels of object oriented testing.

- Operation / Method
- Class
- Integration
- System Testing

3. Define unit.

- a single, cohesive function
- a function which, when coded, fits on one page
- the smallest separately compilable segment of code
- the amount of code that can be written in 4 to 40 hours
- a task in a work breakdown structure
- code that is assigned to one person

4. What is the purpose of unit testing?

The goal of unit testing is to verify that, taken by itself, the unit functions correct.

5. Define integration testing.

Integration testing (sometimes called integration and testing, abbreviated I&T) is the phase in software testing in which individual software modules are combined and tested as a group. It occurs after unit testing and before validation testing.

6. Define MM- path.

A Method/Message Path (MM-Path) is a sequence of method executions linked by messages. An MM-Path starts with a method and ends when it reaches a method which does not issue any messages of its own.

7. Define GUI testing.

GUI testing is the process of ensuring proper functionality of the graphical **user interface** (**GUI**) for a given application and making sure it conforms to its written specifications.

8. What are the difficulties in GUI testing.

a. GUI test automation is difficult
b. Often GUI test automation is technology-dependent
c. Observing and trace GUI states is difficult

SAEC

d. UI state explosion problem
e. Controlling GUI events is difficult
f. GUI test maintenance is hard and costly

SAEC