BACHELOR OF TECHNOLOGY YEAR 3

OBJECT ORIENTED ANALYSIS AND DESIGN

WEEK 1 & 2:

OBJECT ORIENTED PARADIGM

It is a programming paradigm based on the concept of "*objects*", which may contain data, in the form of fields, often known as *attributes*; and code, in the form of procedures, often known as *methods*. It entails the following:-

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 modeled after real-world objects that the system interacts with.

In traditional analysis methodologies, the two aspects - *functions* and *data* - are considered separately.

DEFINITION

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 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.

DEFINITION

Grady Booch has defined object-oriented design as *"a method of design which entails the process of object-oriented decomposition and a notation to represent logical and physical as well as static and dynamic models of the system to be designed"*.

OBJECT-ORIENTED PROGRAMMING

Object-oriented programming (OOP) is a programming paradigm based on *objects* (having both data and methods) that aims to integrate 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#,

DEFINITION

Grady Booch has defined object–oriented programming as *"a method of implementation in which programs are organized as collections of objects, each of which represents an instance of some class, and whose classes are all members of a hierarchy of classes united through inheritance relationships"*.

THE OBJECT MODEL

The Object Model is based on a few key concepts that enable us to view the world around us. The next few sections discuss these key concepts.

1. OBJECTS AND CLASSES

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

Definition

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.

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 components 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, which is a geometrical figure. The attributes of this class can be identified as follows:

- r, to denote the radius of the circle
- d, to denote the diameter of the circle

Some of its operations can be defined as follows:

- findArea(), method to calculate area
- findCircumference(), method to calculate circumference

During instantiation, values are assigned for at least some of the attributes. If we create an object my_circle, we can assign values like r : 7

The object-oriented paradigm views the world as a collection of *unique objects,* often referred to as a society of objects. Each object has a lifecycle where *it knows something*, can *do something*, and *can communicate* with other objects. What an object knows and can do are known as features. For example, a manager

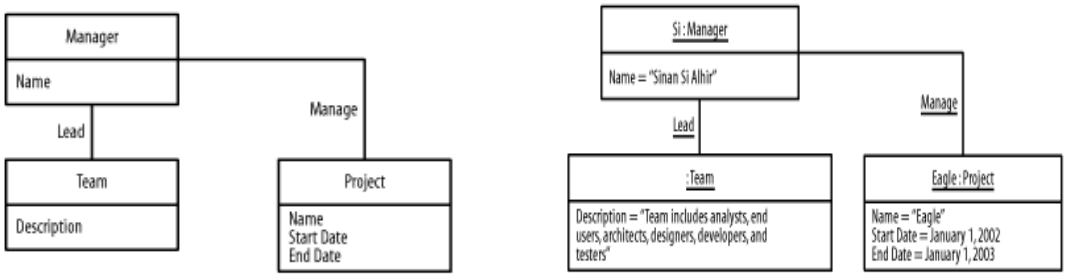
- knows his or her name,
- can initiate or terminate a project,
- can communicate with a team to lead the team to successfully execute the project

Features belong to two broad categories or types: *attributes* and *operations*.

a) Attributes

Something that an object knows is called an *attribute*, which essentially represents data. A class defines *attributes* and an *object* has values for those attributes. Even if two objects have the same values for their attributes, the objects are unique and have their own identities.

In a UML diagram, a class may be shown with a second compartment that lists these attributes as text strings. Likewise, an object may be shown with a second compartment that lists these attributes as text strings, each followed by an equal symbol (=) and its value as shown below.

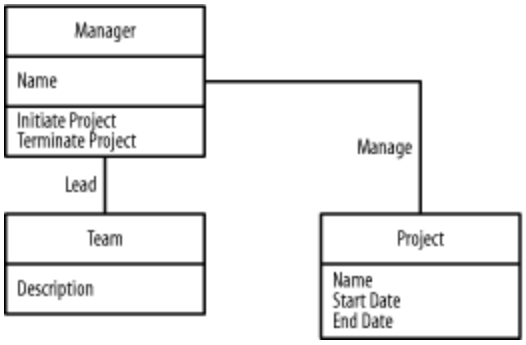


b) Operations and methods

Something an object can do is called an *operation* and essentially represents processing. How an object does the processing for a given operation is known as the *operation's method* or *implementation*.

For example, when using a programming language, we declare functions or procedures and define their bodies (lines of code) that determine what the functions or procedures do when they are invoked and executed; a function's declaration is the operation, and the body definition is the method.

A class defines *operations* and *methods* that apply to its objects. A class may be shown with a third compartment that lists these operations as text strings as shown below.



2. 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 radius, diameter
- getValues(), method to retrieve values of radius, radius

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

3. 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.

4. 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/*sub*classes.

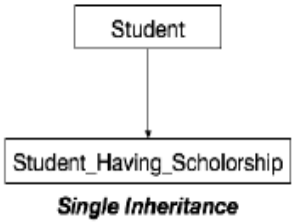
The subclass can *inherit* or *derive* the attributes and methods of the super-class (es) provided that the super-class allows so. Besides, the subclass may add its own attributes and methods and may modify any of the super-class methods. Inheritance defines an "*is – a*" relationship.

Example

From a class Mammal, a number of classes can be derived such as Human, Cat, Dog, Cow, etc. Humans, cats, dogs, and cows all have the distinct characteristics of mammals. In addition, each has its own particular characteristics. It can be said that a cow "is – a" mammal.

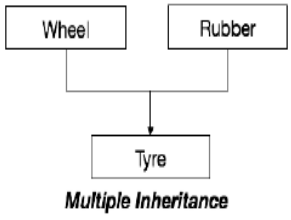
Types of Inheritance

- *Single Inheritance:* A subclass derives from a single super-class.



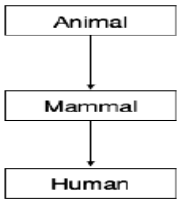
Single Inheritance

- *Multiple Inheritance*: A subclass derives from more than one super-classes.



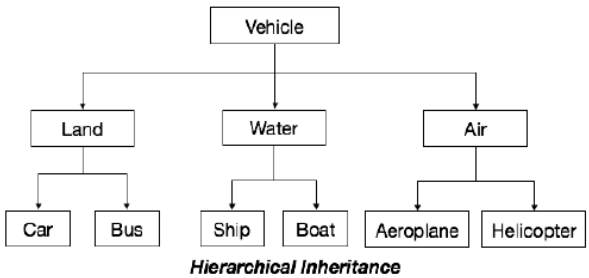
Multiple Inheritance

- *Multilevel Inheritance:* A subclass derives from a super-class which in turn is derived from another class and so on.



Multi-level Inheritance

- *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.



Hierarchical Inheritance

5. 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.

6. GENERALIZATION AND SPECIALIZATION

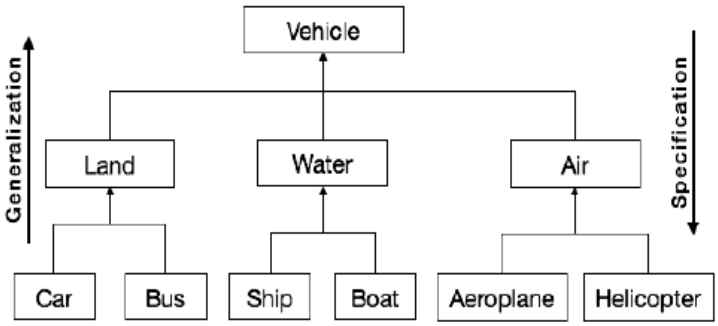
Generalization and specialization represent a hierarchy of relationships between classes, where subclasses inherit from super-classes.

Generalization

In generalization the common characteristics of classes are combined to form a class in a higher level of hierarchy, i.e., subclasses are combined to form a generalized super-class. It represents an *"is – a – kind – of"* relationship. For example, "car is a kind of land vehicle", or "ship is a kind of water vehicle".

Specialization

Specialization is the *reverse process* of generalization. The distinguishing features of groups of objects are used to form specialized classes from existing classes. It can be said that the subclasses are the specialized versions of the super-class. The following figure shows an example of generalization and specialization.



7. Links and Association

A *link* represents a connection through which an object *collaborates* with other objects. It is defined as "a physical or conceptual connection between objects". Through a link, one object may invoke the methods or navigate through another object. A link depicts the relationship between two or more objects.

Association is a group of links having common structure and common behavior. Association depicts the *relationship* between objects of one or more classes. A *link* can be defined as an instance of an *association*.

Degree of an Association

Degree of an association denotes the number of classes involved in a connection. Degree may be unary, binary, or ternary.

- A *unary relationship* connects objects of the same class.
- A *binary relationship* connects objects of two classes.
- A *ternary relationship* connects objects of three or more classes.

Cardinality Ratios of Associations

Cardinality of a binary association denotes the number of instances participating in an association. There are three types of cardinality ratios, namely:

- *One–to–One:* A single object of class A is associated with a single object of class B.
- *One–to–Many:* A single object of class A is associated with many objects of class B.
- *Many–to–Many:* An object of class A may be associated with many objects of class B and conversely an object of class B may be associated with many objects of class A.

8. Aggregation or Composition

Aggregation or *composition* is a relationship among classes by which a class can be made up of any combination of objects of other classes. It allows objects to be placed directly within the body of other classes. Aggregation is referred as a "*part–of*" or "*has–a*" relationship, with the ability to navigate from the whole to its parts. An aggregate object is an object that is composed of one or more other objects.

Example

In the relationship, "*a car has–a motor*", car is the whole object or the aggregate, and the *motor* is a "*part–of*" the car. Aggregation may denote:

- *Physical containment:* Example, a computer is composed of monitor, CPU, mouse, keyboard, and so on.
- *Conceptual containment:* Example, shareholder has–a share.

BENEFITS OF OBJECT MODEL

Now that we have gone through the core concepts pertaining to object orientation, it would be worthwhile to note the advantages that this model has to offer.

The benefits of using the object model are:-

- It helps in faster development of software.
- It is easy to maintain. Suppose a module develops an error, then a programmer can fix that particular module, while the other parts of the software are still up and running.
- It supports relatively hassle-free upgrades.
- It enables reuse of objects, designs, and functions.
- It reduces development risks, particularly in integration of complex systems.

OBJECT-ORIENTED PRINCIPLES

The conceptual framework of object–oriented systems is based upon the object model. There are two categories of elements in an object-oriented system are: -

a) ***Major Elements:*** By major, it is meant that if a model does not have any one of these elements, it ceases to be object oriented. The four major elements are:

i) Abstraction

Abstraction means to focus on the essential features of an element or object in OOP, ignoring its extraneous or accidental properties. The essential features are relative to the context in which the object is being used.

Definition

"*An abstraction denotes the essential characteristics of an object that distinguish it from all other kinds of objects and thus provide crisply defined conceptual boundaries, relative to the perspective of the viewer*."

Example: When a class Student is designed, the attributes enrolment_number, name, course, and address are included while characteristics like height and blood group are eliminated, since they are irrelevant in the perspective of the educational institution.

ii) 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. The class has methods that provide user interfaces by which the services provided by the class may be used.

iii) Modularity

Modularity is the process of decomposing a problem (program) into a set of modules so as to reduce the overall complexity of the problem.

Definition

"*Modularity is the property of a system that has been decomposed into a set of cohesive and loosely coupled modules*."

Modularity is linked with encapsulation. Modularity can be visualized as a way of mapping encapsulated abstractions into real, physical modules having high interconnection within the modules and their inter–module interaction or connection is low.

iv) Hierarchy

"*Hierarchy is the ranking or ordering of abstraction*". Through hierarchy, a system can be made up of interrelated subsystems, which can have their own subsystems and so on until the smallest level components are reached. Hierarchy achieves *reusability*.

Types of hierarchies in OOA are:

- *"IS–A" hierarchy:* It defines the hierarchical relationship in inheritance, whereby from a super-class, a number of subclasses may be derived which may again have subclasses and so on. For example, if we derive a class Rose from a class Flower, we can say that a rose "is–a" flower.
- *"PART–OF" hierarchy:* It defines the hierarchical relationship in aggregation by which a class may be composed of other classes. For example, a flower is composed of sepals, petals, stamens, and carpel. It can be said that a petal is a "part–of" flower.

b) *Minor Elements:* By minor, it is meant that these elements are useful, but not indispensable part of the object model. The three minor elements are:

i) Typing

According to the theories of abstract data type, *a type is a characterization of a set of elements.* In OOP, a class is visualized as a *type* having properties distinct from any other types. *Typing* is the enforcement of the notion that an object is an instance of a single class or type. It also enforces that objects of different types may not be generally interchanged; and can be interchanged only in a very restricted manner if absolutely required to do so.

Two types of typing

Strong Typing: Here, the operation on an object is checked at the time of compilation,

Weak Typing: Here, messages may be sent to any class. The operation is checked only at the time of execution

ii) Concurrency

Concurrency in operating systems allows performing multiple tasks or processes simultaneously. When a single process exists in a system, it is said that there is a single thread of control. However, most systems have multiple threads, some active, some waiting for CPU, some suspended, and some terminated. Systems

with multiple CPUs inherently permit concurrent threads of control; but systems running on a single CPU use appropriate algorithms to give equitable CPU time to the threads so as to enable concurrency.

In an object-oriented environment, there are active and inactive objects. The active objects have independent threads of control that can execute concurrently with threads of other objects. The active objects synchronize with one another as well as with purely sequential objects.

iii) Persistence

This property by which an object continues to exist even after its creator ceases to exist. An object occupies a memory space and exists for a particular period of time. In traditional programming, the lifespan of an object was typically the lifespan of the execution of the program that created it. In files or databases, the object lifespan is longer than the duration of the process creating the object.