Programming Paradigms: Procedural, Structural, and Object-Oriented

Programming paradigms are different ways of designing and writing programs. They define how you approach a problem and how the solution is implemented through code.

Some of the programming paradigm

- **Procedural Programming**: This is one of the earliest programming paradigms. It's all about breaking down a problem into a series of procedures or steps (also called functions or routines). Each procedure performs a specific task, and they work together to solve the problem. Programs follow a logical flow, starting from the first procedure and moving step-by-step through the rest. For example, in a simple program that calculates the area of a rectangle, one procedure might take the length and width as inputs, and another procedure might multiply them to get the area. **Example**: FORTAN, C and COBOL.
- **Structural Programming**: This paradigm improves procedural programming by introducing the concept of breaking down the program into smaller parts (functions and modules) that are easier to understand, maintain, and modify. It focuses on organizing the program's structure using loops, conditionals, and blocks of code. By organizing the code better, it makes debugging and updating the program simpler. **Example:** Algol 60, Algol 68, PL/I, Pascal, C
- **Object-Oriented Programming (OOP)**: This is a more modern paradigm that focuses on creating objects. Instead of writing procedures to manipulate data, OOP organizes code around "objects" that represent real-world things (like a car, a person, etc.). These objects have properties (like a car's color) and methods (actions, like a car driving). OOP allows programmers to model complex systems by breaking them into smaller, manageable objects that interact with each other. **Example:** C++, Ada 95, Java, C#, Ruby, Swift.

Features of OOP: Class, Object, Polymorphism, Inheritance, Encapsulation, Polymorphism and Abstraction.

- **Class**: A class is a blueprint or a template for creating objects. It defines what properties (attributes) and behaviors (methods) the objects will have. For example, consider a class named "Car." This class will define properties like color, model, brand, and behaviors like "drive" or "stop." But the class itself is just a template—it doesn't represent an actual car until an object is created from it.
- **Object**: An object is an instance of a class. If "Car" is a class, then a red Toyota is an object of that class. Every object has its own set of properties defined by the class. For example, if you create two objects from

the "Car" class—one may represent a red Toyota, and the other may represent a blue Honda. Both have the same behaviors (like driving) but may have different values for properties (like color or brand).

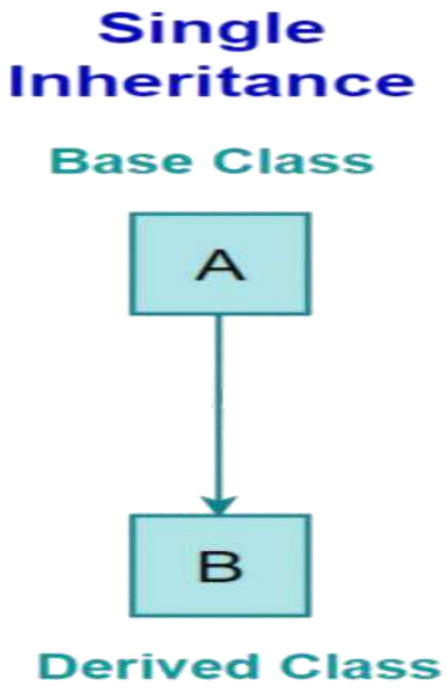
- **Inheritance**: Inheritance allows a new class to be created based on an existing class. The new class (called the **subclass** or **child class**) inherits the properties and behaviors of the existing class (called the **superclass** or **parent class**). For example, if "Vehicle" is a parent class, a "Car" class can inherit the general features of a vehicle, such as wheels or engine, but add its own specific features like air conditioning. This allows for code reuse and makes programs more organized.

  Types of Inheritance:

  i. Single
  ii. Multi level
  iii. Hierarchical
  iv. Multiple

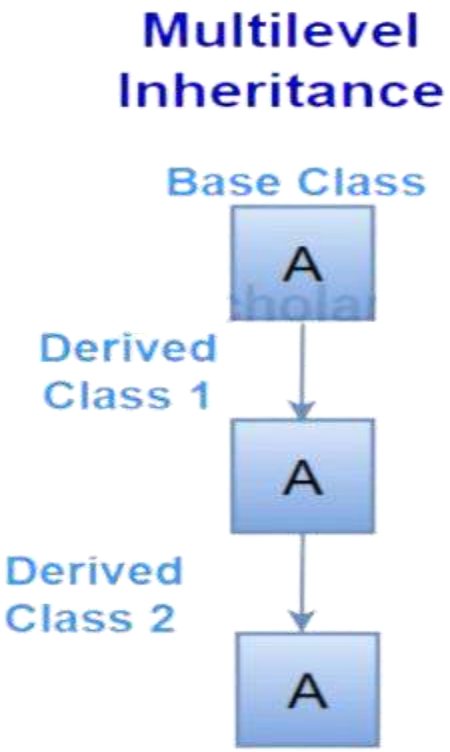
i. Single Inheritance

A class inherits from only one parent class.



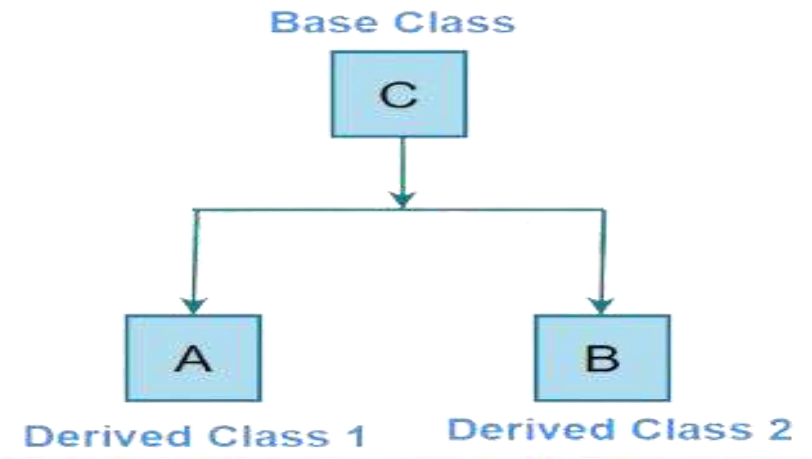
ii. Multi level Inheritance

In this inheritance, a derived class is created from another derived class.



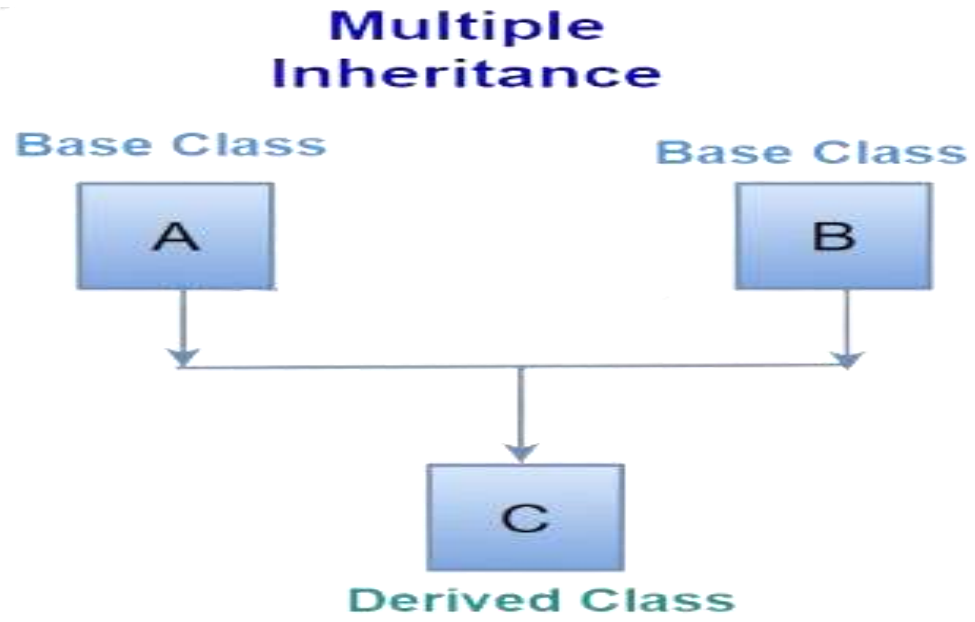
iii. Hierarchy inheritance

In this inheritance, more than one derived class is created from a single base class and further child classes act as parent classes for more than one child class.



iv. Multiple

Under this type of inheritance, the mechanism of deriving a class from another derived class.



- **Encapsulation** is like putting your important things in a box and locking it. You can only open the box with a key. In programming, encapsulation means that the important data (variables) of an object are hidden from the outside world. You can only access or change the data using specific methods, just like you can only unlock the box with the key.

Why Encapsulation is Important: It protects the data from being changed or used incorrectly. For example, if you have a class called "BankAccount," you wouldn't want someone to directly change the balance of the account. Instead, you would create methods like deposit() or withdraw() to manage the balance safely.

- **Polymorphism**: This term comes from Greek, meaning "many shapes." In OOP, polymorphism allows different objects to respond to the same method or action in different ways. For example, a method called "start" might work differently for a "Car" class and a "Bicycle" class. Polymorphism allows the same action to be performed on different types of objects, which gives flexibility to programs.

- **Abstraction** is like using a mobile phone. You don't need to know how the phone works inside to use it. You just need to know how to make a call, send a message, or take a picture. In programming, abstraction means showing only what is necessary and hiding the details of how things work.
  - **Why Abstraction is Important**: It makes using complex systems easier by focusing on what they do, not how they do it. For example, when you use an online shopping app, you just click "Buy" without needing to know how the payment system works in the background.

Advantages of OOP

- **Reusability**: One of the biggest advantages of OOP is that you can reuse existing code. Once a class is written, it can be reused across different programs or parts of the same program. For example, if you create a class for a "Car" in one program, you can reuse it in another program without having to write the code again.
- **Modularity**: In OOP, programs are organized into classes and objects, making the code more modular. This means that different parts of the program are divided into smaller sections that handle specific tasks. It's easier to maintain and modify code because changes in one module don't affect the others.
- **Scalability**: OOP makes it easy to expand or scale a program. If you want to add new features or modify existing ones, you can do so without affecting the entire codebase. For example, you can add a new class to represent a "Motorcycle" without affecting existing "Car" or "Truck" classes.
- **Data Hiding**: In OOP, you can control who has access to certain parts of your data. This concept is called **encapsulation**. By hiding the internal details of objects and only exposing what is necessary, you can make sure that your program runs more securely and that the data inside objects is protected from being changed unexpectedly.
- **Flexibility**: Polymorphism and inheritance give OOP programs flexibility. Polymorphism allows different objects to respond to the same command in different ways, and inheritance allows new objects to build on existing ones without having to start from scratch.

Disadvantages of OOP

1. **Complexity**:
   OOP can be more complex to understand and implement, especially for beginners. Concepts like inheritance, polymorphism, and encapsulation may be difficult to grasp.
2. **Size**:
   OOP programs tend to be larger in size compared to procedural programs. This is because of the use of classes, objects, and additional code for handling features like inheritance and encapsulation.
3. **Performance Overhead**:
   OOP introduces some performance overhead due to object creation, memory allocation, and garbage collection. Features like polymorphism and dynamic method calls can slow down execution.
4. **Longer Development Time**:
   Since OOP emphasizes planning and designing the structure of the system before coding, it can take more time to develop an application compared to simpler procedural approaches.

5. **Requires More Memory**:
   Objects, especially in large systems, can consume a lot of memory. The memory overhead for object storage and manipulation can be significant, particularly for smaller applications.
6. **Not Suitable for All Types of Problems**:
   For simple and small programs, OOP may be overkill. Structured programming or procedural programming might be more straightforward and efficient for smaller, less complex tasks.
7. **Difficulty with Parallel Programming**:
   OOP is less suited for parallel programming compared to functional programming. Sharing objects between multiple threads can lead to complex synchronization issues.
8. **Steep Learning Curve**:
   OOP has a steep learning curve due to the need to understand various concepts like classes, objects, inheritance, polymorphism, and design patterns, which can overwhelm new programmers.

5.4 Application of OOP

The concept of OOP is generally used in following areas.

i. Expert system
ii. Artificial intelligence
iii. Management information system
iv. Decision support system
v. Computer based training & Education
vi. Object-oriented database
vii. Computer games
viii. Mobile applications
ix. Internet based application
x. Design user interface for software
xi. Security system

- **Games**: In video game development, OOP is used to represent characters, weapons, levels, and other game elements as objects. For example, you can have a class for a "Player" with properties like health, score, and abilities, and another class for "Enemy" with different behaviors.
- **Web Applications**: OOP is used in web development to manage different parts of the website. For example, in a shopping website, you can have classes like "Product," "Cart," and "User," and objects of these classes interact to create a complete experience for the user.

- **Graphical User Interface (GUI) Applications**: OOP is widely used in creating desktop or mobile applications with buttons, text fields, and images. Each component of the interface is an object. For example, a button in an app can be an object with properties like color and size, and actions like being clicked.
- **Mobile Apps**: OOP is essential in mobile app development because it organizes complex code into objects that can represent app components, such as forms, data, or even animations.

Difference between

OOP	Structure Programming Language
OOP is a programming paradigm using objects, and methods to design applications and computer programs.	A structured programming language is a collection of instructions, which are executed by the computer sequentially.
It uses a bottom-up development process.	It uses a top-down development process.
Class and object are the main ideas of object-oriented programming.	Function (library or user-defined) is the main idea of structured programming.
It includes features such as data abstraction, encapsulation, modularity, polymorphism, and inheritance.	It doesn't include such features.
In OOP emphasis is given to data than the procedure.	Emphasis is given to the procedure.
Programs development will be easier due to features like inheritance, polymorphism.	Program development will be comparatively difficult.
New data and functions can be easily added when they are required.	Difficult and time-consuming to add new data and functions.
Higher security can be maintained by hiding data so that it cannot be accessed by external functions.	It doesn't contain any mechanism to hide data.
Easy to reuse program codes.	It doesn't contain a proper mechanism for reusing program codes.
Global variables can be restricted to be used by all the functions.	Global variables are accessible to all functions so chances of data corruption exist
Example of OOP language: C++, Java, VB.Net, C#, etc.	Examples of a structured programming language: C, FORTRAN, COBOL, etc.