NEERAJ®

M.C.S.-219

Object Oriented Analysis and Design

Chapter Wise Reference Book
Including Many Solved Sample Papers

Based on

I.G.N.O.U.

& Various Central, State & Other Open Universities

By: *Anand Prakash Srivastava*

NEERAJ PUBLICATIONS
(Publishers of Educational Books)

Mob.: 8510009872, 8510009878 E-mail: info@neerajbooks.com
Website: www.neerajbooks.com

MRP ₹ 350/-

Content

OBJECT ORIENTED ANALYSIS AND DESIGN

S.No.	Chapterwise Reference Book	Page

■■

Sample Preview of the Solved Sample Question Papers

Published by:

NEERAJ PUBLICATIONS

www.neerajbooks.com

QUESTION PAPER

December – 2023

(Solved)

OBJECT ORIENTED ANALYSIS AND DESIGN

M.C.S.-219

Time: 3 Hours] ***[Maximum Marks : 100***

Note: *(i) Question No. 1 is compulsory. (ii) Attempt any three questions from the rest.*

Q. 1. ***(a)*** **"Object Oriented Analysis and Design is better than Structured Analysis and Design." Explain in detail. Justify the above statement.**

Ans. The comparison between Object-Oriented Analysis and Design (OOAD) and Structured Analysis and Design (SA/SD) is a topic of ongoing debate in the field of software engineering. Both methodologies have their strengths and weaknesses, and the choice between them often depends on the specific requirements, complexity, and nature of the software project. However, in many cases, Object-Oriented Analysis and Design is considered superior or more advantageous compared to Structured Analysis and Design. Let's delve into the details and justify this statement.

Object-Oriented Analysis and Design (OOAD):

1. Modularity and Reusability: OOAD emphasizes modularity by organizing software components into objects that encapsulate both data (attributes) and behaviour (methods). This modular approach promotes reusability, as objects can be reused in different parts of the system or in future projects, leading to more efficient and maintainable code.

2. Abstraction and Encapsulation: OOAD uses abstraction to model real-world entities as objects with simplified representations, focusing on essential properties and behaviours. Encapsulation hides the internal implementation details of objects, allowing changes to be made without affecting other parts of the system.

3. Inheritance and Polymorphism: OOAD leverages inheritance to create hierarchical relationships between classes, allowing subclasses to inherit attributes and behaviours from superclasses. This promotes code reuse and supports the "is-a" relationship. Polymorphism enables objects to respond differently to the same message, improving flexibility and extensibility.

4. Object Modeling Techniques: OOAD employs modeling techniques such as Unified Modeling Language (UML) to visualize and document system components, relationships, and interactions. UML diagrams (e.g., class diagrams, sequence diagrams) provide a clear and comprehensive representation of the system's structure and behaviour.

5. Real-World Modeling: OOAD focuses on modeling software systems based on real-world entities and their interactions, leading to more intuitive and user-friendly designs. This approach facilitates communication between stakeholders and developers, ensuring that the software meets the intended requirements.

Structured Analysis and Design (SA/SD):

1. Procedural Approach: SA/SD follows a procedural or functional decomposition approach, breaking down the system into functions or procedures that operate on data structures. While this approach can be effective for certain types of systems, it may lead to complex and tightly coupled designs in larger or more complex projects.

2. Data Flow and Process Modeling: SA/SD emphasizes data flow diagrams (DFDs) and process modeling to represent system processes, data flows, and data stores. While these techniques provide a structured way to analyze and design systems, they may lack the flexibility and modularity of object-oriented approaches.

3. Limited Reusability: In SA/SD, the emphasis is more on designing algorithms and procedures rather than reusable components. This can result in code duplication and limited reusability, especially when similar functionalities need to be implemented in multiple parts of the system.

4. Less Scalable: SA/SD may face scalability challenges when dealing with complex systems or evolving requirements. The procedural nature of the approach can make it harder to accommodate changes and enhancements without impacting the entire system.

Justification for OOAD Over SA/SD:

1. Modularity and Reusability: OOAD promotes better modularity and reusability through object-oriented principles such as encapsulation, inheritance, and polymorphism. This leads to more maintainable, extensible, and scalable software designs.

2. Real-World Modeling: OOAD's focus on real-world modeling results in software systems that closely align with users' mental models and requirements, improving usability and user satisfaction.

QUESTION PAPER

June – 2023

(Solved)

OBJECT ORIENTED ANALYSIS AND DESIGN

M.C.S.-219

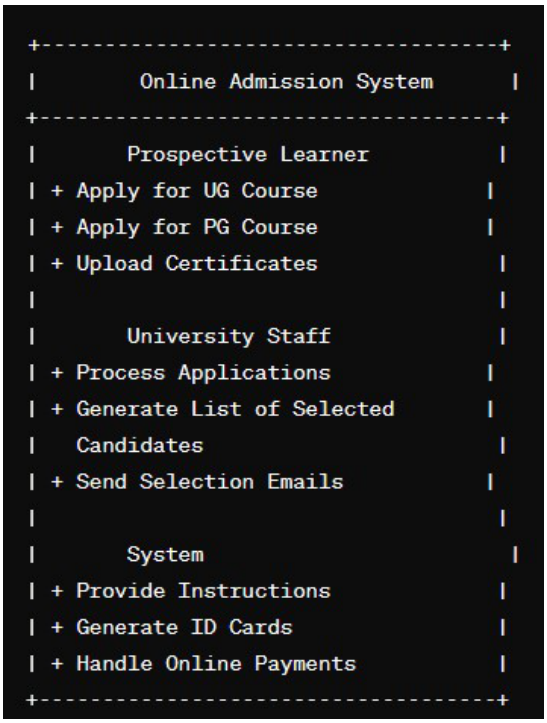
Time: 3 Hours] ***[Maximum Marks : 100***

Note: *(i) Question No. 1 is compulsory. (ii) Attempt any three questions from the rest.*

Q. 1. ***(a)*** **An online admission system of a University provides facility to its prospective learner to apply for various UG and PG courses. During the application process applicants need to provide their basic details such as name, date of birth, mobile number, email-id and address. Also they need to upload their certificates for which system provides proper instructions and interfaces. Subsequent upon processing the applications received, university displays the list of selected candidates and also send them email regarding their selection. Applicants pay the fee online and their id cards are generated. Draw the following diagrams for this system. (You can make necessary assumptions, if required):**

(i) **Use case diagram**

Ans.

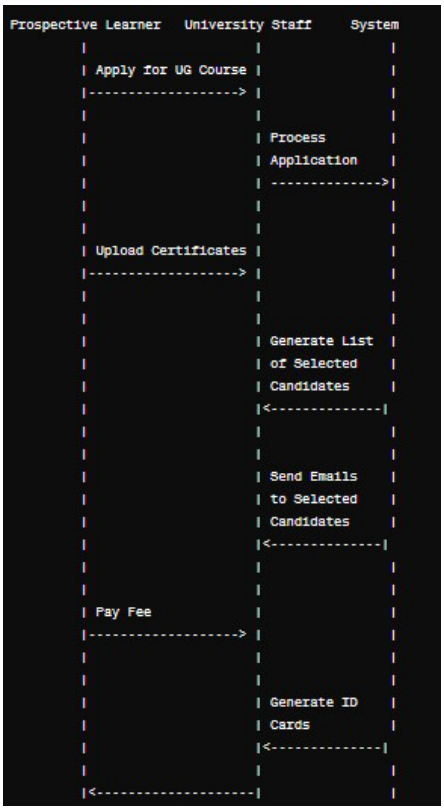


(ii) **Class diagram**

Ans.

(iii) **Sequence diagram**

Ans.



Sample Preview of The Chapter

Published by:

NEERAJ PUBLICATIONS

www.neerajbooks.com

OBJECT ORIENTED ANALYSIS AND DESIGN

Introduction to Object Oriented Modeling

1

INTRODUCTION

A method for creating software systems employing concepts from object-oriented software engineering is called object-oriented analysis and design (OOAD). OOAD emphasises having numerous possibilities. There may be more than one right answer to a problem; there is never just one. The more options you have, the more likely you are to discover an effective solution for any issue. Your system must comply with requirements in order to function properly. The first technical phase in creating a software system using the object-oriented methodology is called object-oriented analysis, or OOA. Based on a set of fundamental ideas, OOA models the issue domain, represents behaviour, and describes system operations. The model produced with OOA is converted into an object-oriented design (OOD) model, which serves as a blueprint for software development. The 1980s saw the emergence of object-oriented design and analysis techniques, while the 1990s saw the emergence of object-oriented analysis techniques. Creating graphical user interfaces (GUIs) and a few other applications were the main early uses of object orientation. "One should model software systems as collections of cooperating objects, treating individual objects as instances of a class within a hierarchy of classes," is a fundamental OOAD principle.

CHAPTER AT A GLANCE

INTRODUCTION TO OBJECT ORIENTATION

The Object Modeling Technique (OMT) for software modeling and designing emerged as the most common OOAD approach after the concept of OOAD was introduced in the 1980s and early 1990s.

In 1991, James Rambaugh made the suggestion. For software modeling, he has advised using three different sorts of models: object, dynamic, and functional.

Another strategy was the Grady Booch-proposed Booch methodology of Object-Oriented Analysis and Design.

The OOAD methodology follows a progressive process. In the analysis phase, the system requirements are established, and domain analysis is done from the viewpoint of the customer. After the analytical stage is over, the iterative design step links the logic design to the physical design. The prototype is built and put to the test. It is a strategy that is frequently employed in Object-Oriented Software Engineering. Ivar Jacobson introduced a popular method for Object-Oriented Software Engineering in 1992. Use case diagrams were used in design for the first time as part of an object-oriented design process. Requirements collection, analysis, design, implementation, and testing are among the aspects of his technique. Additionally, he invented state transition diagrams to show how an object's state changes during execution and interaction diagrams to show how operations are performed in real-time. In the area of object-oriented software engineering, they are the pioneers.

Object-Oriented Modeling and Design: It consists of the linked, but separate phases of analysis and design. Identifying requirements and creating an object model for the application domain are the first steps in the OOAD process; during analysis, software specifications are created. In this stage, multiple models are used in an effort to grasp the issue by examining concepts from the outside world.

The next phase is OOD, where the object is a crucial construct. A class instance, or object, also has a data structure in addition to its behavioural traits. Objects can interact with one another to create a variety of programmes and applications.

Object Oriented Modeling is divided into various stages: The system's OOM goes through the following stages:

- System requirement analysis,
- System design,
- Detailed design with an emphasis on system objects, and
- Model implementation.

The design approach for an object-oriented development model is shown in Figure 1.

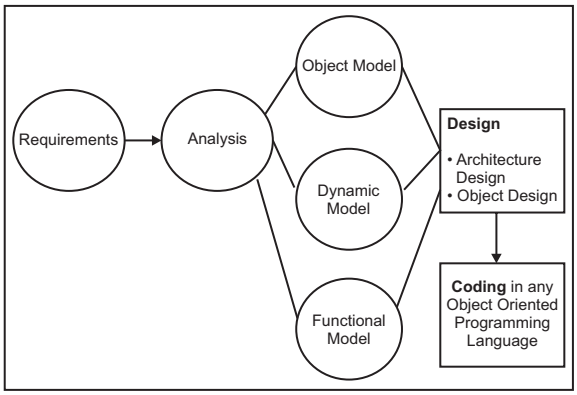


Fig. 1: Object Oriented Development Model

System Analysis: The analyst creates a problem statement and a system model to simulate real-world scenarios. This phase reveals the situation's essentials. The analysis model is a clear, precise abstraction and consensus on "how the desired system must be developed". The goal is to give a model that application specialists, whether programmers or not, can understand and criticise. This phase defines all system needs and properties. This step aims to create a model that novice and expert users can understand.

System Design: This step designs the system architecture. The system analysis model and suggested architecture divide the system into subsystems during system design. This phase defines and designs the system's architecture using the analytical model. The analytical model and system architecture divide large, complicated systems into discrete subsystems.

Object Design: Based on the analytical model from before, a design model is created. The object design uses the analysis model's implementation details to choose data structures and algorithms for each class.

Following are the steps involved in object design:

- The identification and completion of operations and procedures carried out on distinct classes.
- The reasoning needed to carry out the method is chosen.
- All data retrieval routes have been optimised.
- For all the allowed external interference, a list is created.
- In order to increase the inheritance properties across classes, classes and the operations related with them are moved around or modified.
- All relevant classes are packaged in a single module.

Implementation: Using any programming language, classes and relationships defined during object design are finalised. This phase implements database and hardware (if needed). Implementation requires software engineering. This creates an adaptable system.

Object Modeling Technique (OMT): Three different types of models are used throughout object-oriented modeling to cover the entire system description:

- Object model,
- Dynamic model, and
- Function model.

The items in the system and their connections with one another are described using object models. The dynamic model defines how things interact with one another and how information moves through the system. A functional model explains how the system transforms data. All three types can be used at any stage of development. These models are in charge of gathering system development implementation specifics.

Let's compare object-oriented development to structured system development before discussing object-oriented modeling.

Unlike the structured approach, the object-oriented approach identifies application domain objects and associate's procedures (methods) around them.

Object-oriented development takes a holistic perspective of the application domain and identifies objects in the problem domain, making it indirect system development. The application's history helps understand its situations and traits. Object-oriented programming benefits from considering the problem domain as a whole rather than its functional requirements. During problem-solving, objects pass messages.

Object Modeling: Each object has certain characteristics and behaviours. Typical examples of objects include things like a car, a student, a book, a teacher, an employee, etc. We can carry out a wide range of operations on the object, such as classifying, describing, organising, combining, and manipulating it. Class diagrams and object diagrams can be used to graphically depict the object model.

Dynamic Modeling: It is focused on Time, States, and Transition between States. Events that cause the transition, immediate actions that took place after the event.

Functional Modeling: Information is transformed and processed in functional modeling. The data flow from one operation to another is depicted by it. Data Flow Diagrams (DFDs) aid in the design of the entire

system. Entities, Data Store, Processes or Operations, File Store, and Data Flow make up the bulk of the DFD.

BASIC PHILOSOPHY OF OBJECT ORIENTATION

The attributes of object-oriented technology are numerous. Let's now talk about the fundamental traits of object-oriented systems.

Sharing of Structure and Behaviour: Object-oriented methods encourage multi-level sharing, which makes them popular. Object-oriented languages benefit from inheritance because it allows code reuse.

Inheritance of data structure and behaviour allows a base class to be used to construct several subclasses based on its basic traits and develop new classes with less effort.

Object-oriented development promotes application information sharing and reuse and provides a foundation for project enhancement. New features can be added as extensions of fundamental features as needed. Inheritance does this without large code changes. Object orientation does not guarantee reusability and improvement. General system design ensures reusability and enhancement. If the system is understood and its aspects explored, this design can be created.

Emphasis on Object Structure, not on Operation Implementation: Object-orientation emphasises specifying rather than implementing object features. The application and development modifications determine an object's usage. Object features are more stable than their uses as requirements grow. Object-structured software is more secure.

The object-oriented approach should focus on the system's objects' core qualities rather than its process structure. This process contemplates "what an object is and its role in the system."

OOAD creates great software. In truth, OOAD lets programmers and customers develop well-designed applications.

PRINCIPLES OF OBJECT ORIENTATION

The OOAD process makes use of OO principles. These guidelines give software more adaptability, maintainability, and extensibility. Additionally, OOAD applications of these ideas offer reusable design. We shall examine OO fundamentals in this part.

Abstraction

Object-oriented systems rely on abstraction. Abstraction emphasises a system object's core qualities. It does not reflect system accidentals. Abstraction helps system developers decide what an object should perform before implementing it. Avoiding intermediary commitments in problem-solving through abstraction prolongs decision-making freedom. Contemporary languages abstract data. The abstraction gives system developers more freedom to employ inheritance and polymorphism. Abstraction analysis requires application-domain notions. You can design and implement afterwards.

Abstraction is everywhere, but we don't think of it that way. One popular mobile phone interaction example. Open an app and hit a button to call. You're doing a straightforward operation, yet a lot is happening that you don't need to know as a user. OOM uses that abstraction.

Encapsulation

Encapsulation hides data we don't want users to see and shows data we do. It separates an object's exterior from its interior implementation. It conceals object properties. Encapsulation hides internal implementation aspects that don't affect the outside world. Data structure and behaviour can be encapsulated. Encapsulation improves systems. Encapsulation helps update an object's implementation without changing its appearance.

Inheritance

Object-oriented development uses code inheritance. We group comparable classes during modeling to enforce code reuse.

Generalisation, specialisation, and inheritance are interconnected.

Inheritance uses class generalisation to share properties and operations. In inheritance, generalisation and specialisation are two sides of the same coin. A superclass appears like a generalised version of a subclass, and vice versa.

Use inheritance instead of IS-A when one object acts like another. Subclasses can override superclass features by specifying them with the same name. Overriding features refine and replace superclass features.

Let's now examine the diagram shown in Figure 2. In this diagram, the Shape class is the parent of the Circle, Triangle, and Square classes. Since just one class is inherited from in this instance, it is a single inheritance case.

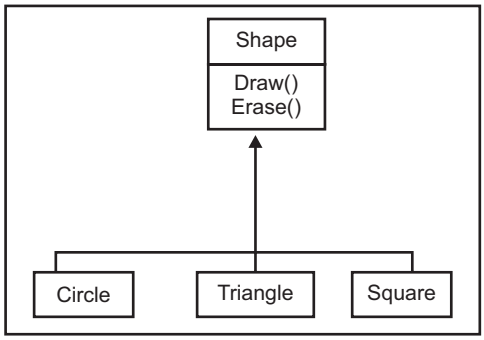


Fig. 2: Single inheritance

In figure 3, multiple inheritance is depicted. In this case, one class is descended from multiple classes.

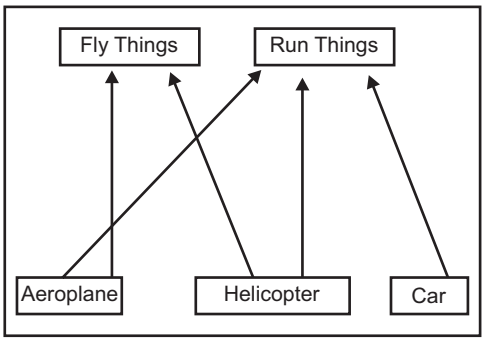


Fig. 3: Multiple inheritance

Polymorphism

Polymorphism and inheritance are linked. Polymorphism lets a subclass replace a superclass.

If many attributes are implemented, the class hierarchy decides. An object-oriented programme to calculate the area of different Figures would simply call the Find_ Area action on each figure, whether it is a circle, triangle, or something else. Based on its class, each object implicitly chooses a process.

Polymorphism simplifies maintenance by avoiding code changes when new classes are added. Polymorphism makes applications more versatile and changeable.

BASIC CONSTRUCTS IN OBJECT ORIENTATION

Objects and their attributes are described in object-oriented modeling. Objects are created in every system to serve a purpose. Some properties of object orientation are utilised to define the responsibilities of objects. We shall talk about these features, which include the following, in this section:

- Class and Objects,
- Links and Association, and
- Generalization and Inheritance.

Class and Objects

A class is a grouping of objects or ideas with similar properties. Each of these items or ideas is referred to as an object.

Classes specify the fundamental terms of the modelled system. It tends to substantially improve understanding and agreement about the definitions of words and other properties of the objects in the system to use a set of classes as the core vocabulary of a software project.

Data modeling can be built on top of classes. Let's examine how attributes and operations represent the traits that classes have in common.

Here is a list of these terms' definitions:

Slots for class-specific data values are referred to as **attributes**. The values of the attributes of the various objects in a particular class often differ to some extent.

Operations are services that an object can use to modify its own behaviour or the behaviour of the system as a whole.

We'll talk about the accepted class notation here. A class is represented as a box with three sections. According to figure 4, the class is divided into three sections: the top portion contains the class name in boldface, the middle section contains the class's characteristics, and the bottom section contains the class's actions.

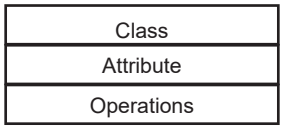


Fig. 4: Class notation

A class can be displayed without its actions or properties, or it can only have its name displayed, as in figure 5.

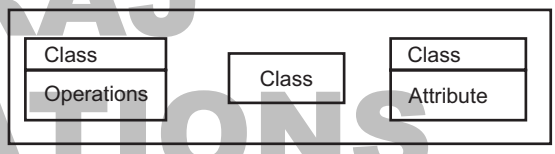


Fig. 5: Alternate class notations

The following is the class naming scheme:

- Simple nouns or noun phrases are used as class names.
- Simple nouns or noun phrases are used as attribute names in classes. The second and following words may be capitalised, but the first word is not.
- Operation names are straightforward verbs. Similar to characteristics, the first word is not capitalised, but further words may be.

Objects: The basic form of the notation for an object and a class is the same. The three distinctions between the notations are as follows:

- The name of the class to which the object belongs is displayed in the class box's upper part following a colon. Depending on whether the object is named—in which case the name appears before the colon—or anonymous, the colon will be followed by nothing.
- The upper compartment's contents are highlighted next to an item.
- Each attribute assigned to the specified class has a unique value for each object that falls under its purview.