INTRODUCTION

UNIT 1

UNIT - I

Introduction: Fundamentals of object oriented programming - procedure oriented programming vs. Object Oriented Programming (OOP), Object Oriented Programming concepts - classes, reusability, encapsulation, inheritance, polymorphism, dynamic binding, and message passing. C++ Programming Basics: Output using cout - directives - input with cin - type bool - Manipulators - type conversions. Functions: Returning values from functions - reference arguments - overloaded functions - inline functions - default arguments - returning by reference

Procedure-oriented language (POL)

- **procedure-oriented language (POL)** is a programming language that follows a step-by-step, sequential approach, breaking down problems into a series of instructions (procedures or functions) that are executed in order to achieve a result, focusing on how to do something rather than *what* to do.
- C, Pascal, and FORTRAN organize code into reusable blocks (routines/subroutines) and control the flow using loops and conditions.
- Procedural languages are simple and good for small tasks, but become hard to maintain and scale for large, real-world systems.

Procedure-oriented language (POL)

What is procedural language?

Follows a top-down, step-by-step approach: program is divided into functions/procedures that are called in some sequence.

Data is mostly global/local variables passed between functions; functions and data are separate concepts.

Examples: C, Fortran, Pascal, BASIC

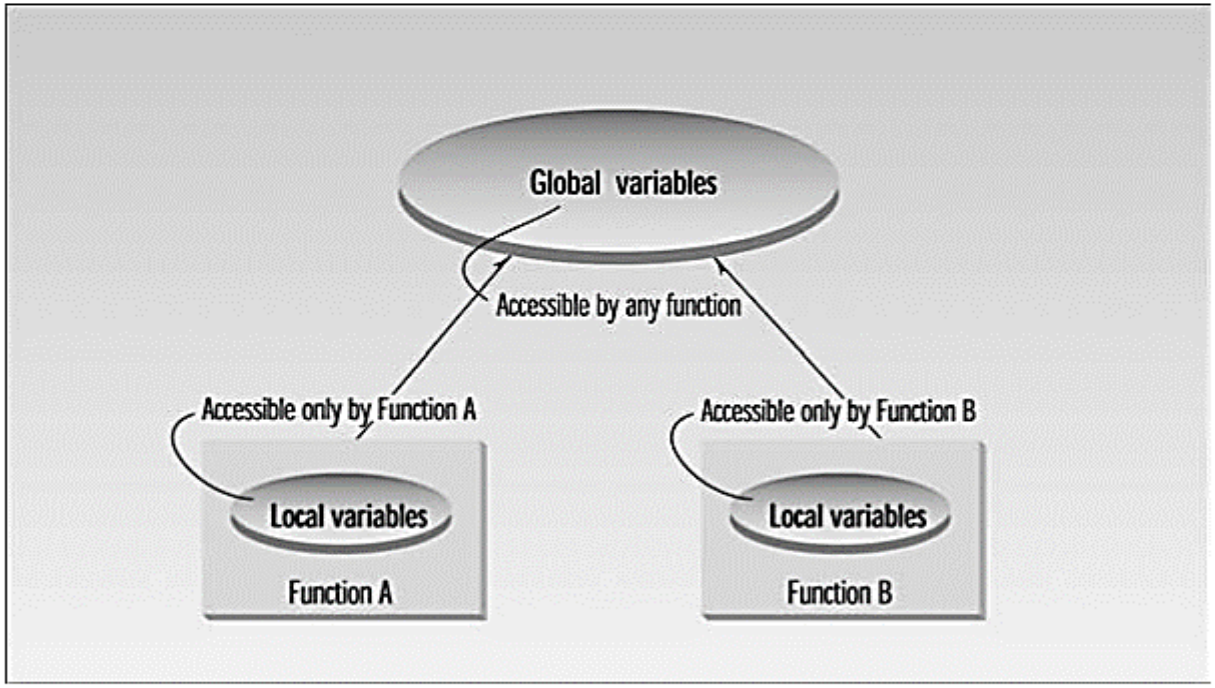
Limitations

Poor data security: No built-in data hiding, global data can be accessed and modified from many functions, making it less secure.

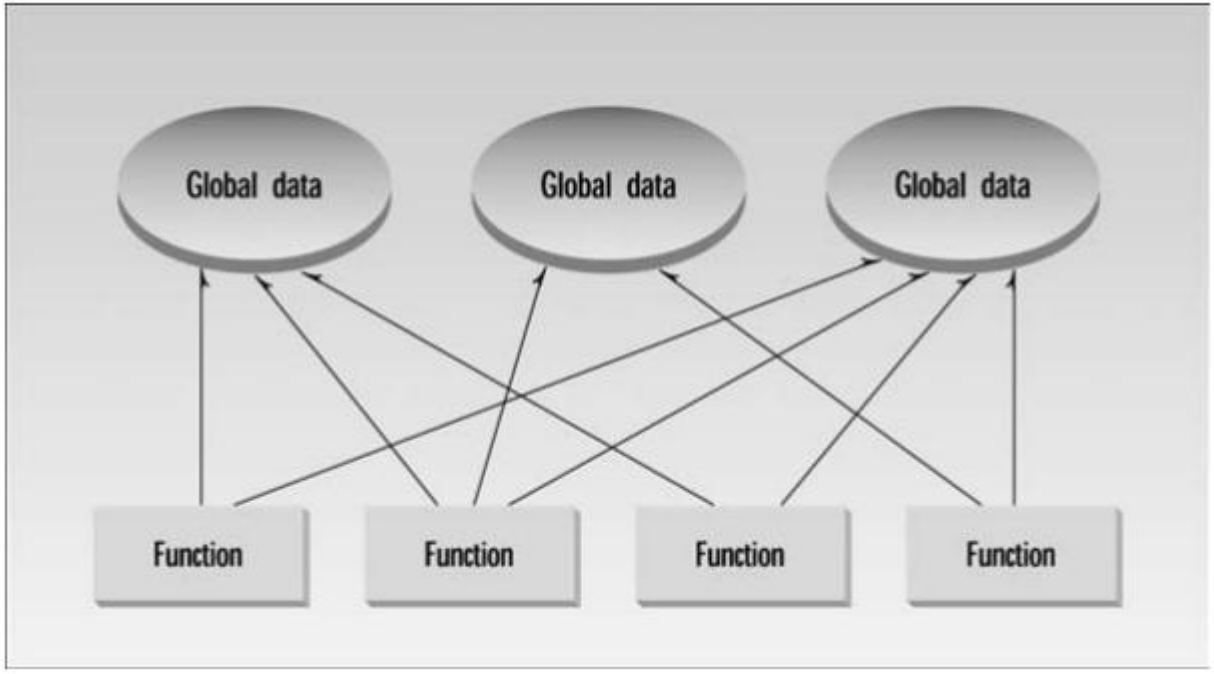
Limited reusability: No inheritance and weak modularity

Not natural for real-world modeling: Hard to represent real-world entities and their relationships; focus is on procedures, not on real objects.

Procedure-oriented language (POL)



Procedure-oriented language (POL)



Fundamentals of Object Oriented Programming

- OOP provides a way to organize and manage complex codebases by encapsulating data and behavior into objects, which can interact with each other to perform complex tasks.
- Object-Oriented Programming makes it easier to create modular, reusable, and maintainable software.
- It is a programming paradigm that relies on the concept of **Objects** & **Classes**.
- With the help of this core principle, designing code and building software becomes much easier. Object-Oriented Programming makes it easier to create modular, reusable, and maintainable software.

Real-World Modeling

Real-world modeling refers to representing real-life entities in a program in a natural way.

In the real world, objects are defined by both attributes and behavior:

Example 1: Car

Attributes (Data): color, speed, number of doors, horsepower

Behavior (Functions): start(), accelerate(), brake(), stop()

A car is not just its data (speed, color) and not just its actions (braking, accelerating), but a combination of both.

Example 2: Person

Attributes (Data): name, age, job title, eye color

Behavior (Functions): speak(), work(), walk(), makeDecision()

A person is defined by what they have (attributes) and what they do (behavior).

Fundamentals of Object Oriented Programming

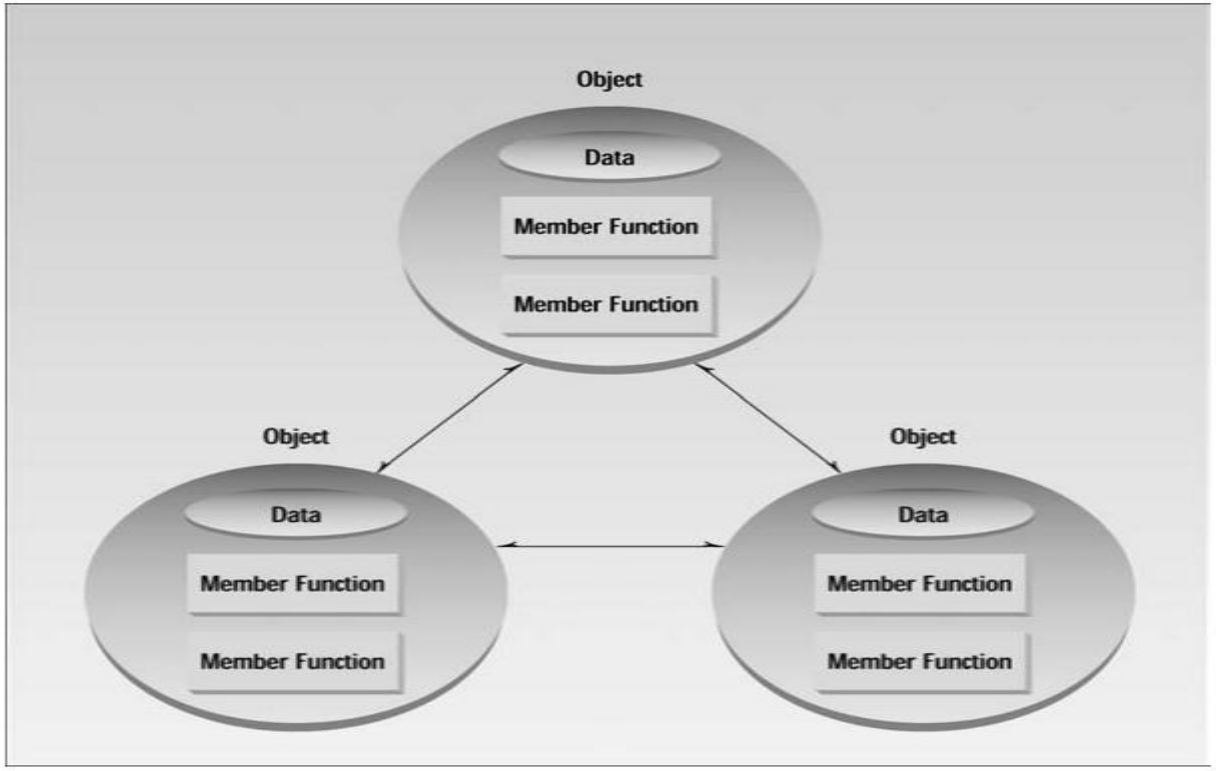
Better modeling of real world: programs are built from objects representing real entities, with both data (attributes) and behavior (methods) together.

Encapsulation and security: data is hidden inside classes, and access is controlled via public/private/protected members, improving data security.

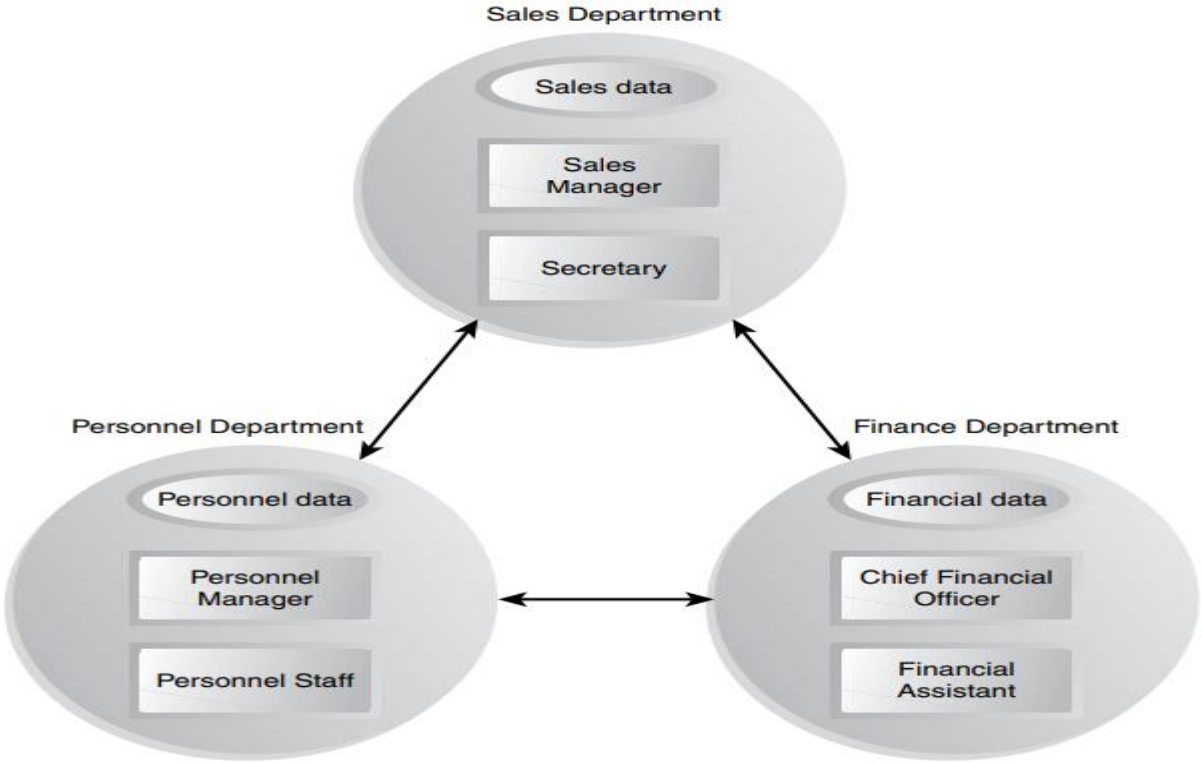
Reuse and scalability: inheritance and polymorphism let you reuse and extend existing classes instead of rewriting code, which is ideal for large systems.

Easier maintenance: modular classes make it simpler to localize bugs and add new features without breaking other parts.

Fundamentals of Object Oriented Programming



Fundamentals of Object Oriented Programming

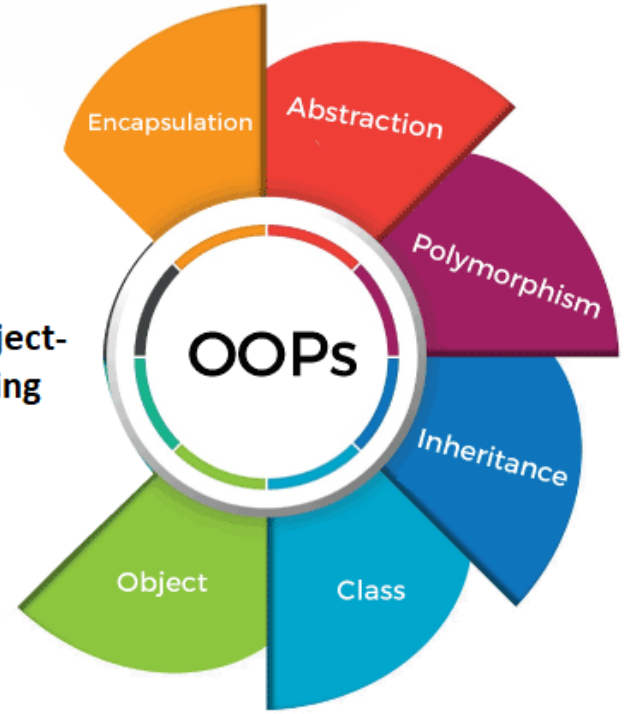


Procedural vs. Object Oriented Programming (OOP)

Procedural Oriented Programming	Object-Oriented Programming
The program is divided into small parts called functions.	The program is divided into small parts called objects.
Top-down approach.	Bottom-up approach.
No access specifier	Access specifiers like private, public, protected are used
It is less secure.	Provides data hiding so it is more secure.
Overloading is not possible.	Overloading is possible
No concept of data hiding and inheritance.	the concept of data hiding and inheritance is used.
The function is more important than the data.	Data is more important than function.
Procedural programming is used for designing medium-sized programs.	Object-oriented programming is used for designing large and complex programs.
Examples: C, FORTRAN, Pascal, Basic, etc.	Examples: C++, Java, Python, C#, etc.

Fundamentals of Object Oriented Programming

Fundamentals of object-oriented programming



Fundamentals of Object Oriented Programming

Object-Oriented Programming (OOP) is a powerful paradigm that allows developers to model real-world entities as objects, making code more organized, modular, and easier to maintain.

Fundamentals concept in OOP:

1.Classes and Objects
2.Encapsulation
3.Inheritance
4.Polymorphism
5.Abstraction

Characteristics of Object Oriented Programming

Class

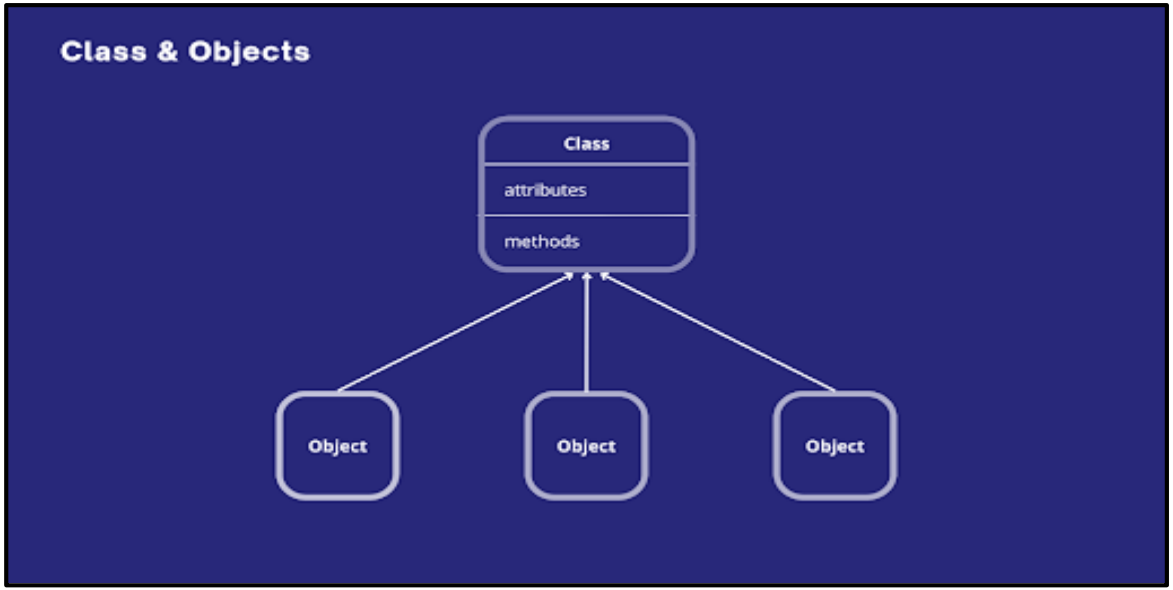
A class is a blueprint or template for creating objects. It defines the properties (attributes) and behaviors (methods) that the objects created from the class will have.
It contains data members (attributes) and member functions (methods).

Object

An object is an instance of a class. It represents a specific entity created using the blueprint provided by the class. Each object has its own copy of the attributes defined in the class but shares the methods.

- Objects are created from classes.
- They have a unique identity and store specific data.

Characteristics of Object Oriented Programming



Characteristics of Object Oriented Programming

Encapsulation

- Encapsulation is the bundling of data (variables) and methods (functions) that operate on the data into a single unit called a class.
- It also restricts direct access to some of an object's components, enforcing controlled interaction through access modifiers (public, private, protected).
- Purpose: To achieve data hiding, ensuring internal object details are hidden from external modification. This helps maintain security and reduces unintended interference.

Data hiding

- Restricts direct access to that internal data using access modifiers

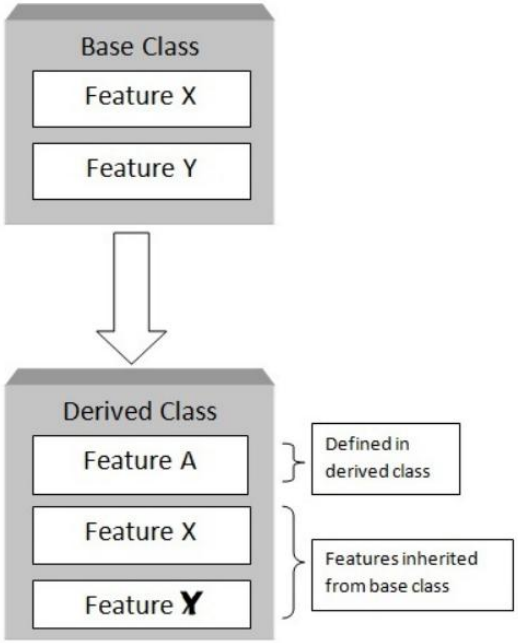
The **data is hidden**, so it is safe from accidental alteration. Data and its functions are said to be encapsulated into a single entity. Data encapsulation and data hiding are key terms in the description of object-oriented languages.

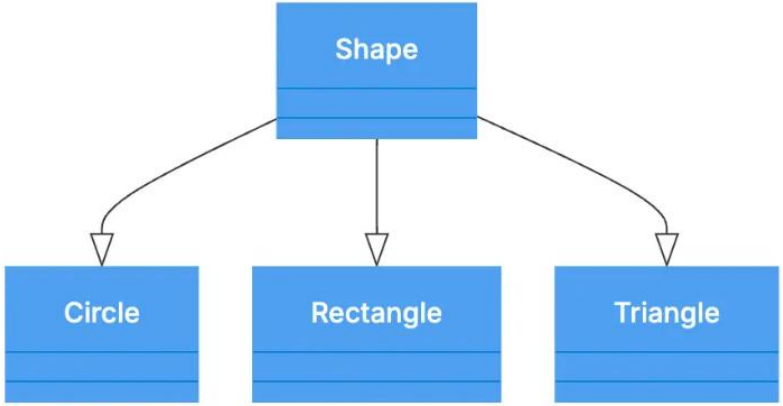
Characteristics of Object Oriented Programming

Inheritance

- Inheritance allows one class (child/derived/subclass) to acquire the properties and methods of another class (parent/base/superclass).
- It promotes code reuse and establishes a hierarchy between classes.
- Purpose: To avoid redundancy, enhance maintainability, and create specialized implementations of existing functionality.

Characteristics of Object Oriented Programming





Characteristics of Object Oriented Programming

Polymorphism

- Polymorphism allows methods or functions to behave differently based on the context, such as different implementations for the same method in different classes.

It comes in two forms:

- Compile-Time Polymorphism (Method Overloading): Same method name with different parameter lists.
- Run-Time Polymorphism (Method Overriding): A subclass redefines a method from the parent class.
- Purpose: To enable flexibility and scalability in code.

Abstraction

- Abstraction is the concept of hiding implementation details and showing only the essential features of an object. It can be achieved using abstract classes and interfaces.
- Purpose: To reduce complexity, improve code readability, and focus on relevant functionalities without exposing unnecessary details.

Characteristics of Object Oriented Programming

Dynamic Binding

- Dynamic binding (also called late binding) is a concept in object-oriented programming (OOP) where the function call is resolved at runtime, not at compile time.

- Dynamic binding or late binding is a programming concept where the specific method to be executed is determined at runtime (when the code runs), not at compile time, allowing for flexible, polymorphic code where the actual object's type dictates which overridden method is called.

Message Passing

Message passing is the process by which one object sends a message to another object to invoke a method.
A message consists of:
Object name
Method (function) name
Arguments (if any)
Syntax: object.method(arguments);