

MODULE-1

INTRODUCTION

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().

Abstraction

Abstraction is the process of taking away or removing characteristics from something in order to reduce it to a set of essential characteristics. In object-oriented programming, abstraction is one of three central principles (along with encapsulation and inheritance). Through the process of abstraction, a programmer hides all but the relevant data about an object in order to reduce complexity and increase efficiency. In the same way that abstraction sometimes works in art, the object that remains is a representation of the original, with unwanted detail omitted. The resulting object itself can be referred to as an abstraction,

meaning a *named entity* made up of selected attributes and behavior specific to a particular usage of the originating entity. Abstraction is related to both encapsulation and data hiding.

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.

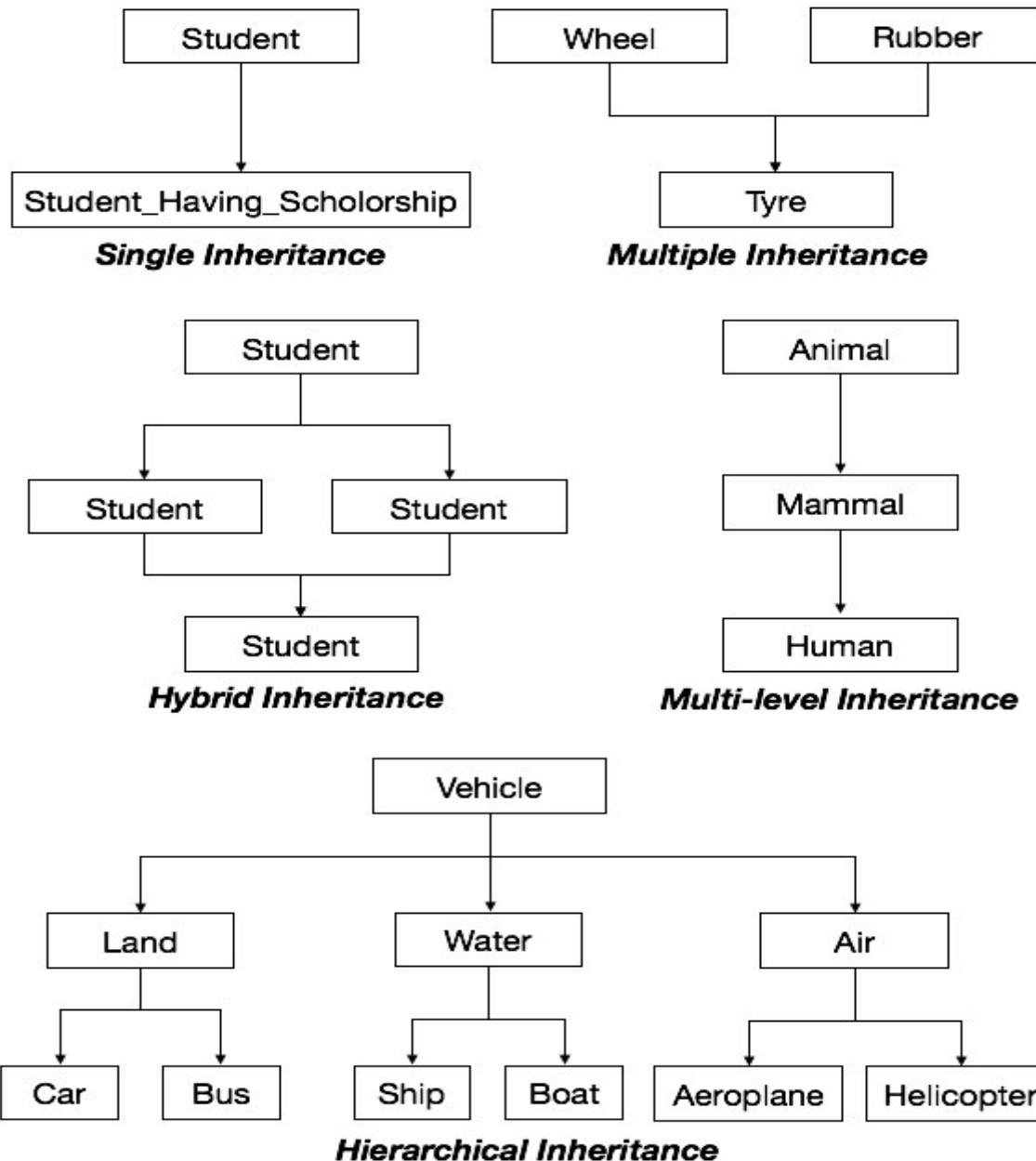
Example

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.
- **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.



Dynamic binding

Dynamic binding also called **dynamic dispatch** is the process of linking procedure call to a specific sequence of code (method) at run-time. It means that the code to be executed for a specific procedure call is not known until run-time. Dynamic binding is also known as *late binding* or *run-time binding*.

Dynamic binding is an object oriented programming concept and it is related with polymorphism and inheritance.

Dynamic binding definition

Dynamic binding(dispatch) means that a block of code executed with reference to a procedure(method) call is determined at run time.

Dynamic dispatch is generally used when multiple classes contain different implementations of the same method. It provides a mechanism for selecting the function to be executed from various function alternatives at the run-time.

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.

Generalization and Specialization

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

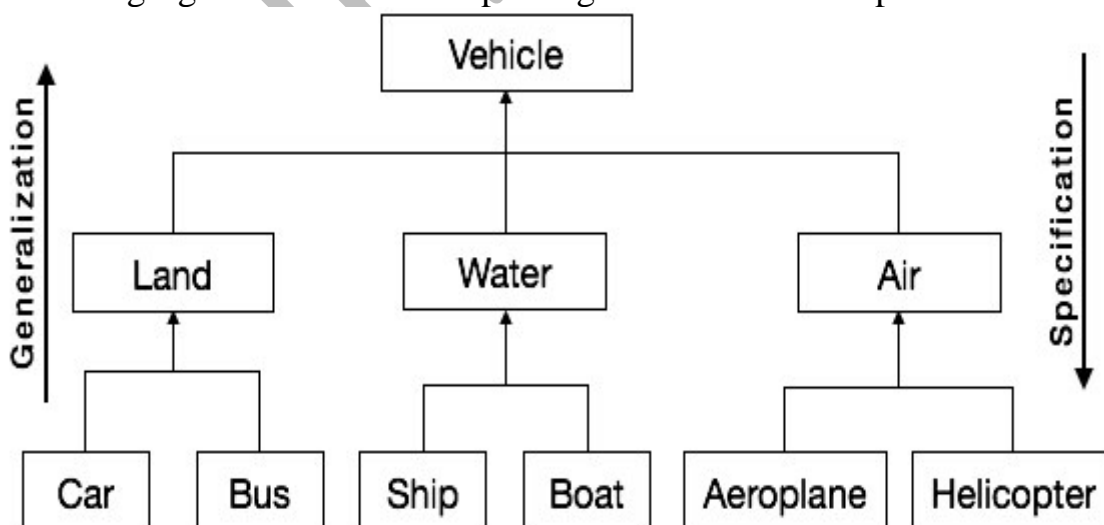
Generalization

In the generalization process, 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. Here, 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.



Overview of OOAD

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”*.