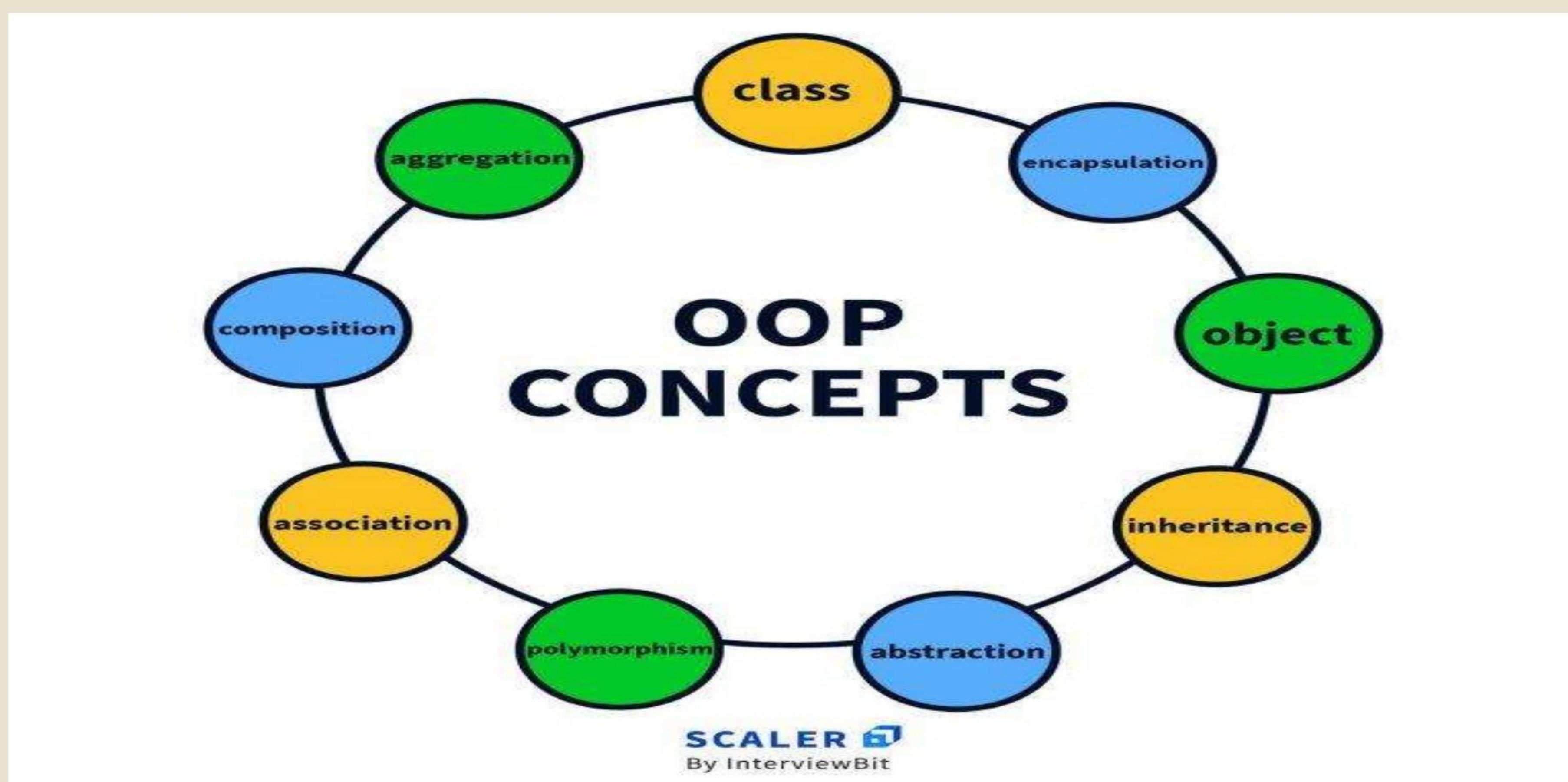


## UNIT 3



Submitted By  
Name:- Abhijeet Raj  
Reg.No.:-PCE23CS003

GROUP 5

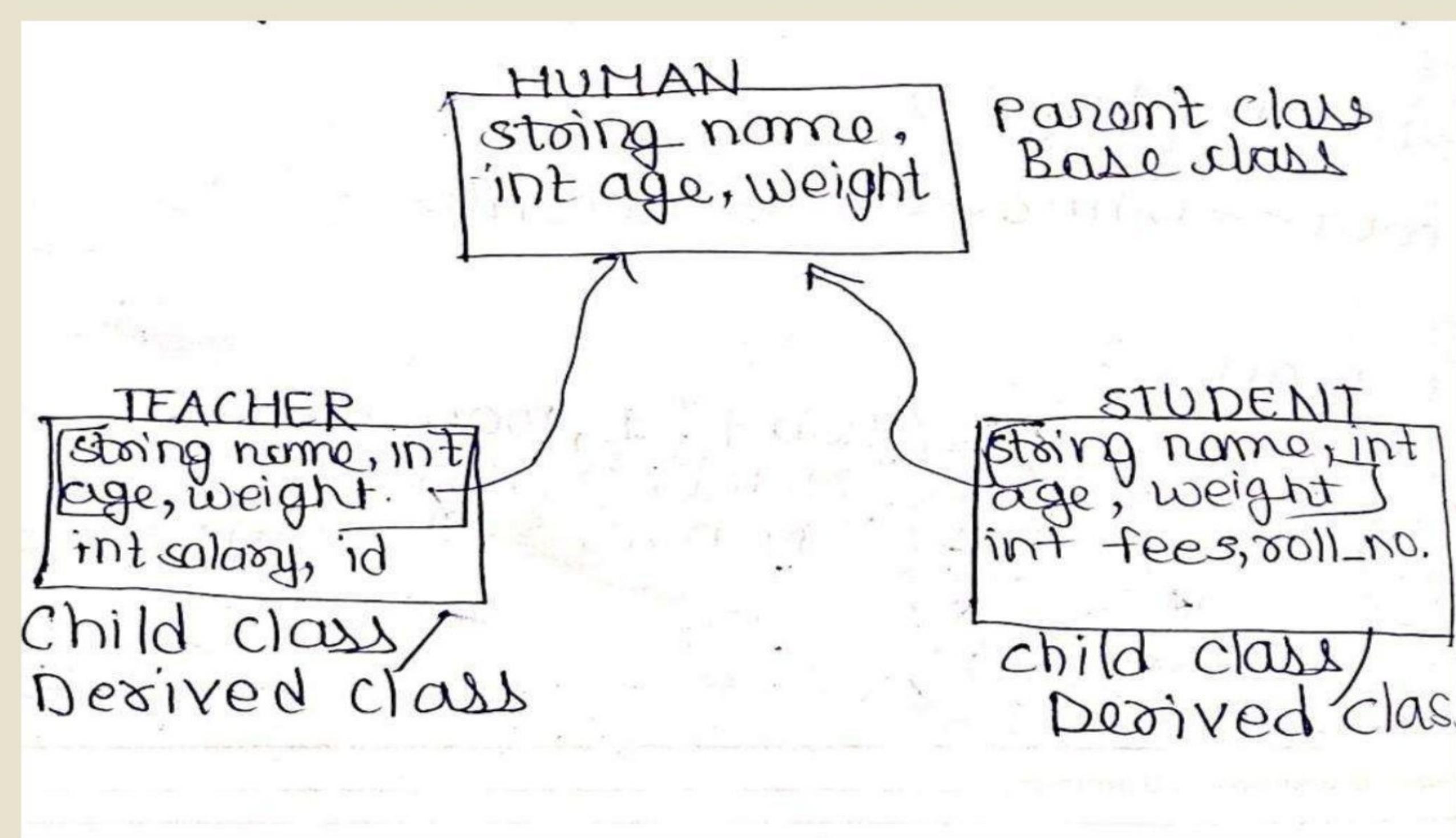
## **TOPIC OVERVIEW**

- **Inheritance**
- **Types of inheritance**
- **Virtual base class**
- **Function overriding**
- **Abstract class**
- **Pure virtual function**

**GROUP 5**

## WHAT IS INHERITANCE?

- The capacity of class to derive property and characteristics from another class.
- Parent Class (base class): The base class whose properties are inherited.
- Child class(derived class):The class that inherits from the parents.



GROUP 5

## ACCESS MODIFIER

- There are three access modifier:-

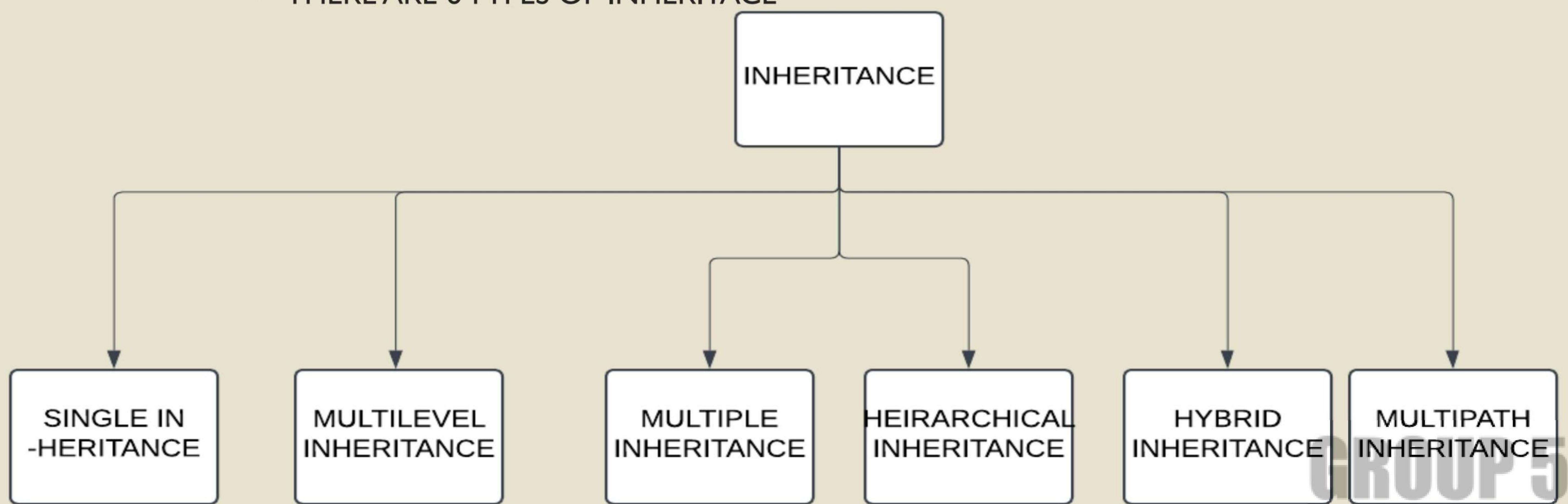
1. Private
2. Public
3. Protected

	external code	within class	derived class	
base class member access specifier				
public	✓	✓	✓	
protected	✗	✓	✓	
private	✗	✓	✗	

base class member access specifier	Type of Inheritance.		
	Public	Protected	Private
Public	Public	Protected	Private
Protected	Protected	Protected	Private
Private	Not accessible (Hidden)	not accessible (Hidden)	not accessible (Hidden)

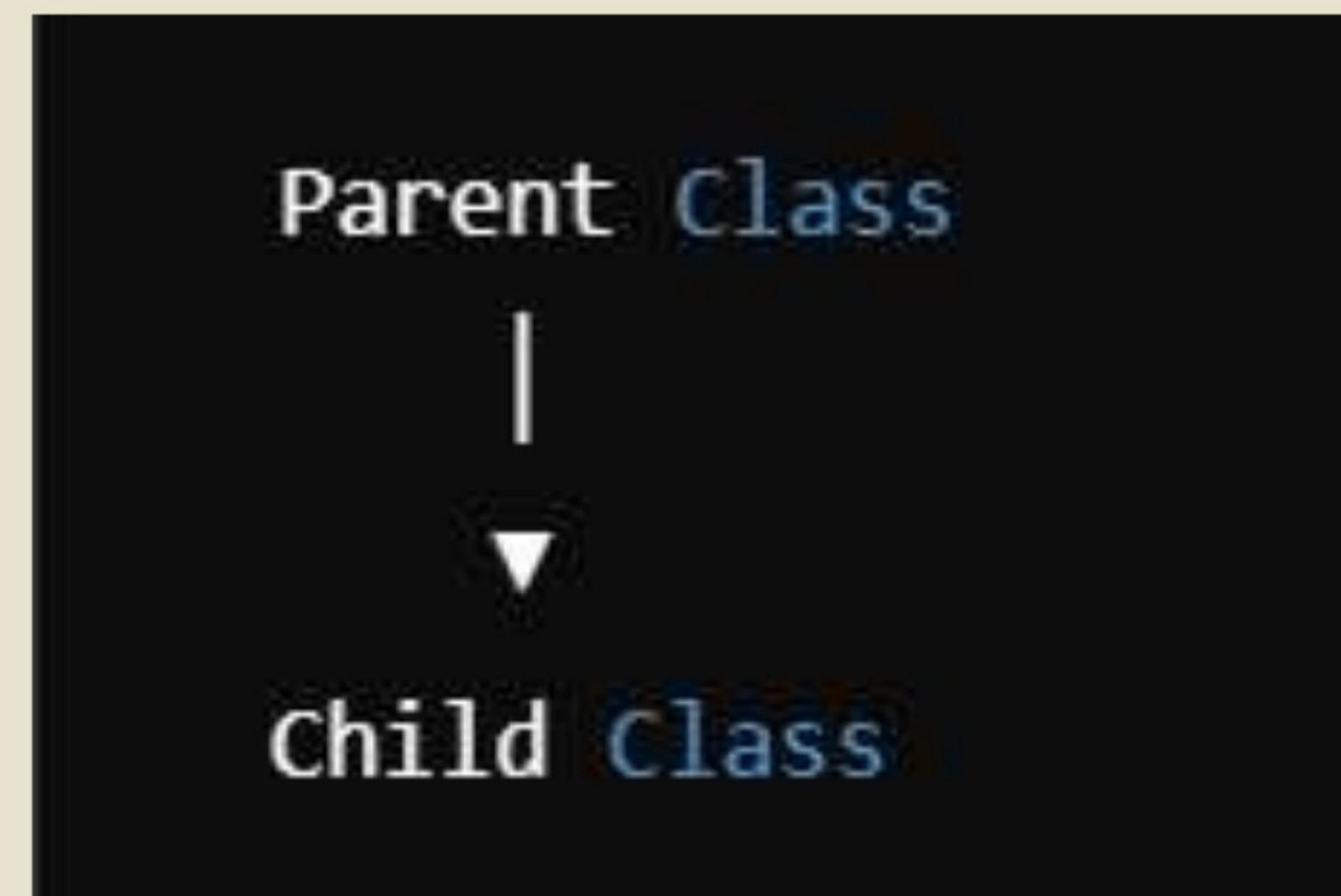
## TYPES OF INHERITANCE

- THERE ARE 6 TYPES OF INHERITANCE



## I.SINGLE INHERITANCE

- A subclass inherits from only one parent class.



**GROUP 5**

## CODE:

main.cpp



Share

Run

Output

```
1 #include <iostream>
2 using namespace std;
3
4 class Vehicle {
5 public:
6     void start() {
7         cout << "Vehicle started." << endl;
8     }
9 };
10
11 class Car : public Vehicle {
12 public:
13     void drive() {
14         cout << "Car is driving." << endl;
15     }
16 };
17
18 int main() {
19     Car car;
20     car.start();
21     car.drive();
22     return 0;
23 }
```

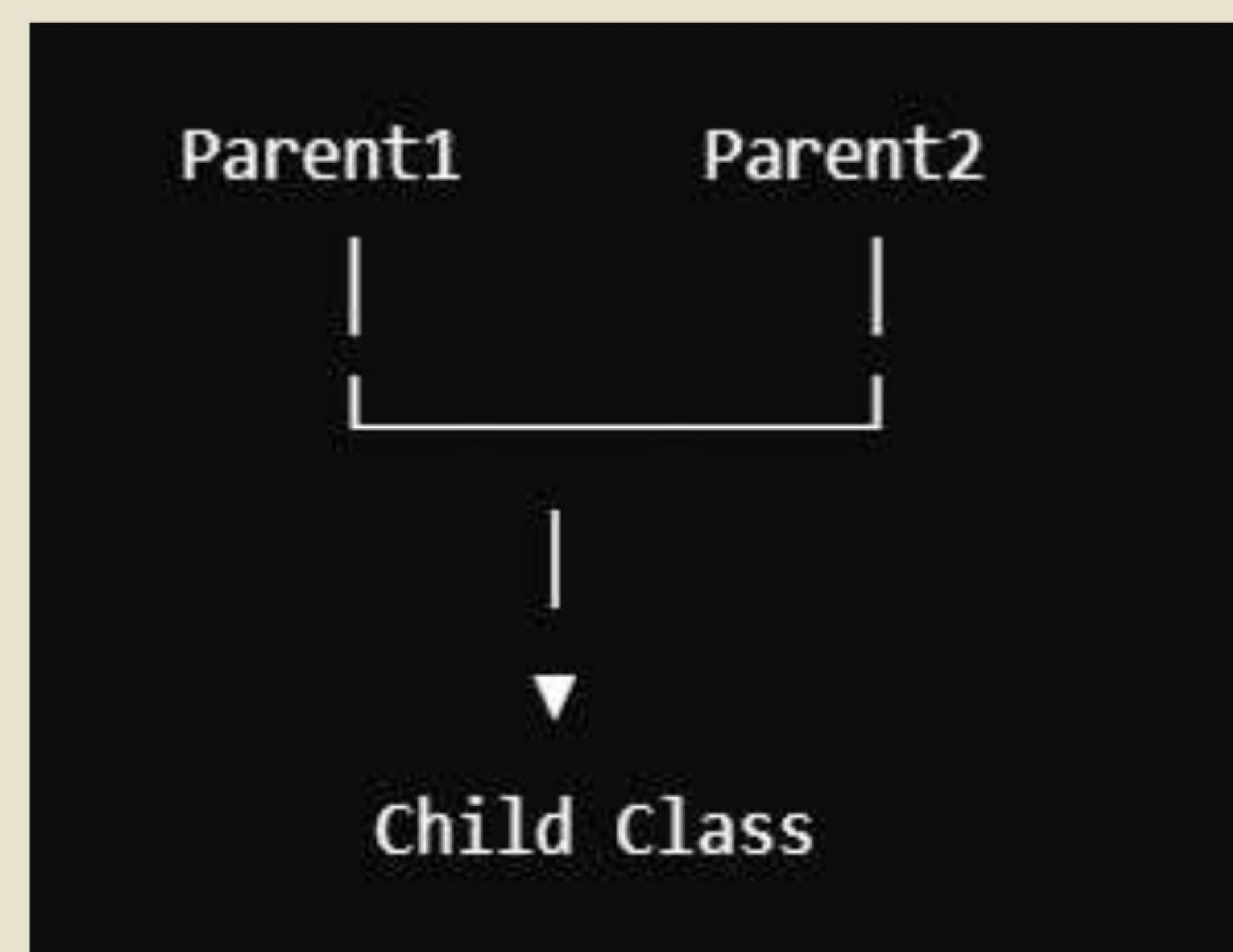
Vehicle started.  
Car is driving.

==== Code Execution Successful ===

GROUP 5

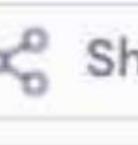
## 2. MULTIPLE INHERITANCE

- A subclass inherits from multiple parent classes.



GROUP 5

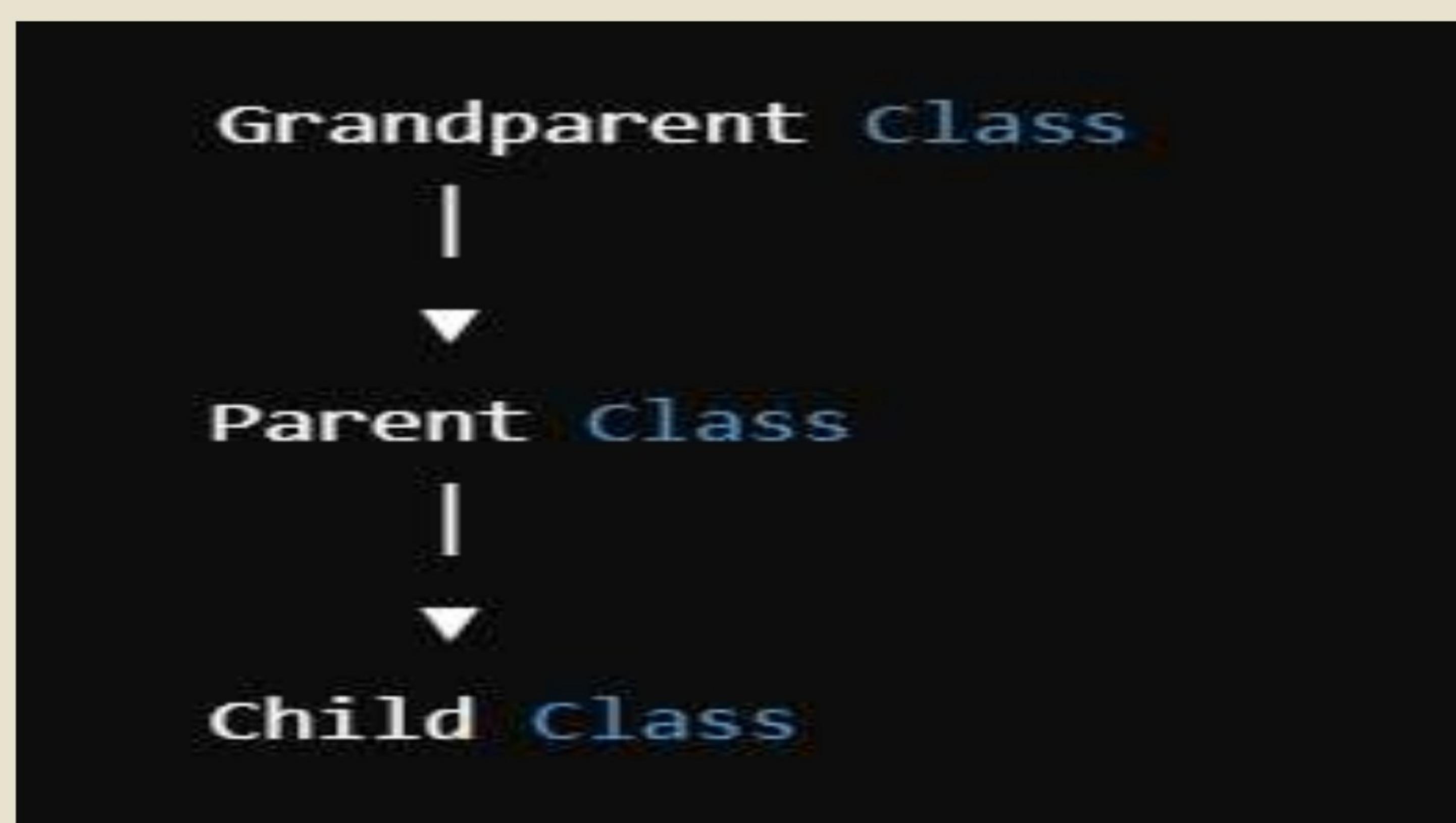
## CODE:

main.cpp	Run	Output
<pre>1 #include &lt;iostream&gt; 2 using namespace std; 3 class Person { 4 public: 5     string name; 6     void display() { 7         cout &lt;&lt; "Name: " &lt;&lt; name &lt;&lt; endl; 8     } 9 }; 10 class Subject { 11 public: 12     string subjectName; 13     void showSubject() { 14         cout &lt;&lt; "Subject: " &lt;&lt; subjectName &lt;&lt; endl; 15     } 16 }; 17 18 class Teacher : public Person, public Subject { 19 public: 20     void teach() { 21         cout &lt;&lt; name &lt;&lt; " teaches " &lt;&lt; subjectName &lt;&lt; "." &lt;&lt; endl; 22     } 23 }; 24 int main() { 25     Teacher teacher; 26     teacher.name = "Ishika"; 27     teacher.subjectName = "Mathematics"; 28 29     teacher.display(); 30     teacher.showSubject(); 31     teacher.teach();   32 33     return 0; 34 }</pre>	  	Name: Ishika Subject: Mathematics Ishika teaches Mathematics.  ==== Code Execution Successful ===

GROUP 5

### **3. MULTILEVEL INHERITANCE**

- A class inherits from a subclass, forming a chain.



**GROUP 5**

## CODE:

The screenshot shows a code editor interface with a yellow header bar containing the word "CODE:". Below the header is a toolbar with icons for file operations, a share button, and a blue "Run" button. The main area is divided into three sections: "main.cpp" on the left, a code editor in the center, and an "Output" window on the right.

**Code (main.cpp):**

```
1 #include <iostream>
2 using namespace std;
3 class Person {
4 public:
5     string name;
6     void display() {
7         cout << "Name: " << name << endl;
8     }
9 };
10
11 class Teacher : public Person {
12 public:
13     void teach() {
14         cout << name << " is teaching." << endl;
15     }
16 };
17
18 class Principal : public Teacher {
19 public:
20     void manage() {
21         cout << name << " is managing the school." << endl;
22     }
23 };
24 int main() {
25     Principal principal;
26     principal.name = " Mrs. Arti";
27
28     principal.display();
29     principal.teach();
30     principal.manage();
31
32     return 0;
33 }
```

**Output:**

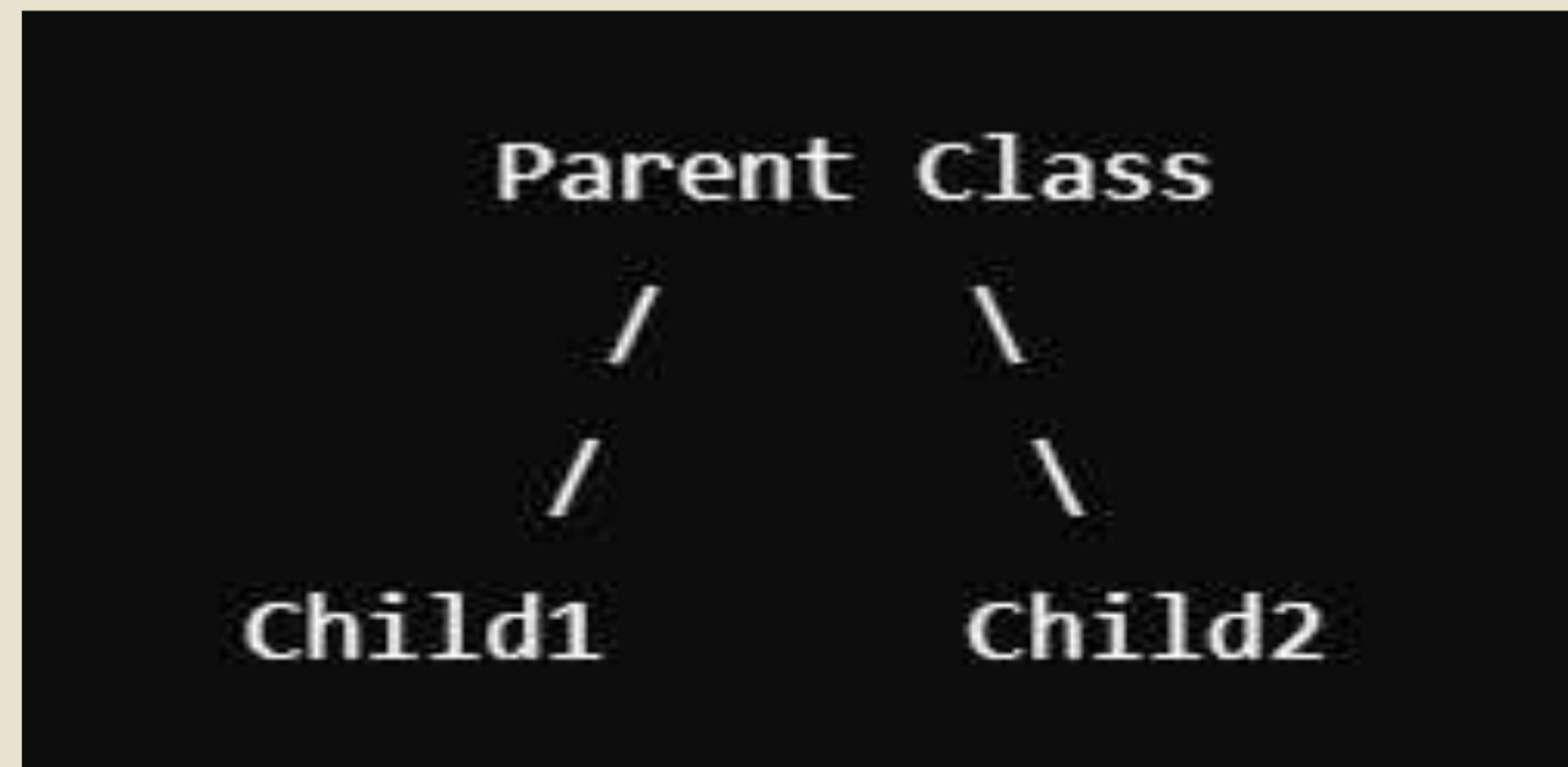
```
Name: Mrs. Arti
Mrs. Arti is teaching.
Mrs. Arti is managing the school.

==== Code Execution Successful ===
```

**GROUP 5**

## 4.HIERARCHICAL INHERITANCE

- Multiple classes inherit from the same parent class.



GROUP 5

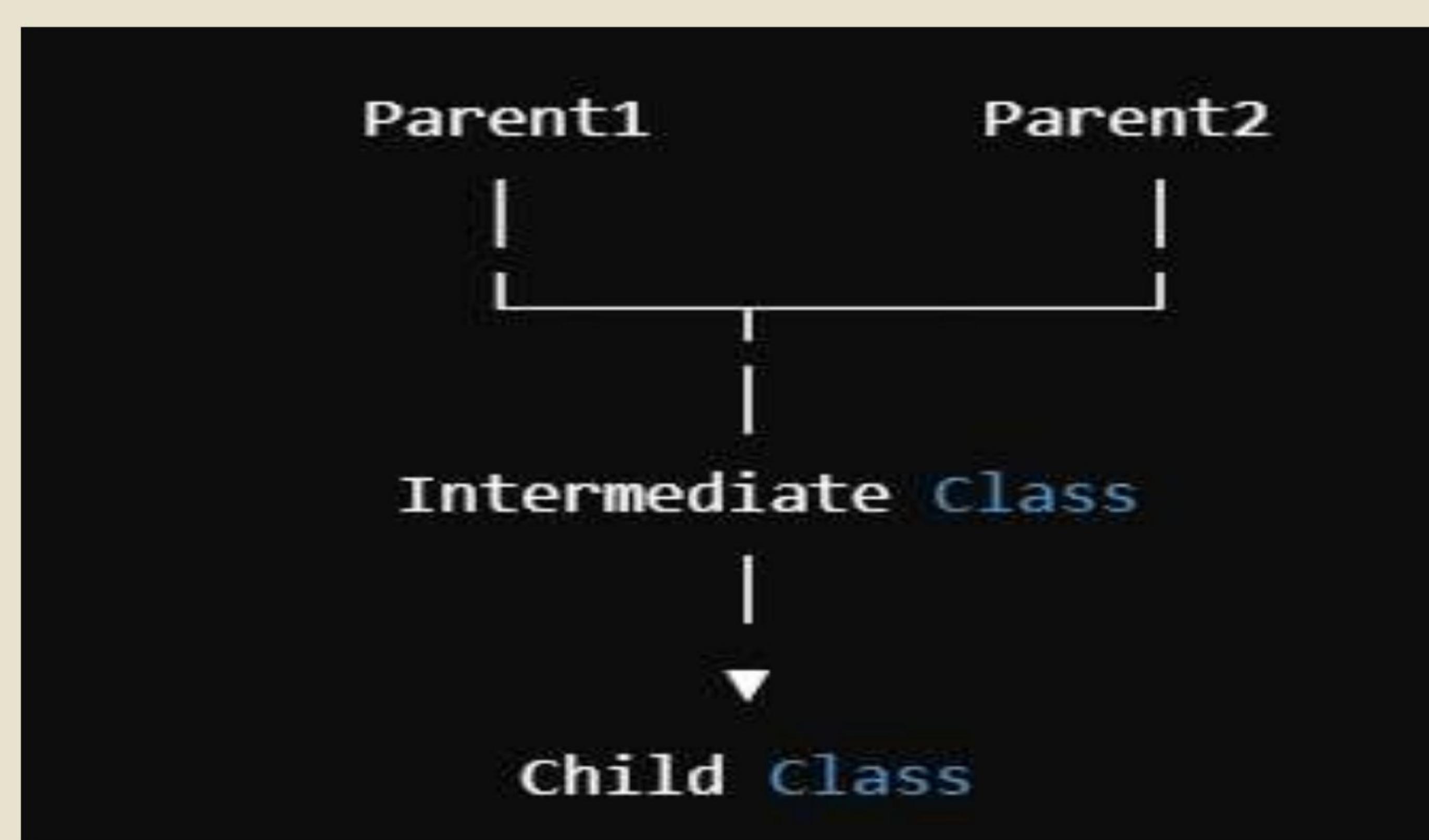
## CODE:

main.cpp	   Share	Run	Output
<pre>1 #include &lt;iostream&gt; 2 using namespace std; 3 class Person { 4 public: 5     string name; 6     void display() { 7         cout &lt;&lt; "Name: " &lt;&lt; name &lt;&lt; endl; 8     } 9 }; 10 class Student : public Person { 11 public: 12     int grade; 13     void showGrade() { 14         cout &lt;&lt; "Grade: " &lt;&lt; grade &lt;&lt; endl; 15     } 16 }; 17 class Teacher : public Person { 18 public: 19     string subject; 20     void showSubject() { 21         cout &lt;&lt; "Subject: " &lt;&lt; subject &lt;&lt; endl; 22     } 23 }; 24 int main() { 25     Student student; 26     student.name = "Ishika"; 27     student.grade = 12; 28     student.display(); 29     student.showGrade(); 30 31     Teacher teacher; 32     teacher.name = "Mr. Akash"; 33     teacher.subject = "Maths"; 34     teacher.display(); 35     teacher.showSubject(); 36 37     return 0; }</pre>	Name: Ishika Grade: 12 Name: Mr. Akash Subject: Maths  == Code Execution Successful ==		

GROUP 5

## 5.HYBRID INHERITANCE

- A combination of two or more types of inheritance. Often includes multiple and hierarchical inheritance.



GROUP 5

## CODE:

main.cpp

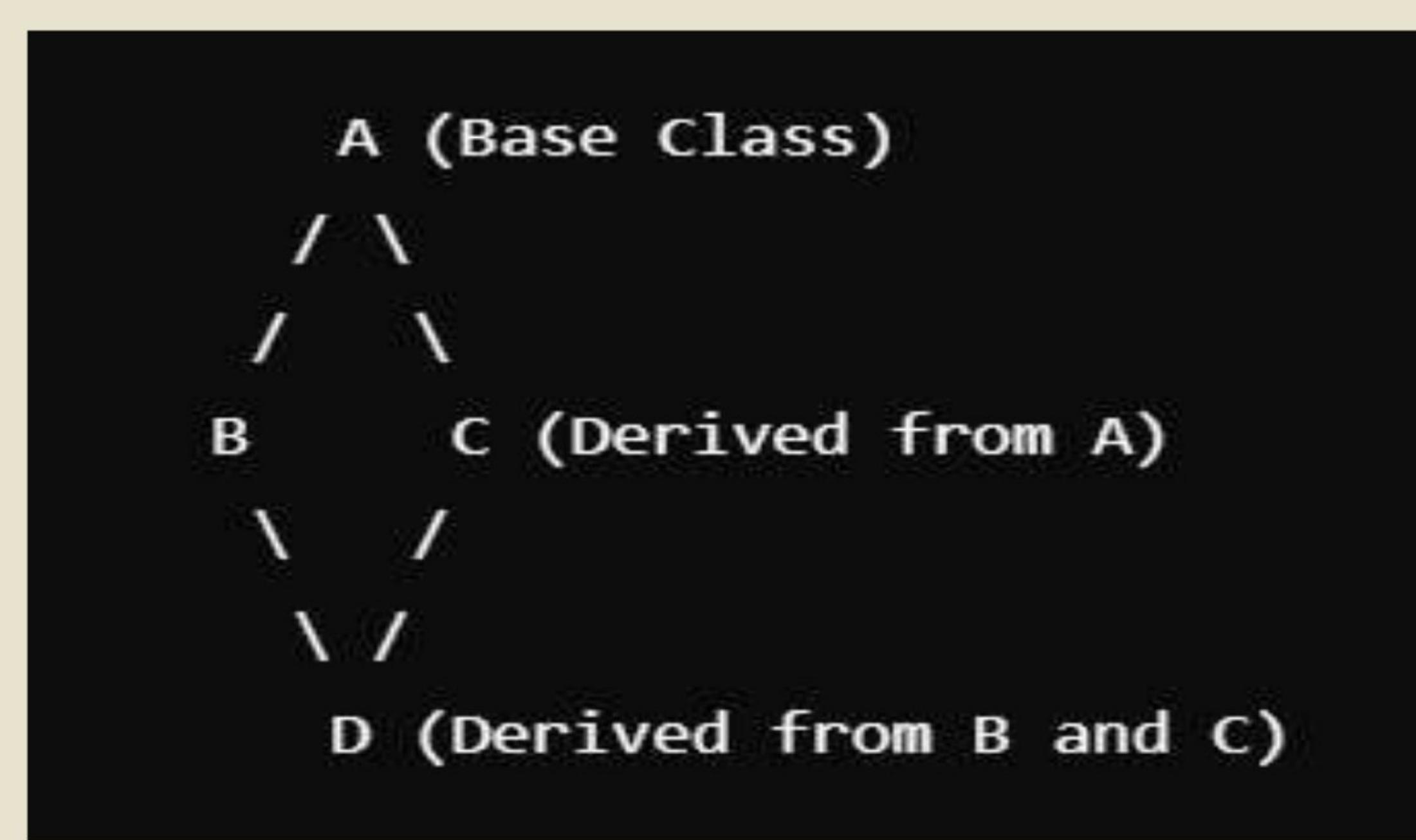
Run	Output
	Name: Dr. Mahesh Staff ID: 101 Course Managed: School Administration Dr. Mahesh (Staff ID: 101) manages the School Administration course. ==== Code Execution Successful ===

```
1 #include <iostream>
2 using namespace std;
3 class Person {
4 public:
5     string name;
6     void display() {
7         cout << "Name: " << name << endl;
8     }
9 class Staff {
10 public:
11     int staffID;
12     void showID() {
13         cout << "Staff ID: " << staffID << endl;
14     }
15 class Course {
16 public:
17     string courseName;
18     void showCourse() {
19         cout << "Course Managed: " << courseName << endl;
20     }
21 };
22 class Principal : public Person, public Staff, public Course {
23 public:
24     void manage() {
25         cout << name << " (Staff ID: " << staffID << ") manages the " << courseName << " course." << endl;
26     }
27 };
28 int main() {
29     Principal principal;
30     principal.name = "Dr. Mahesh";
31     principal.staffID = 101;
32     principal.courseName = "School Administration";
33     principal.display();
34     principal.showID();
35     principal.showCourse();
36     principal.manage();
37     return 0;
}
```

GROUP 5

## 6. MULTIPATH INHERITANCE

- Multipath inheritance occurs when a class is derived from two or more classes that have a common base class. This can lead to the **diamond problem**, where the derived class inherits multiple copies of the base class attributes and methods.



GROUP 5

## CODE:

### Without virtual inheritance

main.cpp	   Share	<b>Run</b>	Output
1 #include <iostream> 2 using namespace std; 3 4 class A { 5 public: 6 void show() { 7 cout << "Class A" << endl; 8 } 9 }; 10 11 class B : public A { 12 }; 13 14 class C : public A { 15 }; 16 17 class D : public B, public C { 18 }; 19 20 int main() { 21 D obj; 22 // Error: Ambiguous because of two 'A' copies in 'D' 23 obj.B::show(); 24 obj.C::show(); 25 26 return 0; 27 } 28	Class A Class A  ==== Code Execution Successful ===		

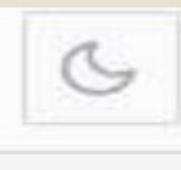
GROUP 5

## VIRTUAL BASE CLASS

- C++ provides a virtual base class, so only one copy of the common base class is inherited in a complicated inheritance hierarchy, which occurs very much in cases like multipath inheritance (commonly the diamond problem).
- **Why Use Virtual Base Classes?**
  1. **Avoids Redundant Data:** Avoids multiple copies of the same base class when the different derived classes have a common base.
  2. This means that it eliminates ambiguity when accessing members of a common base class.
  3. **Memory Efficiency:** This implementation reduces memory overhead by keeping only one copy of the base class.
- **Syntax:-** class Derived : virtual public Base {};

**GROUP 5**

## CODE:-

main.cpp	   Share	<b>Run</b>	Output
1 #include <iostream> 2 using namespace std; 3 4 class A { 5 public: 6     int data; 7     A() { data = 10; } 8     void show() { 9         cout << "Class A Data: " << data << endl; 10    } 11 }; 12 13 class B : virtual public A { 14 }; 15 16 class C : virtual public A { 17 }; 18 19 class D : public B, public C {   20 }; 21 22 int main() { 23     D obj; 24     obj.show(); 25     obj.data = 20; 26     obj.show(); 27 28     return 0; 29 }	Class A Data: 10 Class A Data: 20  ==== Code Execution Successful		

GROUP 5

## FUNCTION OVERRIDING

- Function overriding occurs when a derived class provides a specific implementation of a function that is already defined in its base class. The overridden function in the derived class must have the same signature, or name and parameters, as the one in the base class.
- The function in the derived class should have the same name, return type and parameters as the base class.
- The function is inherited by the derived class from the base class.
- Virtual Keyword is used.
- Using virtual in the base class ensures proper runtime polymorphism.
- The override specifier, in C++ and later, guarantees that such a function is actually an override of a base class method.

GROUP 5

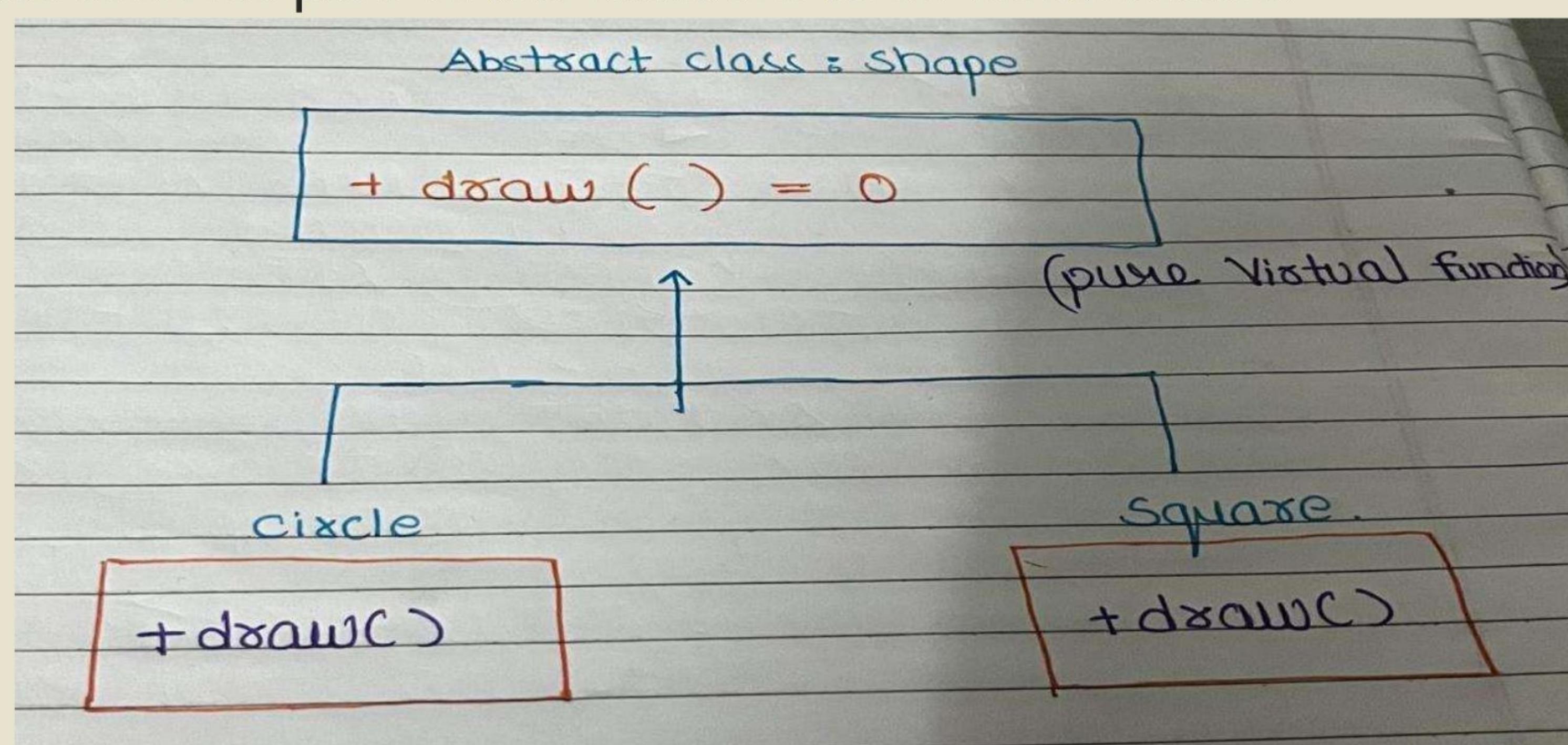
## CODE:-

main.cpp				Run	Output
1 #include <iostream> 2 using namespace std; 3 4 class Base { 5 public: 6     virtual void show() { 7         cout << "Base class show() method" << endl; 8     } 9 }; 10 11 class Derived : public Base { 12 public: 13     void show() override { 14         cout << "Derived class show() method" << endl; 15     } 16 }; 17 18 int main() { 19     Base* basePtr; 20     Derived obj; 21     basePtr = &obj; 22 23     basePtr->show(); 24 25     return 0; 26 } 27	Derived class show() method ==== Code Execution Successful ===				

GROUP 5

## ABSTRACT CLASS AND PURE VIRTUAL CLASS

- An abstract class is a blueprint for derived classes. It cannot be instantiated directly.
- It usually consists of at least one pure virtual function.
- A pure virtual function is a function that has no implementation in the base class and defined by assigning 0 to it.
- Abstract Class: It cannot be instantiated directly.
- Pure Virtual Function: Declared with = 0 and supposed to be implemented by the derived classes.
- Derived Classes: Should override pure virtual functions to be constructible.



GROUP 5

## CODE EXPLANATION:-

The screenshot shows a code editor interface with a tab labeled "main.cpp". The code itself is as follows:

```
1 #include <iostream>
2 using namespace std;
3 // abstract class
4 class Shape {
5 public:
6     virtual void draw() = 0;
7 };
8 // Derived class 1: Circle
9 class Circle : public Shape {
10 public:
11     void draw() override {
12         cout << "Drawing a Circle." << endl;
13     }
14 };
15 // Derived class 2: Square
16 class Square : public Shape {
17 public:
18     void draw() override {
19         cout << "Drawing a Square." << endl;
20     }
21 };
22 int main() {
23     Circle circle;
24     Square square;
25     Shape* shapePtr; // Pointer to abstract class
26     shapePtr = &circle;
27     shapePtr->draw(); // Output: Drawing a Circle.
28     shapePtr = &square;
29     shapePtr->draw(); // Output: Drawing a Square.
30 }
```

The code defines an abstract class `Shape` with a pure virtual function `draw()`. It then defines two derived classes, `Circle` and `Square`, which inherit from `Shape` and implement the `draw()` function. In the `main()` function, objects of both `Circle` and `Square` are created, and a pointer to `Shape` is used to call the `draw()` function on each object. The output of the program is "Drawing a Circle." followed by "Drawing a Square.", demonstrating the concept of polymorphism.

GROUP 5