Unit 1

OOAD (object oriented analysis and design)

Object-Oriented Analysis

Object–Oriented Analysis (OOA) is the procedure of identifying software engineering requirements and developing software specifications in terms of a software system's object model, which comprises of interacting objects.

The main difference between object-oriented analysis and other forms of analysis is that in object-oriented approach, requirements are organized around objects, which integrate both data and functions. They are modelled after real-world objects that the system interacts with. In traditional analysis methodologies, the two aspects - functions and data - are considered separately.

Grady Booch has defined OOA as, *"Object-oriented analysis is a method of analysis that examines requirements from the perspective of the classes and objects found in the vocabulary of the problem domain"*.

The primary tasks in object-oriented analysis (OOA) are:

- Identifying objects
- Organizing the objects by creating object model diagram
- Defining the internals of the objects, or object attributes
- Defining the behavior of the objects, i.e., object actions
- Describing how the objects interact

The common models used in OOA are use cases and object models.

Object-Oriented Design

Object–Oriented Design (OOD) involves implementation of the conceptual model produced during object-oriented analysis. In OOD, concepts in the analysis model, which are technology−independent, are mapped onto implementing classes, constraints are identified and interfaces are designed, resulting in a model for the solution domain, i.e., a detailed description of how the system is to be built on concrete technologies.

The implementation details generally include:

- Restructuring the class data (if necessary),
- Implementation of methods, i.e., internal data structures and algorithms,

- Implementation of control, and
- Implementation of associations.

Grady Booch has defined object-oriented design as "*a method of design encompassing the process of object-oriented decomposition and a notation for depicting both logical and physical as well as static and dynamic models of the system under design*".

Object-Oriented Programming

Object-oriented programming (OOP) is a programming paradigm based upon objects (having both data and methods) that aims to incorporate the advantages of modularity and reusability. Objects, which are usually instances of classes, are used to interact with one another to design applications and computer programs.

The important features of object–oriented programming are:

- Bottom–up approach in program design
- Programs organized around objects, grouped in classes
- Focus on data with methods to operate upon object's data
- Interaction between objects through functions
- Reusability of design through creation of new classes by adding features to existing classes

Some examples of object-oriented programming languages are C++, Java,C# etc.

➢ Objects and Classes

The concepts of objects and classes are intrinsically linked with each other and form the foundation of object–oriented paradigm.

Object

An object is a real-world element in an object–oriented environment that may have a physical or a conceptual existence. Each object has:

- Identity that distinguishes it from other objects in the system.
- State that determines the characteristic properties of an object as well as the values of the properties that the object holds.
- Behavior that represents externally visible activities performed by an object in terms of changes in its state.

Objects can be modelled according to the needs of the application. An object may have a physical existence, like a customer, a car, etc.; or an intangible conceptual existence, like a project, a process, etc.

Class

A class represents a collection of objects having same characteristic properties that exhibit common behavior. It gives the blueprint or description of the objects that can be created from it. Creation of an object as a member of a class is called instantiation. Thus, object is an instance of a class.

The constituents of a class are:

- A set of attributes for the objects that are to be instantiated from the class. Generally, different objects of a class have some difference in the values of the attributes. Attributes are often referred as class data.
- A set of operations that portray the behavior of the objects of the class. Operations are also referred as functions or methods.

Example

Let us consider a simple class, Circle, that represents the geometrical figure circle in a two–dimensional space. The attributes of this class can be identified as follows:

- x–coord, to denote x–coordinate of the center
- y–coord, to denote y–coordinate of the center
- a, to denote the radius of the circle

Some of its operations can be defined as follows:

- findArea(), method to calculate area
- findCircumference(), method to calculate circumference
- scale(), method to increase or decrease the radius

During instantiation, values are assigned for at least some of the attributes. If we create an object my_circle, we can assign values like x-coord : 2, y-coord : 3, and a : 4 to depict its state. Now, if the operation scale() is performed on my_circle with a scaling factor of 2, the

value of the variable a will become 8. This operation brings a change in the state of my_circle, i.e., the object has exhibited certain behavior.

Encapsulation and Data Hiding

Encapsulation

Encapsulation is the process of binding both attributes and methods together within a class. Through encapsulation, the internal details of a class can be hidden from outside. It permits the elements of the class to be accessed from outside only through the interface provided by the class.

Data Hiding

Typically, a class is designed such that its data (attributes) can be accessed only by its class methods and insulated from direct outside access. This process of insulating an object's data is called data hiding or information hiding.

Example :In the class Circle, data hiding can be incorporated by making attributes invisible from outside the class and adding two more methods to the class for accessing class data, namely:

- setValues(), method to assign values to x-coord, y-coord, and a
- getValues(), method to retrieve values of x-coord, y-coord, and a

Here the private data of the object my_circle cannot be accessed directly by any method that is not encapsulated within the class Circle. It should instead be accessed through the methods setValues() and getValues().

Message Passing

Any application requires a number of objects interacting in a harmonious manner. Objects in a system may communicate with each other using message passing. Suppose a system has two objects: obj1 and obj2. The object obj1 sends a message to object obj2, if obj1 wants obj2 to execute one of its methods.

The features of message passing are:

- Message passing between two objects is generally unidirectional.
- Message passing enables all interactions between objects.
- Message passing essentially involves invoking class methods.

- Objects in different processes can be involved in message passing.

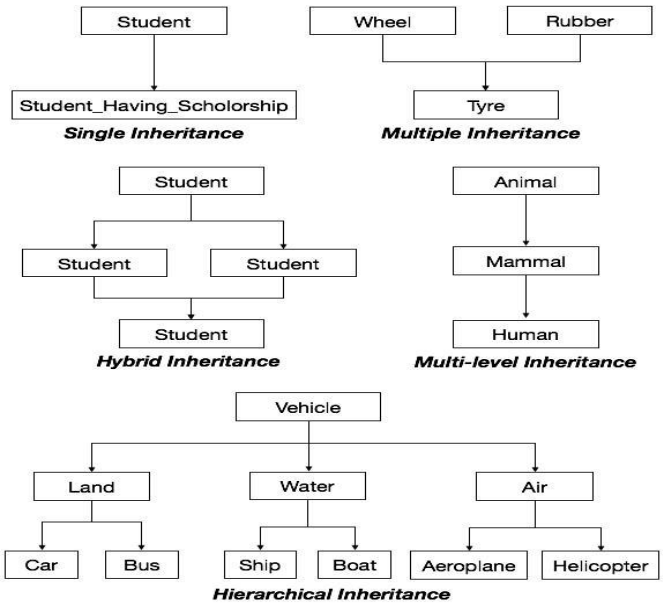
Inheritance

Inheritance is the mechanism that permits new classes to be created out of existing classes by extending and refining its capabilities. The existing classes are called the base classes/parent classes/super-classes, and the new classes are called the derived classes/child classes/subclasses. The subclass can inherit or derive the attributes and methods of the super-class(es) provided that the super-class allows so. Besides, the subclass may add its own attributes and methods and may modify any of the super-class methods. Inheritance defines an "is – a" relationship.

Types of Inheritance:

- Single Inheritance : A subclass derives from a single super-class.
- Multiple Inheritance : A subclass derives from more than one super-classes.
- Multilevel Inheritance : A subclass derives from a super-class which in turn is derived from another class and so on.
- Hierarchical Inheritance : A class has a number of subclasses each of which may have subsequent subclasses, continuing for a number of levels, so as to form a tree structure.
- Hybrid Inheritance : A combination of multiple and multilevel inheritance so as to form a lattice structure.

The following figure depicts the examples of different types of inheritance.



Polymorphism

Polymorphism is originally a Greek word that means the ability to take multiple forms. In object-oriented paradigm, polymorphism implies using operations in different ways, depending upon the instance they are operating upon. Polymorphism allows objects with different internal structures to have a common external interface. Polymorphism is particularly effective while implementing inheritance.

Example :Let us consider two classes, Circle and Square, each with a method findArea(). Though the name and purpose of the methods in the classes are same, the internal implementation, i.e., the procedure of calculating area is different for each class. When an object of class Circle invokes its findArea() method, the operation finds the area of the circle without any conflict with the findArea() method of the Square class.

Advantages of Object-Oriented Analysis and Design

- It is easy to understand.
- It is easy to maintain. Due to its maintainability OOAD is becoming more popular day by day
- It provides re-usability
- It reduce the development time & cost
- It improves the quality of the system due to program reuse

➢ Object-Oriented Modeling (OOM)

Object-oriented modeling (OOM) is the construction of objects using a collection of objects that contain stored values of the instance variables found within an object. Unlike models that are record-oriented, object-oriented values are solely objects.

The object-oriented modeling approach creates the union of the application and database development and transforms it into a unified data model and language environment. Object-oriented modeling allows for object identification and communication while supporting data abstraction, inheritance and encapsulation.

Object-oriented modeling is the process of preparing and designing what the model's code will actually look like. During the construction or programming phase, the modeling techniques are implemented by using a language that supports the object-oriented programming model.

OOM consists of progressively developing object representation through three phases: analysis, design, and implementation. During the initial stages of development, the model developed is abstract because the external details of the system are the central focus. The model becomes more and more detailed as it evolves, while the central focus shifts toward understanding how the system will be constructed and how it should function.

➢ Object Oriented Testing

Software typically undergoes many levels of testing, from unit testing to system or acceptance testing. Typically, in-unit testing, small "units", or modules of the software, are tested separately with focus on testing the code of that module. In higher, order testing (e.g, acceptance testing), the entire system (or a subsystem) is tested with the focus on testing the functionality or external behavior of the system.

Testing classes is a fundamentally different problem than testing functions. A function (or a procedure) has a clearly defined input-output behavior, while a class does not have an input-output behavior specification. We can test a method of a class using approaches for testing functions, but we cannot test the class using these approaches.

Techniques of object-oriented testing are as follows:

1. **Fault Based Testing:**
   This type of checking permits for coming up with test cases supported the consumer specification or the code or both. It tries to identify possible faults (areas of design or code that may lead to errors.). For all of these faults, a test case is developed to "flush" the errors out. These tests also force each time of code to be executed.
   This method of testing does not find all types of errors. However, incorrect specification and interface errors can be missed. These types of errors can be uncovered by function testing in the traditional testing model. In the object-oriented model, interaction errors can be uncovered by scenario-based testing. This form of Object oriented-testing can only test against the client's specifications, so interface errors are still missed.
2. **Class Testing Based on Method Testing:**
   This approach is the simplest approach to test classes. Each method of the class performs a well defined cohesive function and can, therefore, be related to unit testing of the traditional testing techniques. Therefore all the methods of a class can be involved at least once to test the class.
3. **Random Testing:**
   It is supported by developing a random test sequence that tries the minimum variety of operations typical to the behavior of the categories
4. **Partition Testing:**
   This methodology categorizes the inputs and outputs of a category so as to check them severely. This minimizes the number of cases that have to be designed.
5. **Scenario-based Testing:**
   It primarily involves capturing the user actions then stimulating them to similar actions throughout the test.
   These tests tend to search out interaction form of error.

Object Oriented Testing methods:

Testing is a continuous activity during software development. In object-oriented systems, testing encompasses three levels, namely, unit testing, subsystem testing, and system testing.

Unit Testing:

- In unit testing, the individual classes are tested. It is seen whether the class attributes are implemented as per design and whether the methods and the interfaces are error-free.
- Unit testing is the responsibility of the application engineer who implements the structure.

Subsystem Testing:

- This involves testing a particular module or a subsystem and is the responsibility of the subsystem lead. It involves testing the associations within the subsystem as well as the interaction of the subsystem with the outside.
- Subsystem tests can be used as regression tests for each newly released version of the subsystem.

System Testing:

- System testing involves testing the system as a whole and is the responsibility of the quality-assurance team. The team often uses system tests as regression tests when assembling new releases.

Object-Oriented Testing Techniques:

Grey Box Testing:

The different types of test cases that can be designed for testing object-oriented programs are called grey box test cases. Some of the important types of grey box testing are:

- **State model based testing:** This encompasses state coverage, state transition coverage, and state transition path coverage.
- **Use case based testing:** Each scenario in each use case is tested.
- **Class diagram based testing:** Each class, derived class, associations, and aggregations are tested.
- **Sequence diagram based testing:** The methods in the messages in the sequence diagrams are tested.

Techniques for Subsystem Testing:

The two main approaches of subsystem testing are:

- **Thread based testing:** All classes that are needed to realize a single use case in a subsystem are integrated and tested.
- **Use based testing:** The interfaces and services of the modules at each level of hierarchy are tested. Testing starts from the individual classes to the small modules comprising of classes, gradually to larger modules, and finally all the major subsystems.

Categories of System Testing:

- **Alpha testing:** This is carried out by the testing team within the organization that develops software.
- **Beta testing:** This is carried out by select group of co-operating customers.
- **Acceptance testing:** This is carried out by the customer before accepting the deliverables.

Black-box testing:
Testing that verifies the item being tested when given the appropriate input provides the expected results.

Boundary-value testing:

Testing of unusual or extreme situations that an item should be able to handle.

Class testing:
The act of ensuring that a class and its instances (objects) perform as defined.

Component testing:
The act of validating that a component works as defined.

Inheritance-regression testing:

The act of running the test cases of the super classes, both direct and indirect, on a given subclass.

Integration testing:
Testing to verify several portions of software work together.

Model review:
An inspection, ranging anywhere from a formal technical review to an informal walkthrough, by others who were not directly involved with the development of the model.

Path testing:
The act of ensuring that all logic paths within your code are exercised at least once.

Regression testing:

The acts of ensuring that previously tested behaviors still work as expected after changes have been made to an application.

Stress testing:

The act of ensuring that the system performs as expected under high volumes of transactions, users, load, and so on.

Technical review:

A quality assurance technique in which the design of your application is examined critically by a group of your peers. A review typically focuses on accuracy, quality, usability, and completeness. This process is often referred to as a walkthrough, an inspection, or a peer review.

User interface testing:

The testing of the user interface (UI) to ensure that it follows accepted UI standards and meets the requirements defined for it. Often referred to as graphical user interface (GUI) testing.

White-box testing:
Testing to verify that specific lines of code work as defined. Also referred to as clear-box testing.