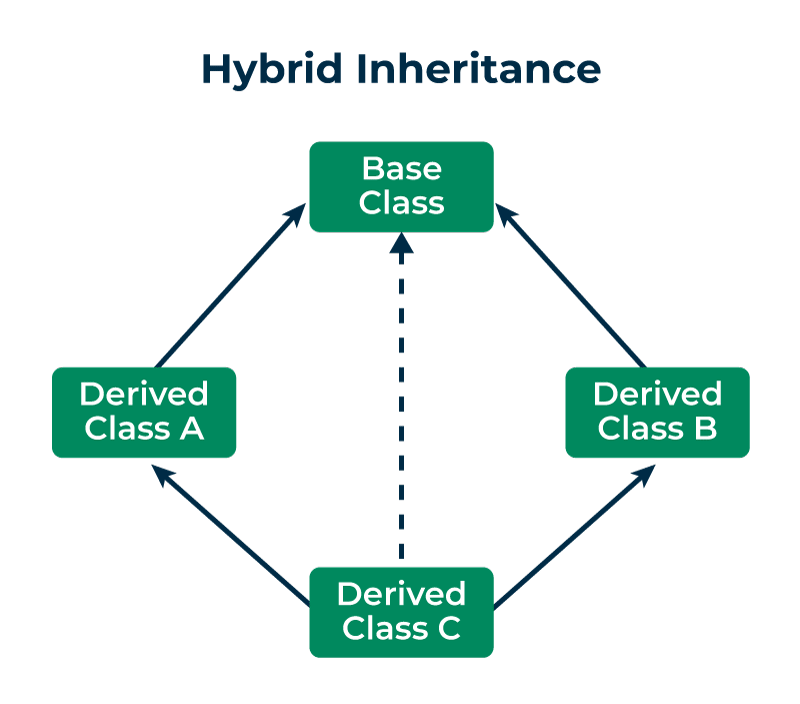
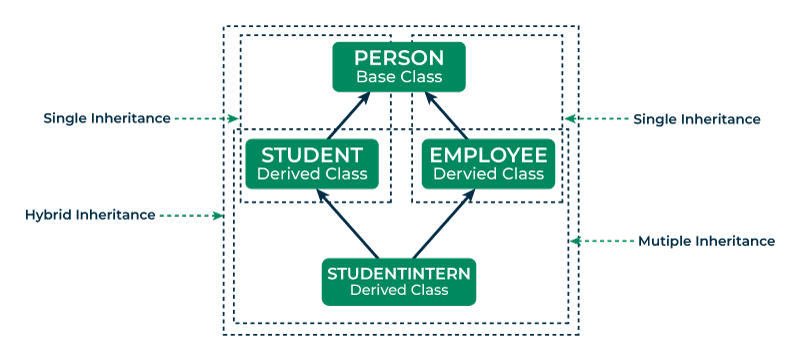
* **Hybrid Inheritance in C++**Hybrid inheritance is a complex form of inheritance in object-oriented programming ([OOP](https://www.geeksforgeeks.org/object-oriented-programming-in-cpp/)). In Hybrid Inheritance, multiple types of inheritance are combined within a single class hierarchy, enabling a varied and flexible structure of classes. In hybrid inheritance, within the same class, we can have elements of **single inheritance**, **multiple inheritance**, **multilevel inheritance**, and **hierarchical inheritance**.  
    
  *Hybrid Inheritance*The major goal of **hybrid inheritance** is to enhance code reusability by making it simpler for programmers to use the methods and attributes that are already present in other classes. Hybrid inheritance has several advantages over other inheritances as it increases the reusability of software elements, enables quicker code development lowers coding errors by avoiding code duplication across classes, and establishes a more clearly defined relationship between classes in **object-oriented programming**. It also provides a structured way to organize classes with shared attributes and behaviors.  
  However, Hybrid Inheritance also poses some difficulties, like the possibility of ambiguity or inconsistencies between inherited attributes and methods. While implementing hybrid inheritance, careful design and a thorough understanding of the principles of programming languages are essential to ensure that it achieves its goals successfully and without introducing any unnecessary complications.  
  **Examples of Hybrid Inheritance  
  Example 1: Using Single Inheritance and Multiple Inheritance**Let us consider a scenario where we have a base class “Person”, a derived class “Employee” that uses **single inheritance**, and another derived class “Student” that also uses [single inheritance](https://www.geeksforgeeks.org/difference-between-single-and-multiple-inheritance-in-c/) but combines with “Employee” to create a hybrid inheritance.  
     
     
  // C++ program to illustrate the hybrid inheritance   
  #include <bits/stdc++.h>   
  **using** **namespace** std;  
     
  // Base class   
  **class** Person {   
  **protected**:   
   string name;   
     
  **public**:  
   Person(**const** string& name)   
   : name(name)   
   {   
   }   
   **void** display() { cout << "\nName: " << name << endl; }   
  };   
     
  // Derived class 1: Employee (Single Inheritance)   
  **class** Employee : **public** Person {   
  **protected**:   
   **int** employeeId;   
     
  **public**:  
   Employee(**const** string& name, **int** id)  
   : Person(name)   
   , employeeId(id)   
   {   
   }   
   **void** displayEmployee()   
   {   
   display();   
   cout << "Employee ID: " << employeeId << endl;   
   cout << "Method inside Derived Class Employee"  
   << endl;   
   }   
  };   
     
  // Derived class 2: Student (Single Inheritance)   
  **class** Student : **public** Person {   
  **protected**:   
   **int** studentId;   
     
  **public**:  
   Student(**const** string& name, **int** id)  
   : Person(name)   
   , studentId(id)   
   {   
   }   
   **void** displayStudent()   
   {   
   display();   
   cout << "Student ID: " << studentId << endl;   
   cout << "Method inside Derived Class Student"  
   << endl;   
   }   
  };   
     
  // Derived class 3: StudentIntern (Multiple Inheritance)   
  **class** StudentIntern : **public** Employee, **public** Student {   
  **public**:  
   StudentIntern(**const** string& name, **int** empId, **int** stuId)   
   : Employee(name, empId)   
   , Student(name, stuId)   
   {   
   }   
   **void** displayStudentIntern()   
   {   
   cout << "Methods inside Derived Class "  
   "StudentIntern : "  
   << endl;   
   displayEmployee();   
   displayStudent();   
   }   
  };   
     
  // driver code   
  **int** main()   
  {   
   StudentIntern SI("Riya", 67537, 2215);   
   SI.displayStudentIntern();   
     
   **return** 0;   
  }  
  **Output**Methods inside Derived Class StudentIntern :   
  Name: Riya  
  Employee ID: 67537  
  Method inside Derived Class Employee  
  Name: Riya  
  Student ID: 2215  
  Method inside Derived Class Student  
  It is important to note that the “StudentIntern” class exhibits the [diamond problem](https://www.geeksforgeeks.org/inheritance-ambiguity-in-cpp/) of [multiple inheritances](https://www.geeksforgeeks.org/multiple-inheritance-in-c/) as it inherits display() from both “Employee” and “Student” which in turn inherits from “Person”.  
  **Explanation**1. An instance of the “StudentIntern” class named “SI” is created in the main function with arguments:
  + Name: Riya
  + Employee ID: 67537
  + Student ID: 2215
* 2. The “displayStudentIntern” method of the “SI” object is called.  
  3. Inside the “displayStudentIntern” method,
  + Methods inside Derived Class “StudentIntern” is displayed as a header.
  + The “displayEmployee” method of the “Employee” class is called.
  + Inside “displayEmployee”, the “display” method of the “Person” class (base class) is called to display the name along with the employee ID.
  + The “displayStudent” method of the “Student” class is called.
* Inside “displayStudent”, the display method of the “Person”class (base class) is called to display the name again along with the student ID.  
    
     
     
     
     
     
     
     
     
     
  **Example 2: Using Multilevel Inheritance and Hierarchical Inheritance**
  + C++
* // C++ program to illustrate the hybrid inheritance Using   
  // Multilevel Inheritance and Hierarchical Inheritance   
  #include <bits/stdc++.h>   
  **using** **namespace** std;  
     
  // Base class 1   
  **class** Meal {   
  **public**:  
   **void** print()   
   {   
   cout << "Different types of meals are served :"  
   << endl;   
   }   
  };   
     
  // Derived class 1 from Meal (Hierarchical Inheritance)   
  **class** Breakfast : **public** Meal {   
  **public**:  
   **void** print()   
   {   
   cout << "\nBreakfast is a type of meal." << endl;   
   }   
  };   
     
  // Derived class from breakfast (Multilevel Inheritance)   
  **class** Milk : **public** Breakfast {   
  **public**:  
   **void** print()   
   {   
   cout << "Milk served in breakfast." << endl;   
   }   
  };   
     
  // Derived class from Milk (Multilevel Inheritance)   
  **class** Yoghurt : **public** Milk {   
  **public**:  
   **void** print()   
   {   
   cout << "Yoghurt made from milk." << endl;   
   }   
  };   
     
  // Derived class 2 from Meal (Hierarchical Inheritance)   
  **class** Dessert : **public** Meal {   
  **public**:  
   **void** print()   
   {   
   cout << "\nDessert is a type of meal." << endl;   
   }   
  };   
     
  // Derived class from Dessert (Multilevel Inheritance)   
  **class** Sweets : **public** Dessert {   
  **public**:  
   **void** print()   
   {   
   cout << "Sweets served in Dessert." << endl;   
   }   
  };   
     
  // Derived class from Sweets (Multilevel Inheritance)   
  **class** Pastry : **public** Sweets {   
  **public**:  
   **void** print()   
   {   
   cout << "Pastry is a type of sweet." << endl;   
   }   
  };   
     
  **int** main()   
  {   
   Meal types;   
   Breakfast servedBreakfast;   
   Milk milk;   
   Yoghurt yoghurt;   
   Dessert servedDessert;   
   Sweets sweet;   
   Pastry pastry;   
     
   types.print();   
   servedBreakfast.print();   
   milk.print();   
   yoghurt.print();   
   servedDessert.print();   
   sweet.print();   
   pastry.print();   
     
   **return** 0;   
  }  
  **Output**Different types of meals are served :  
  Breakfast is a type of meal.  
  Milk served in breakfast.  
  Yoghurt made from milk.  
  Dessert is a type of meal.  
  Sweets served in Dessert.  
  Pastry is a type of sweet.  
  This code defines a set of classes and demonstrates **multilevel inheritance** and [hierarchical inheritance](https://www.geeksforgeeks.org/cpp-hierarchical-inheritance/) creating a hierarchy of different types of meals served and their descriptions.  
  **Explanation**1. “Meal” is the base class that has a single member function “print()” that displays a generic message about different types of meals.  
  a. “Breakfast” is derived from the class “Meal” which demonstrates Hierarchical Inheritance and overrides the function “print()” to provide information about breakfast.
  + “Milk” is derived from “Breakfast” representing Multilevel Inheritance which specializes in information about milk served in breakfast.
  + “Yoghurt” is derived from “Milk” representing Multilevel Inheritance.
* b. “Dessert” is another derived class from “Meal” which demonstrates Hierarchical Inheritance.
  + “Sweets” is derived from “Dessert” representing **Multilevel Inheritance**.
  + “Pastry” is derived from “Sweets” representing **Multilevel Inheritance**.

2. In the “main()” function, objects of various classes are created, and “print()” method is called on each object to display information about the respective type of meals. Due to method overriding, the most specific type of “print()” function for each object’s class is invoked.  
