OBJECT ORIENTED ANALYSIS AND DESIGN

UNIT – 1

PART – A

1. 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.

2. 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. The combination of these two concepts shortly known as object oriented analysis and design.

3. **What is the UML?**
The Unified Modeling Language is a visual language for specifying, constructing and documenting the artifacts of systems.

4. **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.

5. **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

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

7. **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.

8. **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.

9. **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.

10. **What is the main advantage of object-oriented development?**
• High level of abstraction
• Seamless transition among different phases of software development
• Encouragement of good programming techniques.
• Promotion of reusability.

PART – B

1. Elaborate use case modelling process with suitable examples.
2. With suitable example explain the use case include relationship and extend relationship.
3. Define Unified Process Model? Exemplify the iterations, outcomes and workflow in unified Process Model with neat sketch.
4. Deliberate different notations of UML diagrams in detail.
5. Explain about the phases of unified process model

UNIT – 2

PART – A

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 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.

5. **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

6. **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

7. **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.

8. **Define Association.**

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

9. **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.

10. **What is an activity diagram?**

A UML activity diagram shows sequential and parallel activities in a process. They are useful for modelling 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

PART – B

1. Differentiate Elaboration and Inception. List any five and artifacts related to Inception.

2. With an illustration, explain the class hierarchies. Also state the guidelines for defining a super class.

3. Explain different categories of conceptual classes with examples and discuss the three strategies to find a conceptual class.

4. Describe in detail, the Associations, Attributes, Aggregation and Composition give suitable example.

5. Compare cohesion and coupling with suitable example

UNIT – 3

PART – A

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. **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.

3. **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

4. **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

5. **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.

6. **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.

7. **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.

8. **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.

9. **List the relationships used in class diagram?**
- Generalization(class to class)
- Association (object to object)
- Aggregation (object to object)
- Composition

10. **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 (use case sessions).

PART – B

1. Justify the need for component and deployment diagrams with a suitable real time example.

2. Differentiate state independent and state dependent objects. How to model them using State Machine Diagrams ?

3. Consider the Library management system. It should provide the facility to issue the book, to calculate the fine during book return, the placement of the books on the shelfs, adding new books to the shelfs and removing old books from the shelfs. Draw the activity diagram with swim lanes for each and every components of the above scenario.

4. Consider a House keeping system in a five star hotel. Draw a sequence and collaboration diagram for the given scenario.

5. Describe use case modeling for student information system

UNIT – 4

PART – A

1. **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.*

2. **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.

3. **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.

4. **How to Apply the GRASP Patterns?**
The following sections present the first five GRASP patterns:
. Information Expert
. Creator
. High Cohesion
. Low Coupling
. Controller

5. **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 (thus B is an Expert with
respect to creating A).
B is a *creator* of an object.
If more than one option applies, prefer a class B which *aggregates* or *contains* class A.

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

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

7. **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.

8. **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

9. **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.

10. **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). This behavior will turn out to be necessary for our application.

PART – B

1. What is GRASP ? List and explain the nine object oriented design principles.

2. With an illustrated example diagram, brief on adapter pattern.

3. Illustrate and provide an interface for creating families of related or dependent objects without specifying their concrete classes using factory method.

4. Elucidate Creator pattern and controller pattern with real time examples.

5. Describe MVC pattern and draw MV model for a game application

UNIT – 5

PART – A

1. **Define post condition.**
The post conditions describe changes in the state of objects in the Domain Model. Domain Model state changes include instances created, associations formed or broken, and attributes changed.

2. **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

3. **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

4. **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.

5. **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

6. **Define Events, States, and Transitions. APRIL/MAY-2011**

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.

7. **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.

8. **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.

9. **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.
-Suppose that after an *end Sale* operation occurs, a *make Payment* operation must occur within five minutes, otherwise the current sale is automatically purged.

10. **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.
- When *a Sale* receives a *make Line item message*, an internal event has occurred.

PART – B

1. What are test cases ? List the guidelines for developing quality assurance test cases.

2. Suggest strategies to carry out unit testing and integration testing in an object oriented development environment

3. Explicate in detail the various testing strategies and the impact of object orientation on testing.

4. Illustrate with neat sketch the software development life cycle of object-oriented system.

5. Explain briefly about testing issues in OO testing and class testing