Object Oriented System Design

(KCS-054)

Unit 1

INDEX

- Introduction:
- The meaning of Object Orientation,
- object identity
- Encapsulation
- information hiding
- Polymorphism
- Generosity
- Importance of modelling
- Principles of modelling
- Object oriented modelling

Object Oriented System Design
Course Outcome (CO)

- CO 1 Understand the application development and analyze the insights of object oriented programming to implement application K2, K4
- CO 2 Understand, analyze and apply the role of overall modeling concepts (i.e. System, structural) K2, K3
- CO 3 Understand, analyze and apply oops concepts (i.e. abstraction, inheritance) K2, K3, K4
- CO 4 Understand the basic concepts of C++ to implement the object oriented concepts K2, K3
- CO 5 To understand the object oriented approach to implement real world problem. K2, K3

A Brief History

The object-oriented paradigm took its shape from the initial concept of a new programming approach, while the interest in design and analysis methods came much later.

- The first object-oriented language was Simula (Simulation of real systems) that was developed in 1960 by researchers at the Norwegian Computing Center.
- In 1970, Alan Kay and his research group at Xerox PARK created a personal computer named Dynabook and the first pure object-oriented programming language (OOPL) - Smalltalk, for programming the Dynabook.

- In the 1980s, Grady Booch published a paper titled Object Oriented Design that mainly presented a design for the programming language, Ada. In the ensuing editions, he extended his ideas to a complete object-oriented design method.
- In the 1990s, Coad incorporated behavioral ideas to object-oriented methods.

Object-Oriented Analysis

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".

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

- 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 Design

The implementation details generally include –

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

Object-Oriented Programming

- Grady Booch has defined object-oriented programming as

- *“a method of implementation in which programs are organized as cooperative collections of objects, each of which represents an instance of some class, and whose classes are all members of a hierarchy of classes united via inheritance relationships”.*

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

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.

Some examples of object-oriented programming languages are

- C++
- Java
- Smalltalk,
- Delphi,
- C#,
- Perl,
- Python,
- Ruby,
- PHP.

Object Model

- The object model visualizes the elements in a software application in terms of objects. In this, we will look into the basic concepts and terminologies of object-oriented systems.

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.

How to Create Class in C++

```
class MyClass {         // The class
  public:               // Access specifier
    int myNum;          // Attribute (int variable)
    string myString;    // Attribute (string variable)
};
```

How to Create Object in C++

```
int main() {
  MyClass myObj;  // Create an object of MyClass

  // Access attributes and set values
  myObj.myNum = 15;
  myObj.myString = "Some text";

  // Print attribute values
  cout << myObj.myNum << "\n";
  cout << myObj.myString;
  return 0;
}
```

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

- 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.
- getValues(), method to retrieve values of x-coord, y-coord.

- 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

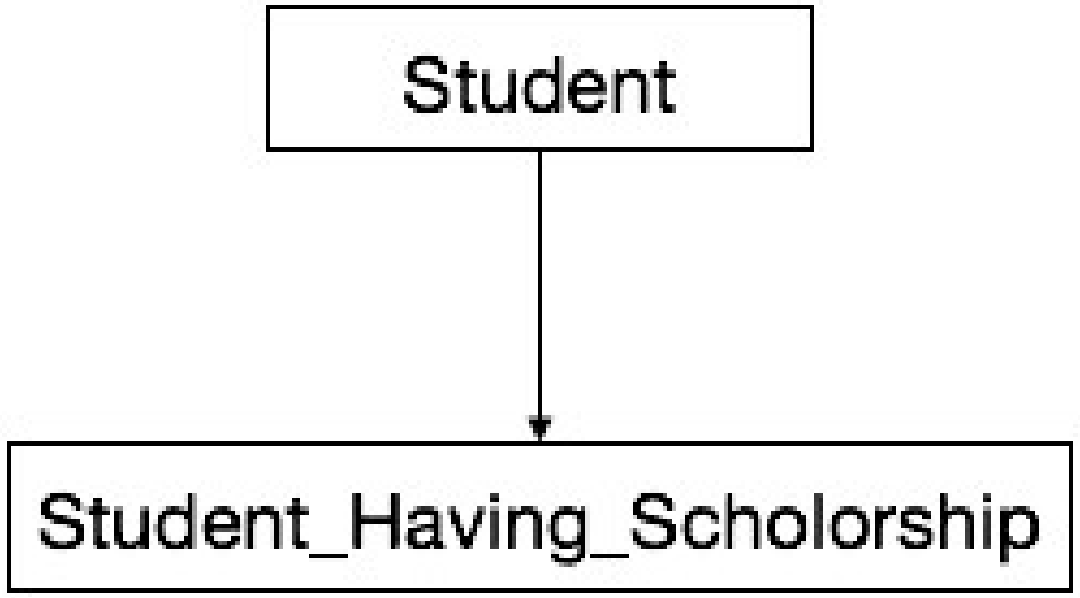
- Inheritance is the mechanism that permits new classes to be created out of existing classes by extending and refining its capabilities.
- The existing classes are called the base classes/parent classes/super-classes, and the new classes are called the derived classes/child classes/subclasses.
- The 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.

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.

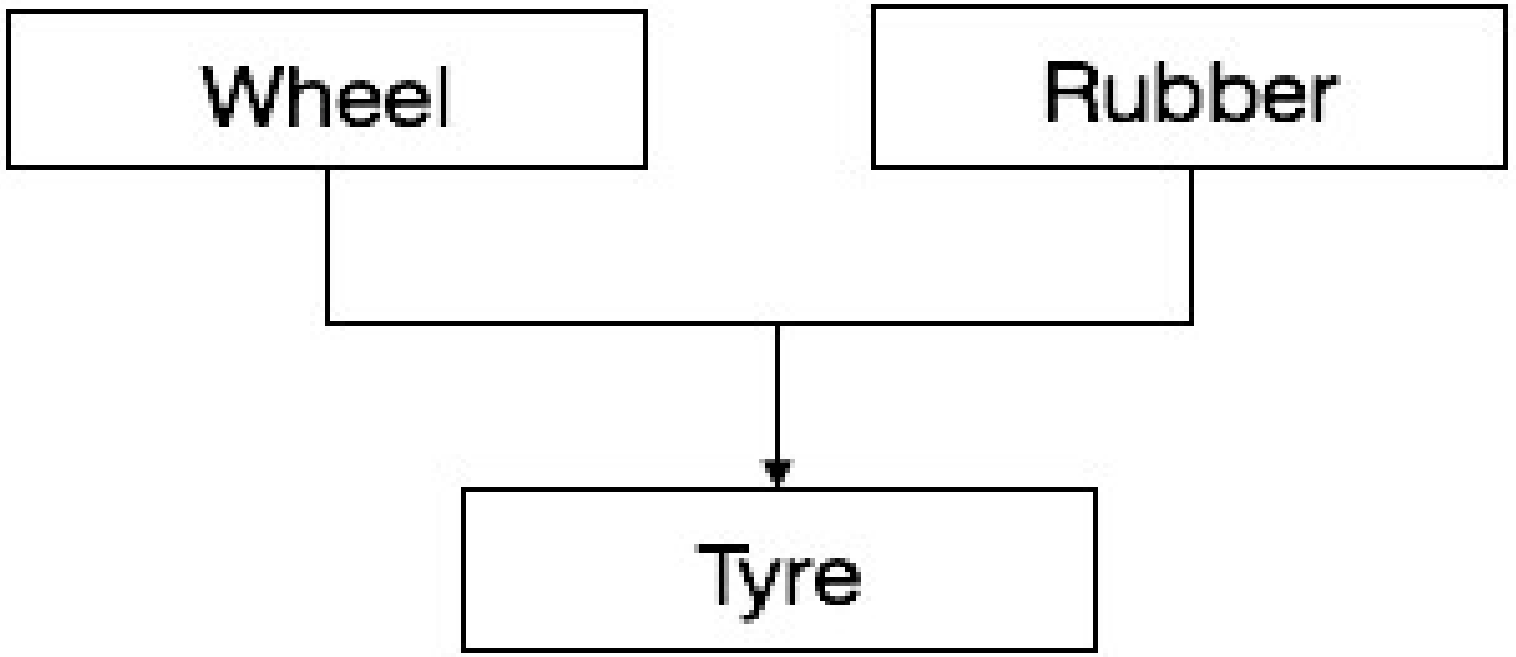
Types of Inheritance

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



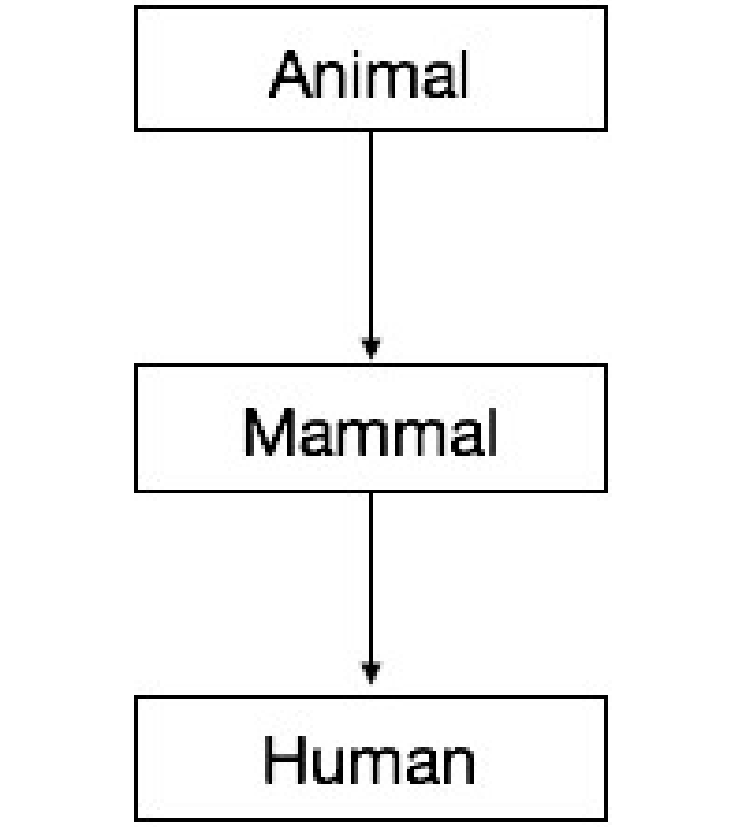
Single Inheritance

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



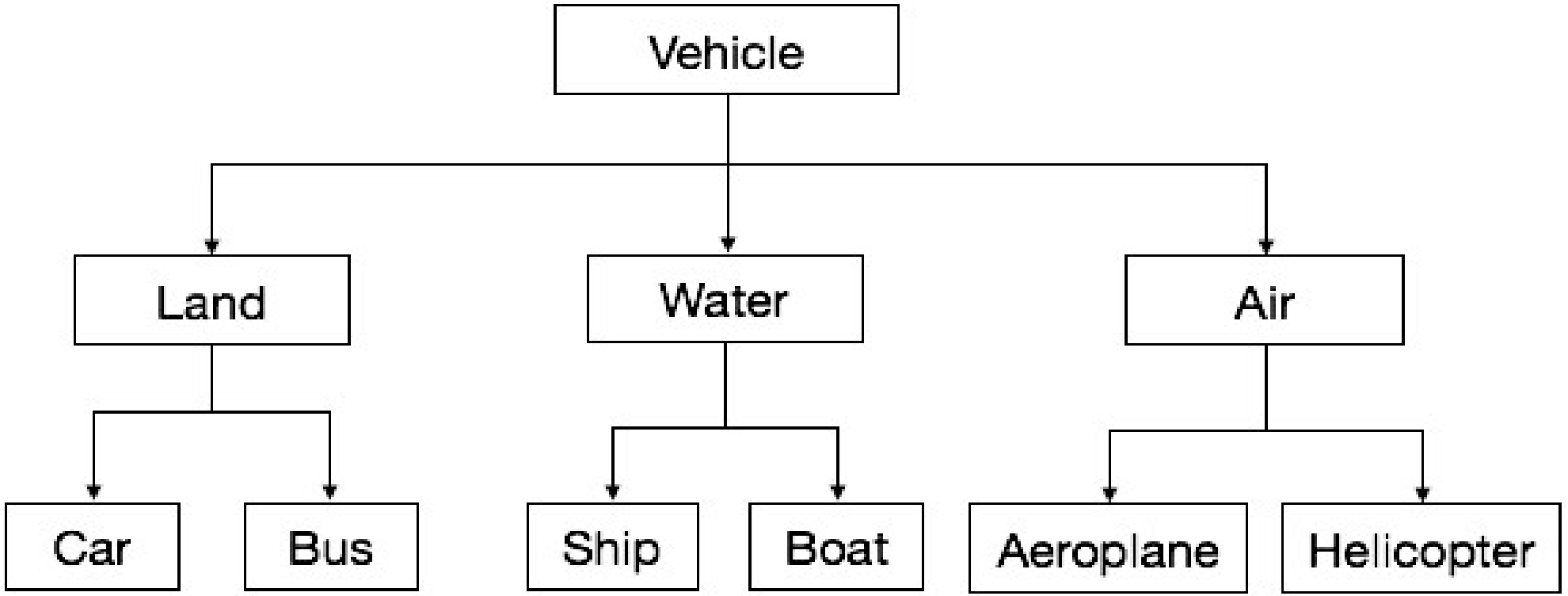
Multiple Inheritance

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



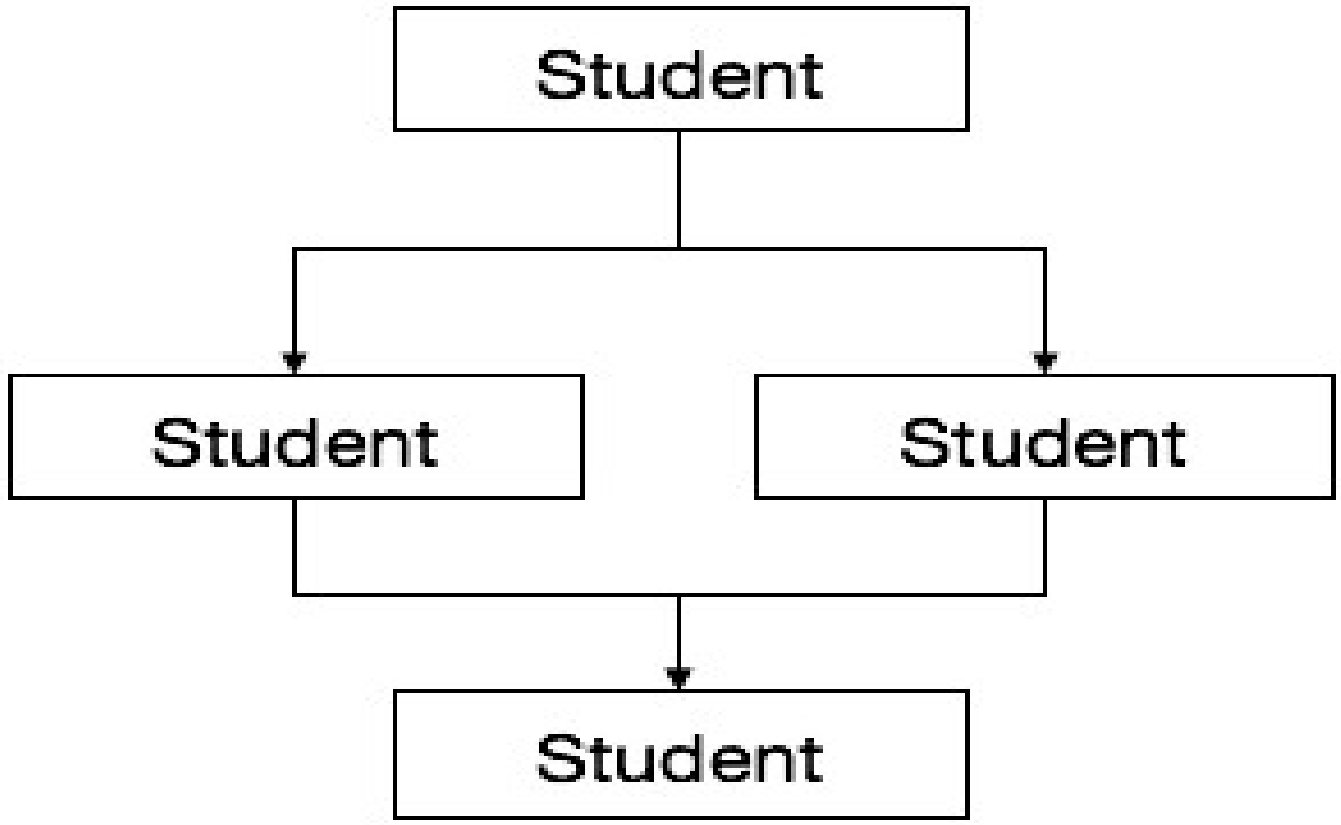
Multi-level Inheritance

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

Hybrid Inheritance – A combination of multiple and multilevel inheritance so as to form a lattice structure.



Hybrid Inheritance

Polymorphism

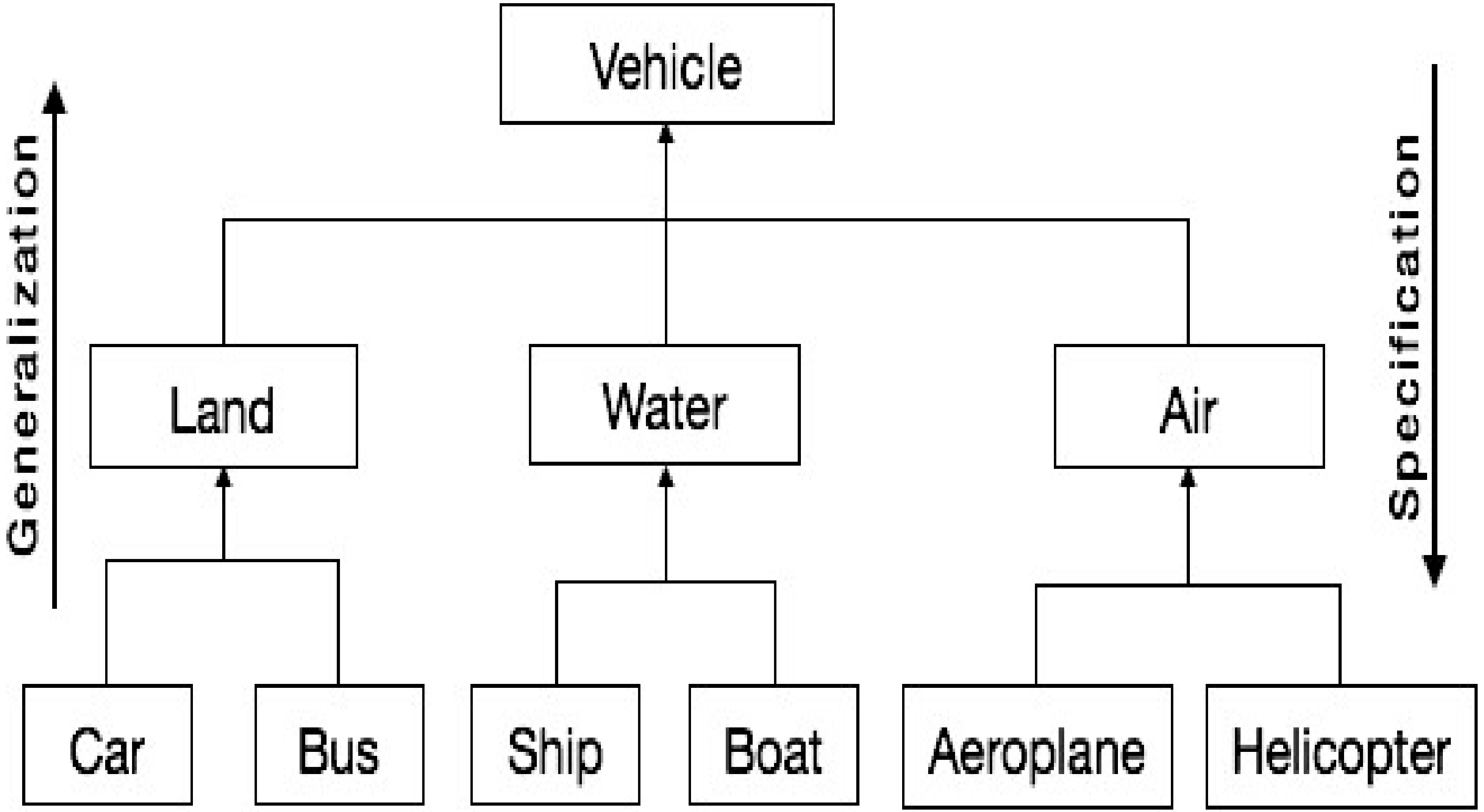
- Polymorphism is originally a Greek word that means the ability to take multiple forms.
- In object-oriented paradigm, polymorphism implies using operations in different ways, depending upon the instance they are operating upon.
- Polymorphism allows objects with different internal structures to have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

Generalization and Specialization

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


Vehicle
Generalization
Specification
Land
Water
Air
Car
Bus
Ship
Boat
Aeroplane
Helicopter

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.

Links and Association

Link

- A link represents a connection through which an object collaborates with other objects.
- Rumbaugh has defined it 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

- 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

- **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.

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.

Importance of Modeling

Through modeling, we achieve four aims:

- Models help us to visualize a system as it is or as we want it to be.
- Models permit us to specify the structure or behavior of a system.
- Models gives us a template that guides us in constructing a system.
- Models document the decisions we have made.

Principles of Modeling

- Modeling has rich history in all the engineering disciplines.
- That experience suggests four basic principles of modeling.

First principle of modeling

- The choice of what models to create has a profound influence on how a problem is attacked and how a solution is shaped.

- The right models will highlight the most nasty development problems.
- Wrong models will mislead you, causing you to focus on irrelevant issues.

Second principle of modeling

Every model may be expressed at different levels of precision.

Degree of detail, depending on who is viewing it.

- An analyst or end user - issues of what
- A developer - issues of how.

Third principle of modeling

- The best models are connected to reality.

- In software, the gap between the analysis model and the system's design model must be less. Failing to bridge this gap causes the system to diverge over time. In object-oriented systems, it is possible to connect all the nearly independent views of a system into one whole.

Fourth principle of modeling

- No single model is sufficient.

- Ex: In the case of a building, you can study electrical plans in isolation, but you can also see their mapping to the floor plan and perhaps even their interaction with the routing of pipes in the plumbing plan.