**Object-Oriented Programming & System (OOPS) concepts helps reduce code complexity and enables the reusability of code. Programmers feel like working with real-life entities or objects. Object-oriented programming is a programming paradigm that brings together data and methods in a single entity called object. This promotes greater understanding as well as flexibility and maintenance of code over a long period of time.**

## Class

A user-defined data type that defines properties and functions. A class is a logical representation and doesn't occupy memory until an object is created. Example: A human is a class with body parts as properties and actions as functions.

## Object

An **object** is an instance of a class that holds specific values for the class's attributes and can perform actions defined by its methods.

student s = new student();

Note : When an object is created using a new keyword, then space is allocated for the variable in a heap, and the starting address is stored in the stack memory. When an object is created without a new keyword, then space is not allocated in the heap memory, and the object contains the null value in the stack.

 P p(0, 0, 0); is allocated on the stack, with automatic memory management.

 P\* p = new P(0, 0, 0); is allocated on the heap, requiring manual memory management.

## Inheritance

**Inheritance is a mechanism where one class (child) derives attributes and behaviours from another class (parent)**

class derived\_class : visibility-mode base\_class;

Visibility modes: {private, protected, public}

1. **Single Inheritance**: A class inherits from one base class.
2. **Multiple Inheritance**: A class inherits from two or more base classes.
3. **Hierarchical Inheritance**: Multiple classes inherit from a single base class.
4. **Multilevel Inheritance**: A class is derived from another derived class.
5. **Hybrid Inheritance** A combination of two or more types of inheritance.

**Examples:**

* **Single Inheritance:** A Smart Lock system where a BasicLock class is extended by a SmartLock class to add features like remote access and notifications.
* **Multiple Inheritance:** A Flying Car inherits properties from both Car and Airplane. (Note: In Python, multiple inheritance is supported directly, but in Java and C++, it's done using interfaces or mixins.)
* **Multilevel Inheritance:** A University system where the University has a Department, which has a Professor.
* **Hierarchical Inheritance:** A Vehicle class has subclasses like Car and Bike, which inherit common attributes.
* **Hybrid Inheritance:** A Robot that can both Walk and Speak using properties inherited from both Walker and Speaker. (Note: Hybrid Inheritance is not directly supported in Java and C++ due to complexity and the diamond problem but can be achieved using interfaces or virtual inheritance.)

## Encapsulation

**Binding (or wrapping) code and data together into a single unit is known as encapsulation.**

Data is accessed through the functions present inside the class. Attributes of the class are kept private, public getter and setter methods are provided to manipulate these attributes.

Encapsulation is the concept of wrapping data (attributes) and methods that operate on that data into a single unit (class) while restricting direct access to some of the object's components

* Imagine a Safe Box where valuables (attributes) are stored. You can only access the contents using a key (methods), not directly.
* A capsule which is mixed of several medicines. The medicines are hidden data to the end user.

## Abstraction

Abstraction is the concept of hiding the internal implementation details and exposing only the essential features of an object. Exposing only the information that is absolutely necessary while concealing implementation or background information.

* A Coffee Machine exposes a simple interface (buttons for coffee, latte, etc.) but hides the internal brewing process.
* A driver will focus on the car functionality (Start/Stop -> Accelerate/Brake), he/she does not bother about how the accelerate/brake mechanism works internally. And this is how the abstraction works.

## Polymorphism

Polymorphism is the ability of a function, object, or method to take on multiple forms. It allows various classes to implement the same interface while having distinct underlying data. For instance:

**Real-Life Examples:**

* A MusicPlayer can play songs, but it may play different formats like MP3, WAV, or FLAC using the same method.
* A delivery person delivers items to the user: If it’s a postman, he will deliver the letters; if it’s a food delivery boy, he will deliver the foods. Polymorphism implements different ways for the delivery function.

**Examples:**

* **Method Overloading:** A Smart Door can be opened using a key, a password, or a fingerprint.
* **Method Overriding:** Different types of doors (manual, automatic) have different ways to open.

Types of Polymorphism

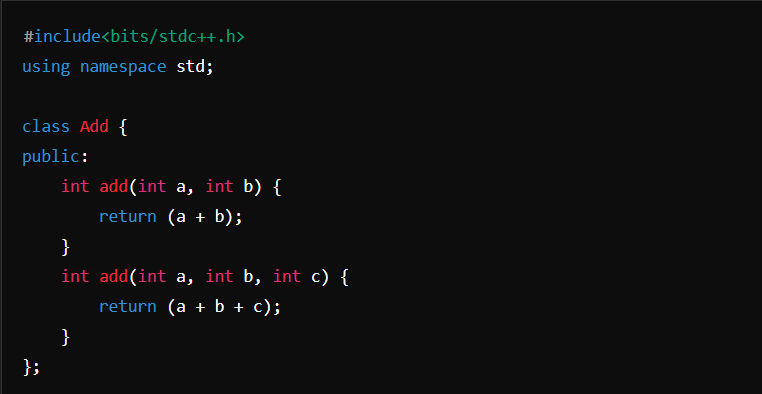
Compile-Time Polymorphism: Implemented at the compile time, Includes **method overloading and operator overloading.**

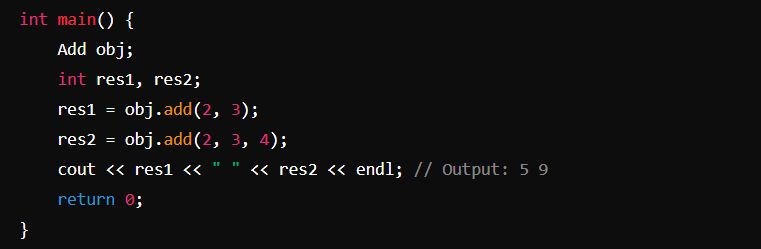
Method Overloading: More than one function with the same function name but with different functionality.

1. The return type of the overloaded function.

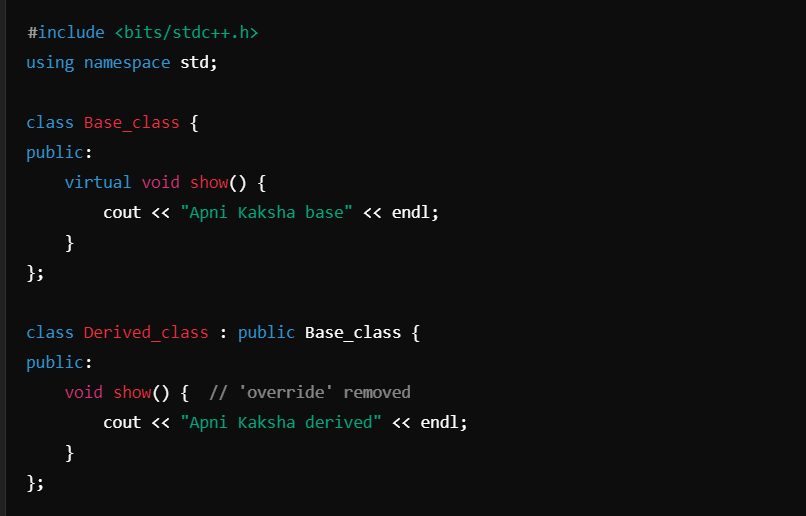
2. The type of the parameters passed to the function.

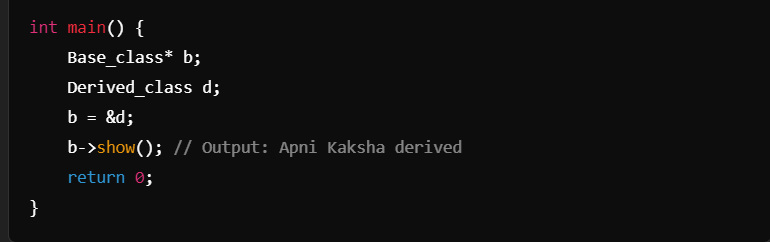
3. The number of parameters passed to the function.





Runtime Polymorphism: The ability of a program to determine which method to execute at runtime based on the actual object type, rather than the type known at compile time. This is achieved through **method overriding.**





Difference Between Encapsulation and Data Hiding

**Encapsulation**: Encapsulation refers to the concept of bundling data (attributes) and methods that operate on that data into a single unit (class). It restricts direct access to some components but allows controlled access through public methods.

**Example**: A class with private attributes that can be accessed and modified through public getter and setter methods.

**Data Hiding**: Data hiding is a specific aspect of encapsulation that focuses on restricting access to certain details of an object’s implementation. It prevents outside interference and misuse of the object's internal state.

**Example**: Declaring class attributes as private, making them inaccessible from outside the class.

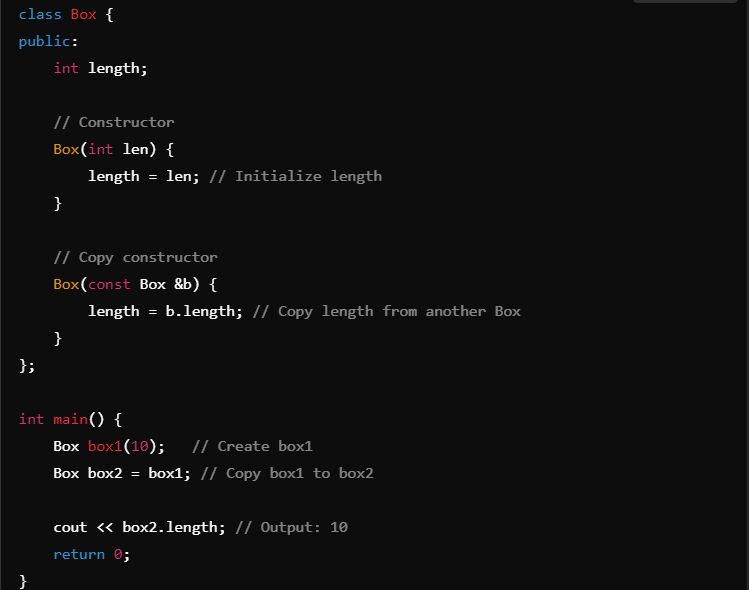
## Constructor

Constructor is a special method which is invoked automatically at the time of object creation. It is used to initialize the data members of new object. It has the same name as class, should be public and does not have return type.

1. Default constructor: A constructor which has no argument.

2. Parameterized constructor: Constructor which has parameters.

3. Copy Constructor: A Copy constructor is an overloaded constructor used to declare and initialize an object from another object. **types** - default copy constructor and user defined copy constructor.



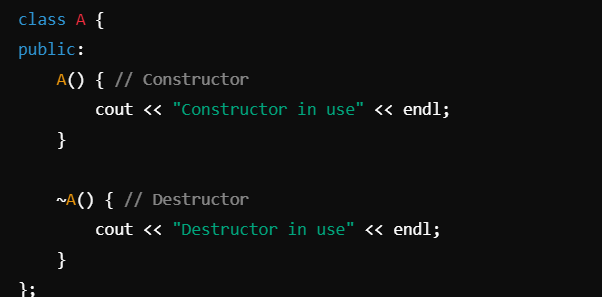
**Constructor overloading** is the process of defining multiple constructors in the same class with different numbers or types of parameters. This allows objects to be initialized in different ways based on the arguments passed during object creation.

**Constructor overriding** does not exist in C++ because constructors are not inherited from base classes.

**Constructor chaining** in C++ refers to the practice of calling one constructor from another within the same class or from a base class to initialize an object. It is commonly used to avoid redundancy when multiple constructors perform similar initialization tasks.

**Two Types of Constructor Chaining: Constructor Chaining within the Same Class, Constructor Chaining between Base and Derived Classes:**

**Destructors** are methods invoked automatically when an object goes out of scope. They are used to perform cleanup tasks and have the same name as the class, prefixed with a tilde (~).



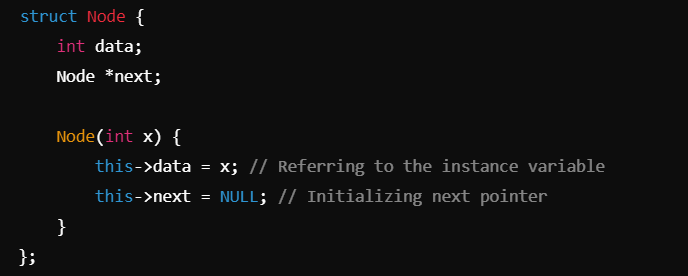
## ‘this’ Pointer

The this keyword in C++ refers to the current object. In a constructor, you can use this to refer to the current object’s data members and differentiate between local variables and data members if they have the same name.

1. It can be used to pass the current object as a parameter to another method

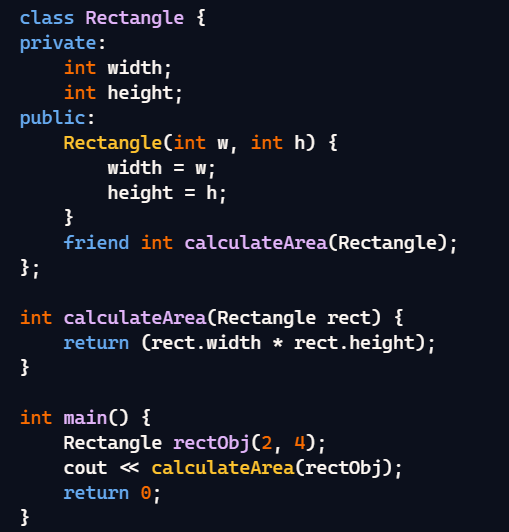
2. It can be used to refer to the current class instance variable.

3. It can be used to declare indexers.



## Friend Function

The friend function is a non-member function and has the ability to access the private and protected members of the class.





## Aggregation

It is a process in which one class defines another class as any entity reference. It is another way to reuse the class. It is a form of association that represents the HAS-A relationship.

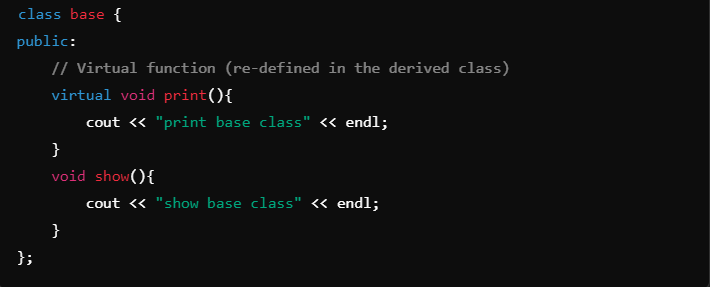
## Virtual Function

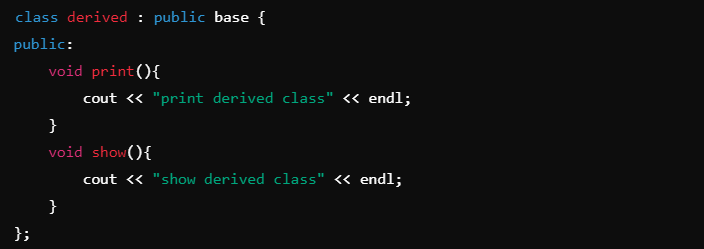
A **virtual function** is a member function in a base class that can be overridden in derived classes. It enables **runtime polymorphism**, allowing the program to decide which function to execute based on the type of the object being pointed to, rather than the type of pointer used to call the function.

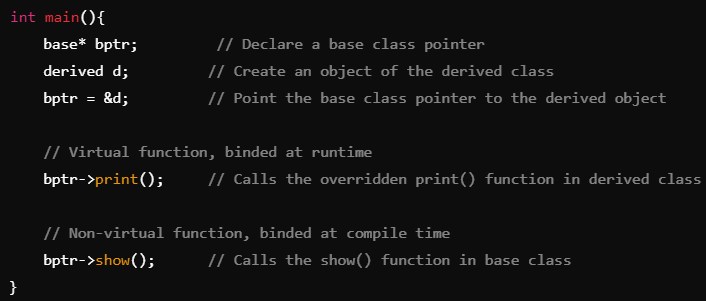
Key Points :

1. Virtual functions cannot be static.

2. A class may have a virtual destructor but it cannot have a virtual constructor.

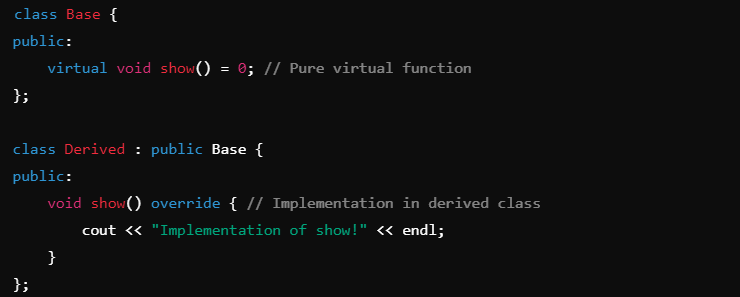






## Pure Virtual Function

A **pure virtual function** is a function declared in a base class that has no implementation in that class and is meant to be overridden in derived classes.

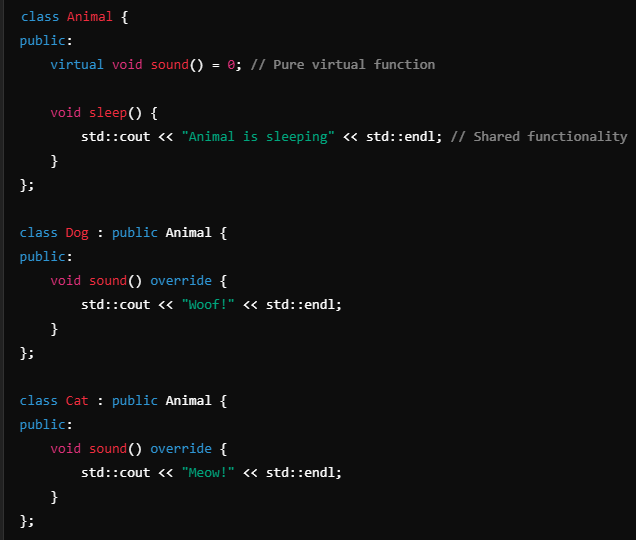


## **Abstract class**

is a class that cannot be instantiated directly and is designed to serve as a base class for other classes. It contains at least one pure virtual, which must be implemented by derived classes.

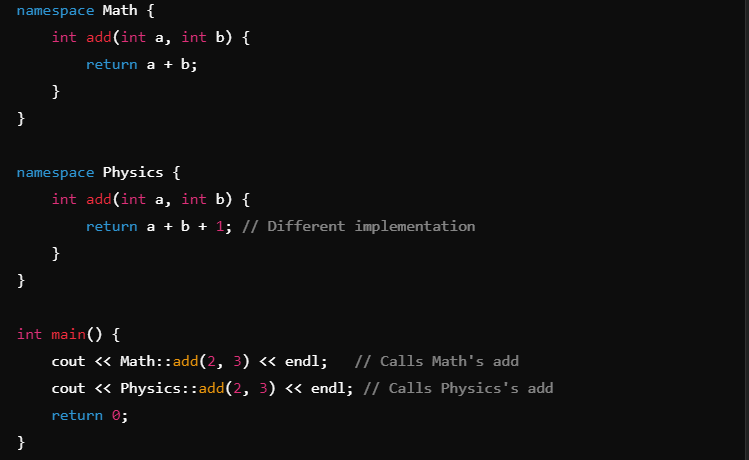
An **abstract class** is used as a blueprint for other classes. It is a class that cannot be instantiated on its own but can be used to define common behaviors for derived classes

Ex: A blueprint for a building:



## Namespaces

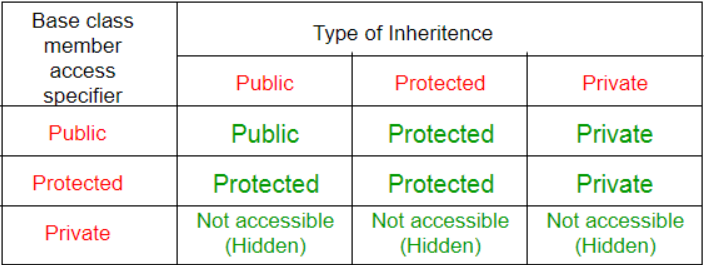
Namespaces are logical divisions of code that help prevent naming conflicts. When multiple entities share the same name, namespaces help avoid ambiguity by allowing you to specify which version to use.



## Access Specifiers

Define how functions and variables can be accessed outside the class.

Public, Private, Protected



**Memory Deallocation**: delete: single object, delete[] for an array of objects.

**Virtual Inheritance**: Ensures only one copy of a base class object exists in the derived class hierarchy, preventing ambiguity.

**Function Overloading**: Allows multiple functions with the same name but different parameter types or counts, enabling different behaviors based on arguments.

**Operator Overloading**: Enables redefining standard operators (like +, -, etc.) to work with user-defined class types, giving them specific meanings.

**Binding Types**:

* **Overloading**: Static binding. Same method name with different parameters within the same class.
* **Overriding**: Dynamic binding. Same method name and parameters in both base and derived classes, where the derived class method replaces the base class method.

**Data binding** : Data binding is a process of binding the application UI and business logic. Any change made in the business logic will reflect directly to the application UI.