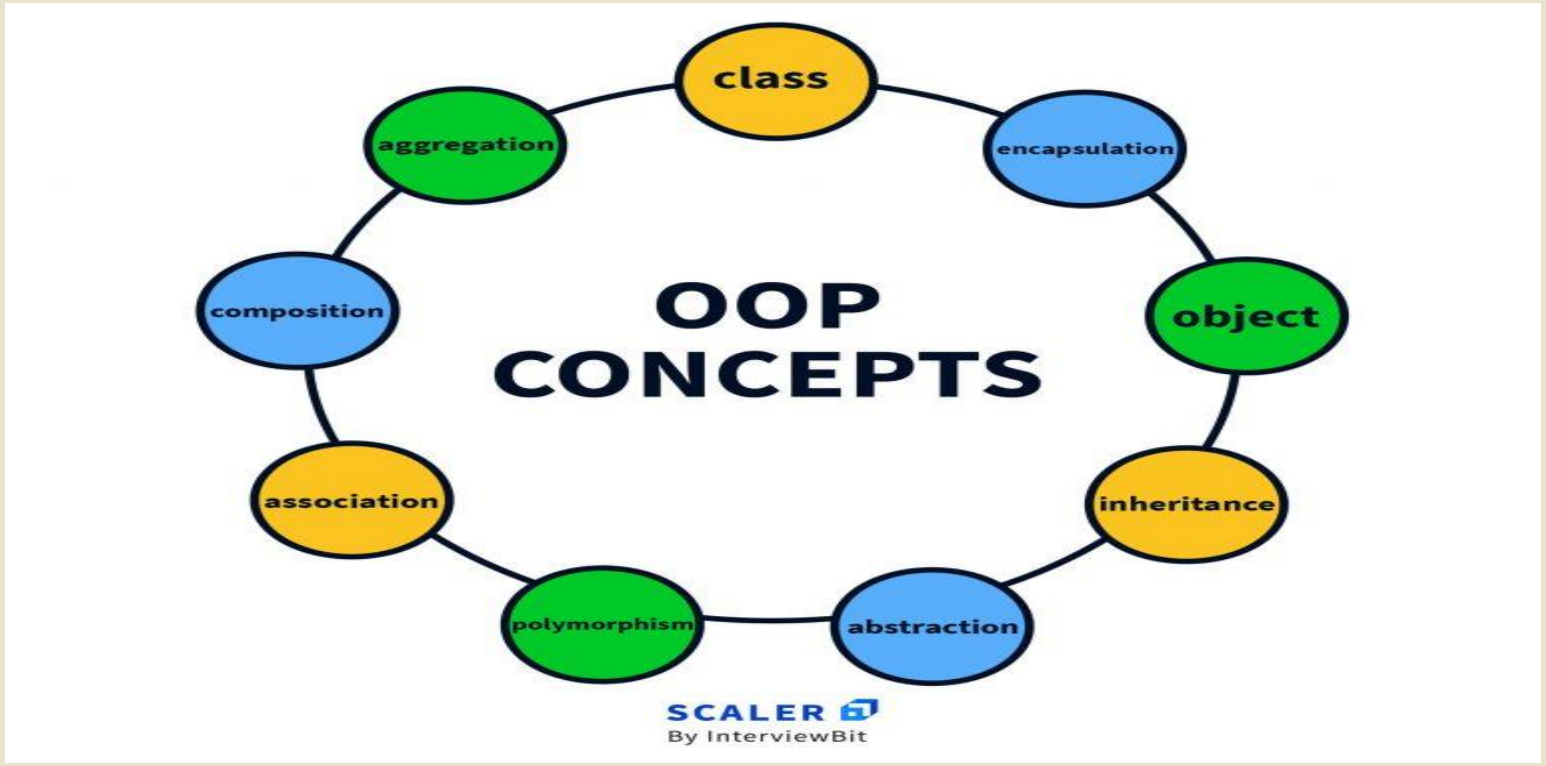
UNIT 3



Submitted By:-

Name:- Ishika Jain

Reg. no.:- PCE23CS066

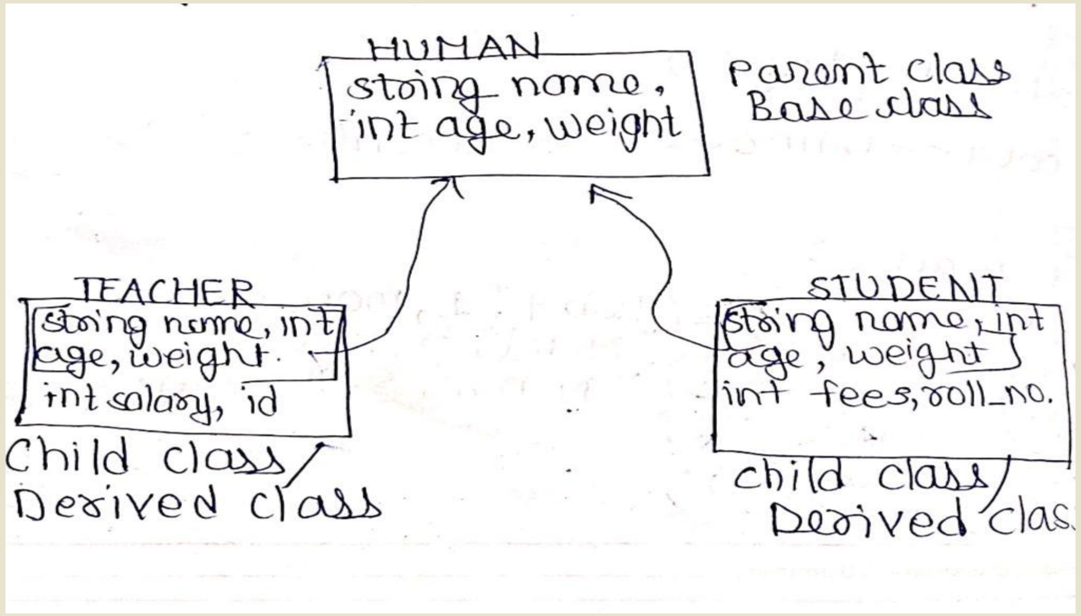
GROUP 5

TOPIC OVERVIEW

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

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.

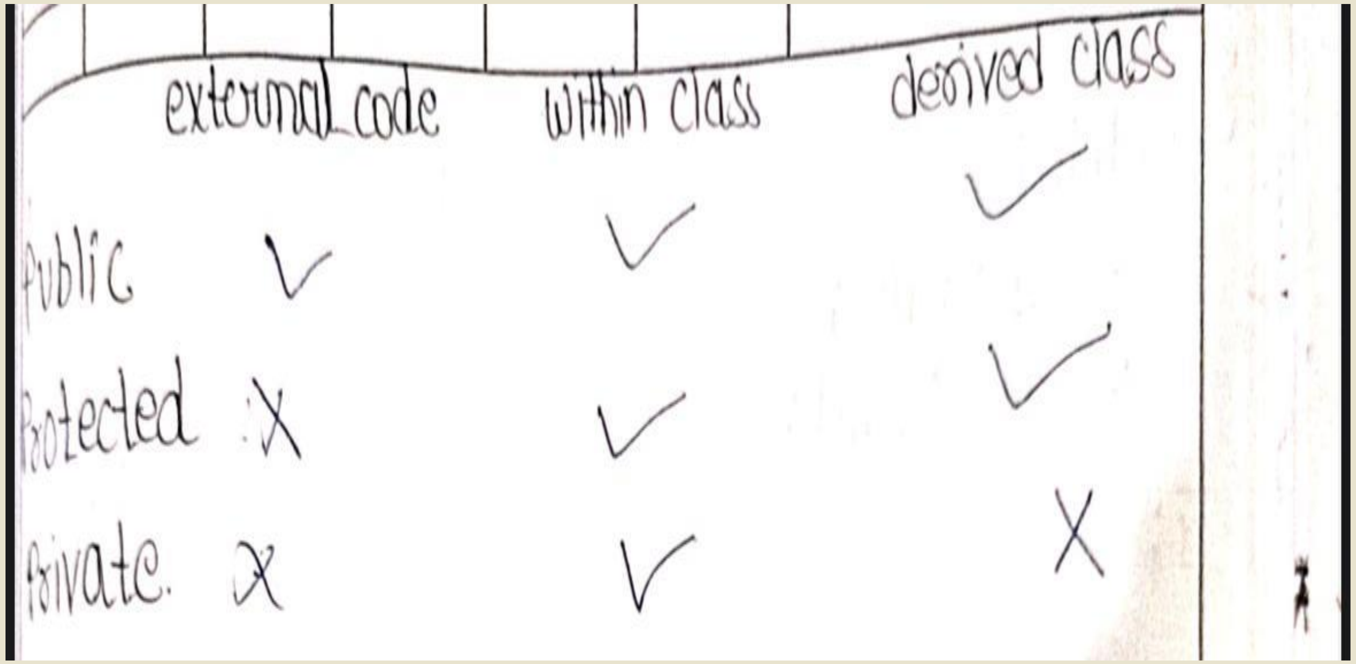


GROUP 5

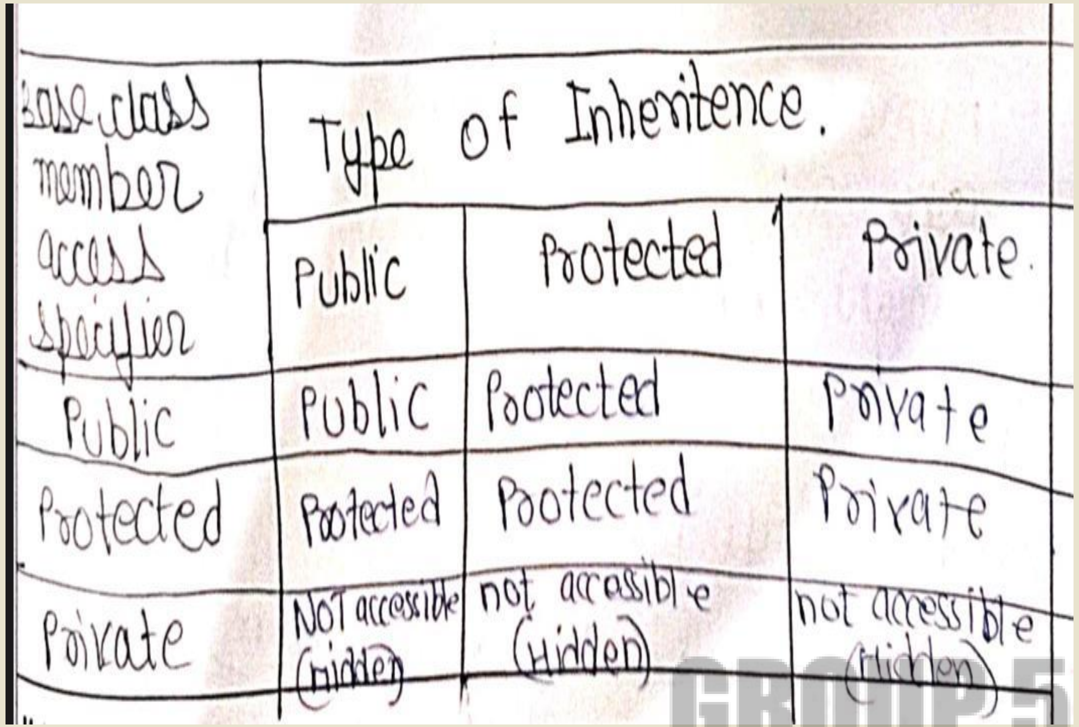
ACCESS MODIFIER

- There are three access modifier:-

1. Private
2. Public
3. Protected



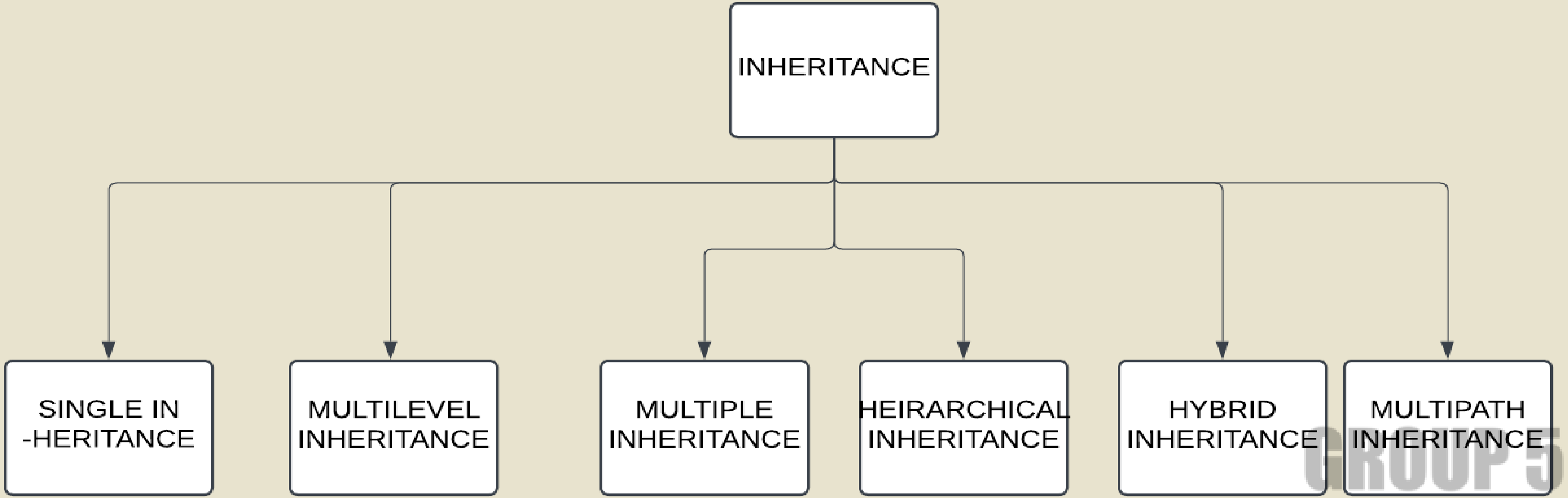
	external code	within class	derived class
Public	✓	✓	✓
Protected	X	✓	✓
Private.	X	✓	X



Base class member access specifier	Type of Inheritence.		
	Public	Protected	Private.
Public	Public	Protected	Private
Protected	Protected	Protected	Private
Private	Not accessible (hidden)	not accessible (Hidden)	not accessible (Hidden)

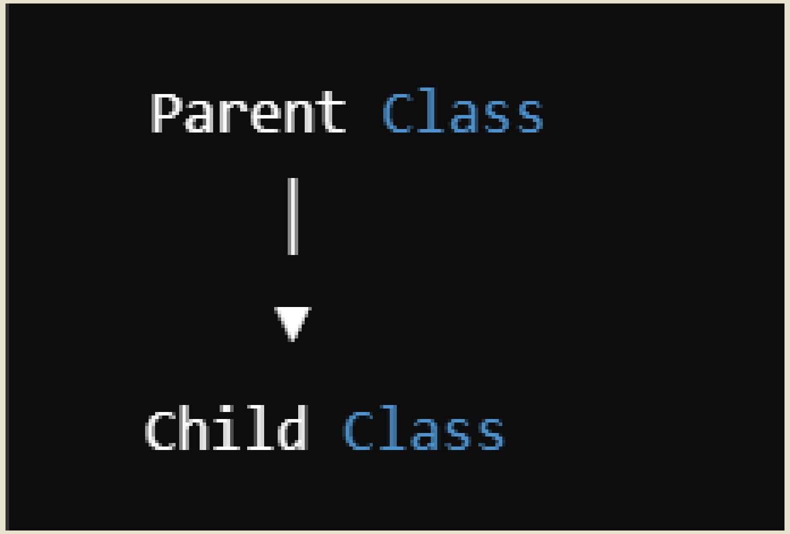
TYPES OF INHERITANCE

- THERE ARE 6 TYPES OF INHERITACE



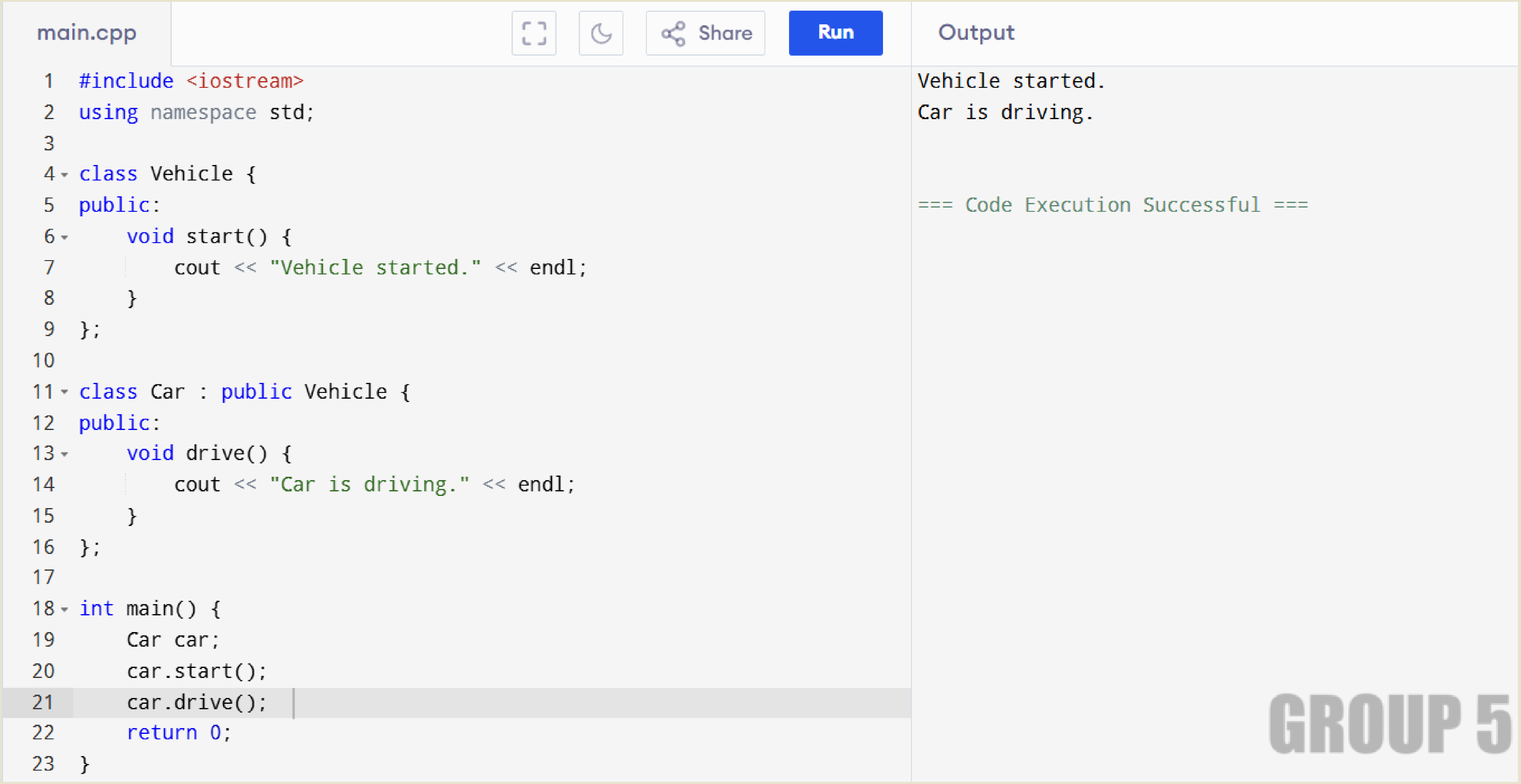
1.SINGLE INHERITANCE

- A subclass inherits from only one parent class.



```
Parent Class
     |
     ▼
Child Class
```

CODE:



```cpp
#include <iostream>
using namespace std;

class Vehicle {
public:
    void start() {
        cout << "Vehicle started." << endl;
    }
};

class Car : public Vehicle {
public:
    void drive() {
        cout << "Car is driving." << endl;
    }
};

int main() {
    Car car;
    car.start();
    car.drive();
    return 0;
}
```

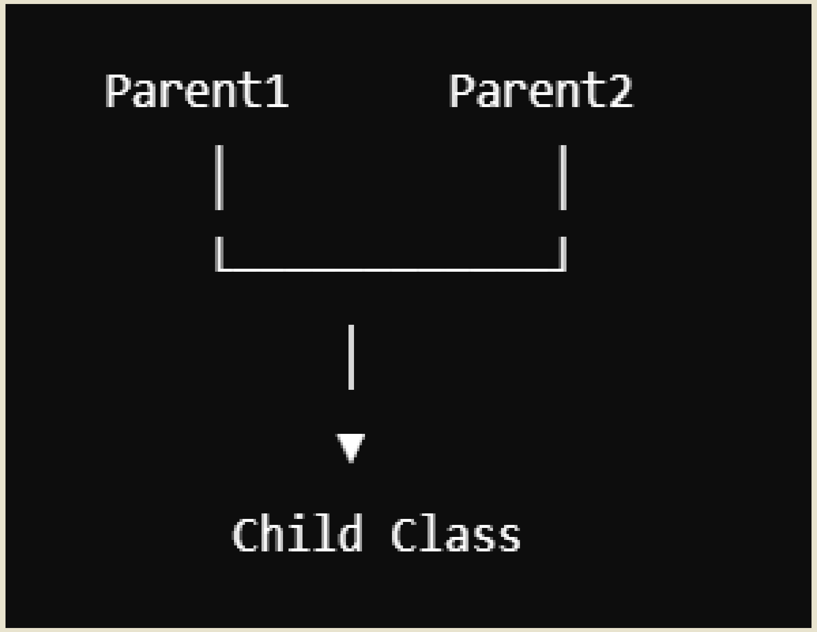
```
Vehicle started.
Car is driving.


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

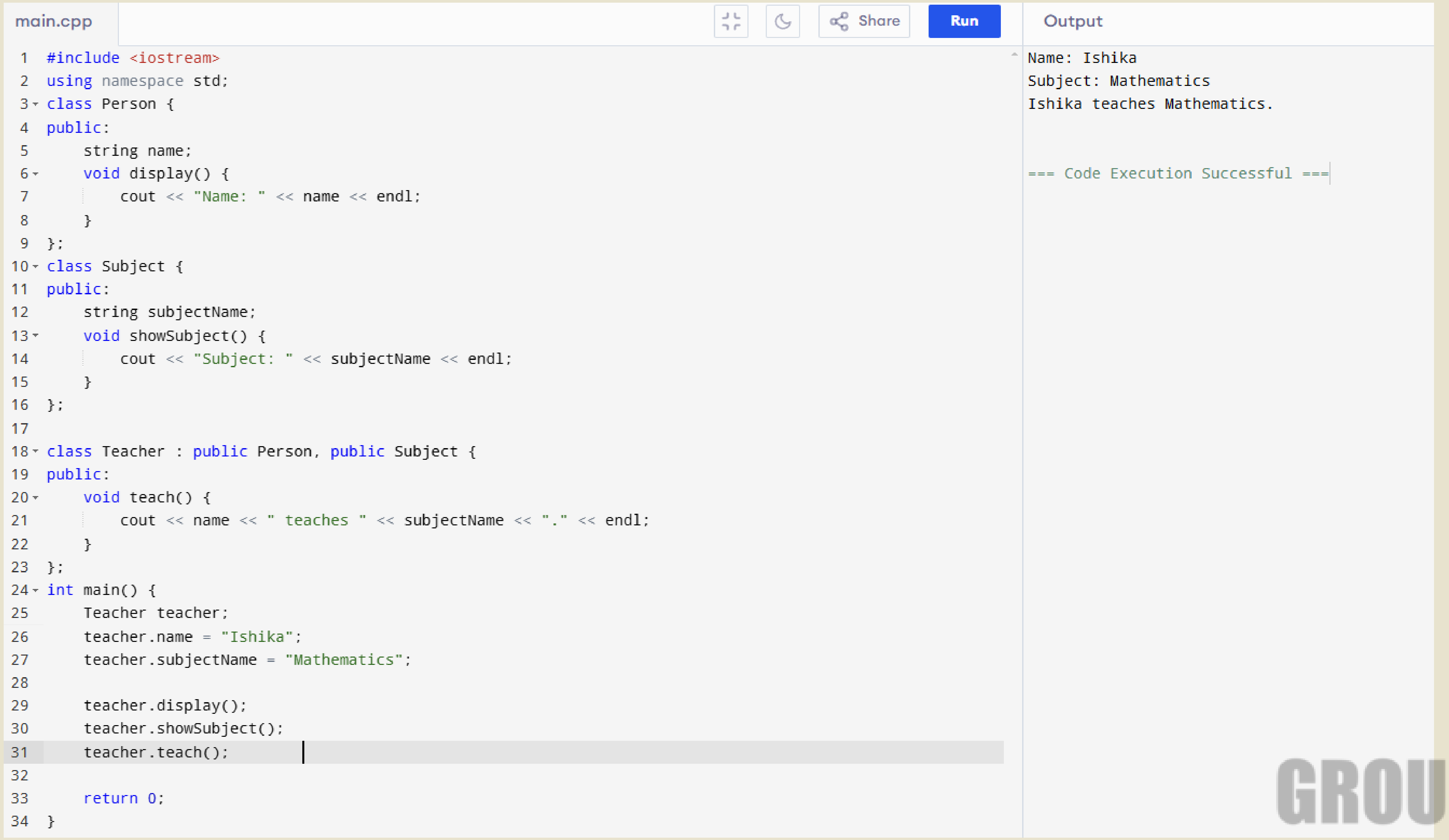
GROUP 5

2.MULTIPLE INHERITANCE

- A subclass inherits from multiple parent classes.



CODE:



```cpp
#include <iostream>
using namespace std;
class Person {
public:
    string name;
    void display() {
        cout << "Name: " << name << endl;
    }
};
class Subject {
public:
    string subjectName;
    void showSubject() {
        cout << "Subject: " << subjectName << endl;
    }
};

class Teacher : public Person, public Subject {
public:
    void teach() {
        cout << name << " teaches " << subjectName << "." << endl;
    }
};
int main() {
    Teacher teacher;
    teacher.name = "Ishika";
    teacher.subjectName = "Mathematics";

    teacher.display();
    teacher.showSubject();
    teacher.teach();

    return 0;
}
```

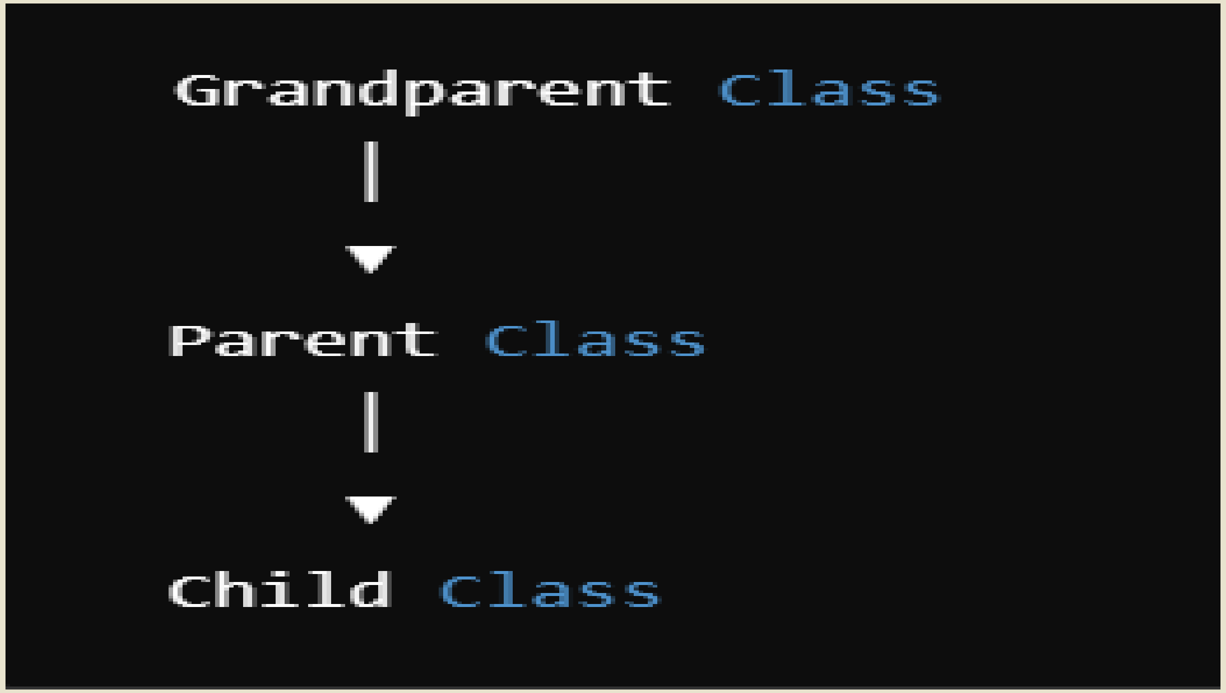
```
Name: Ishika
Subject: Mathematics
Ishika teaches Mathematics.


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

GROUP 5

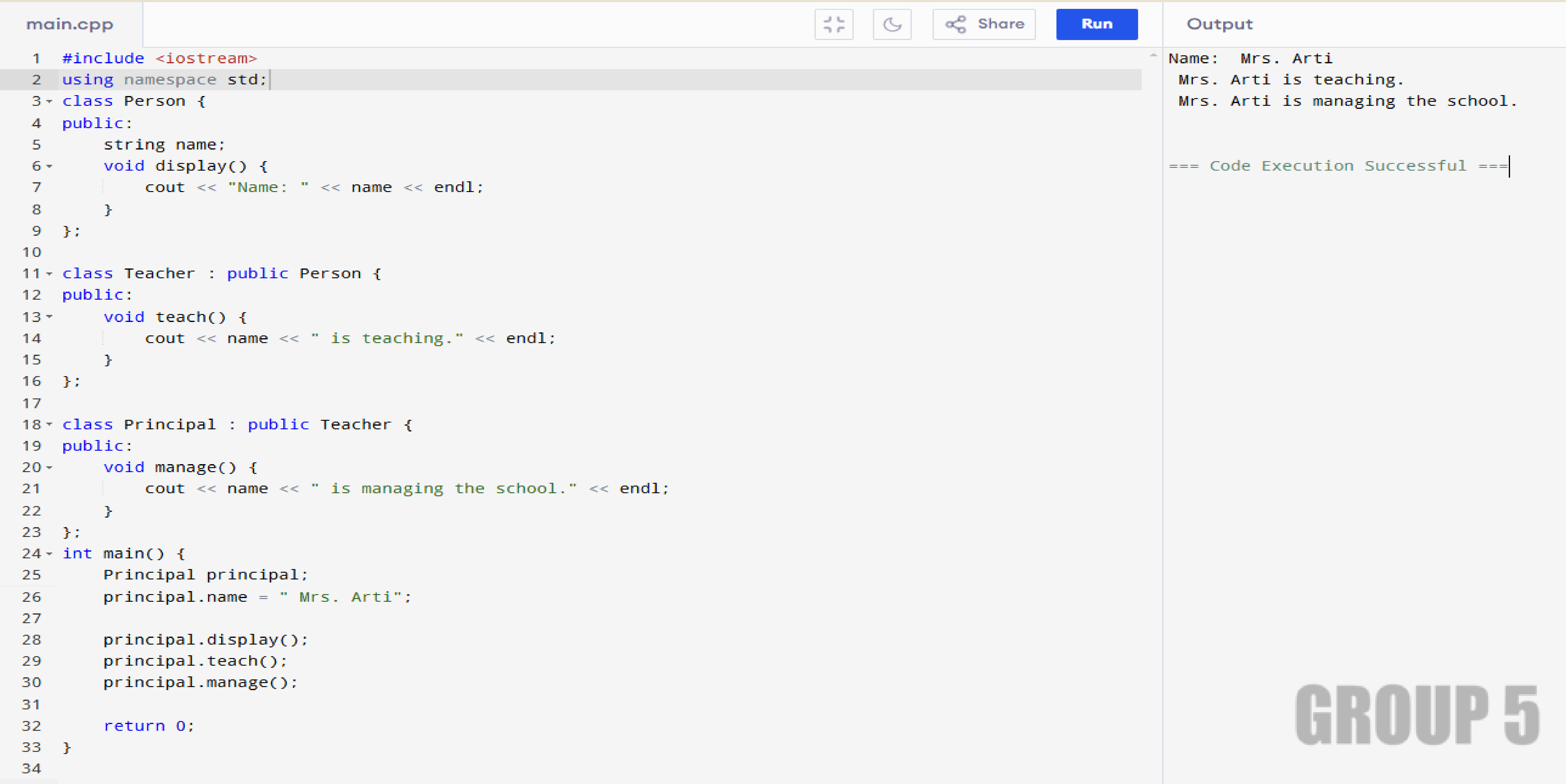
3.MULTILEVEL INHERITANCE

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


```
Grandparent Class
     |
     ▼
Parent Class
     |
     ▼
Child Class
```

GROUP 5

CODE:



```cpp
#include <iostream>
using namespace std;
class Person {
public:
    string name;
    void display() {
        cout << "Name: " << name << endl;
    }
};

class Teacher : public Person {
public:
    void teach() {
        cout << name << " is teaching." << endl;
    }
};

class Principal : public Teacher {
public:
    void manage() {
        cout << name << " is managing the school." << endl;
    }
};
int main() {
    Principal principal;
    principal.name = " Mrs. Arti";

    principal.display();
    principal.teach();
    principal.manage();

    return 0;
}

```

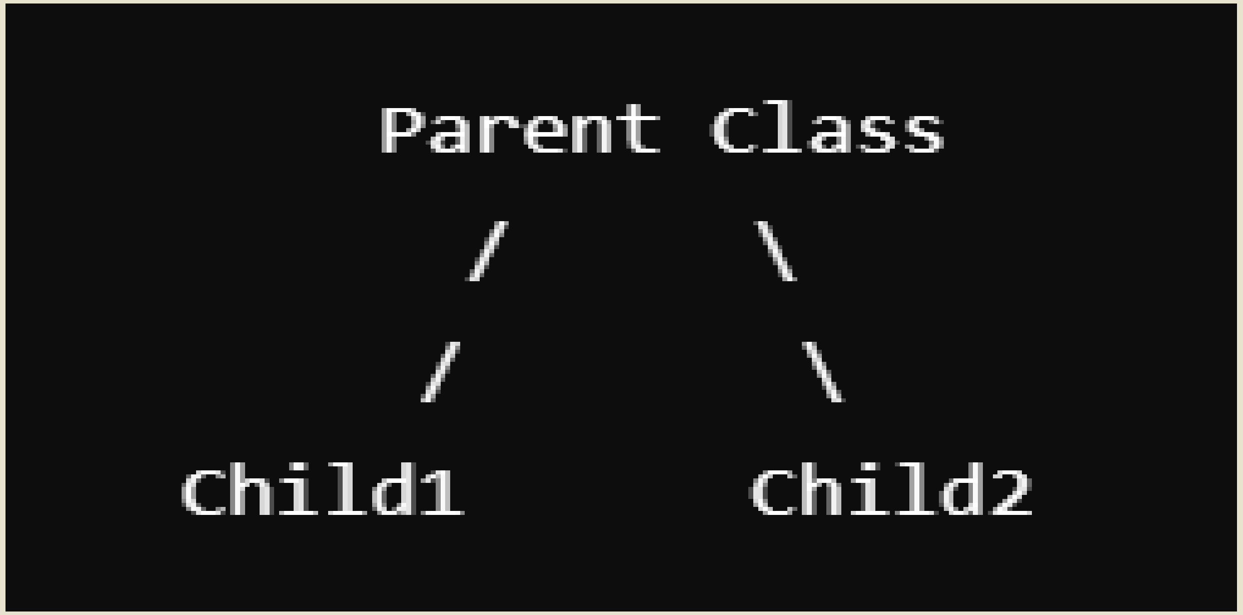
```
Name:  Mrs. Arti
 Mrs. Arti is teaching.
 Mrs. Arti is managing the school.


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

GROUP 5

4.HIERARCHICAL INHERITANCE

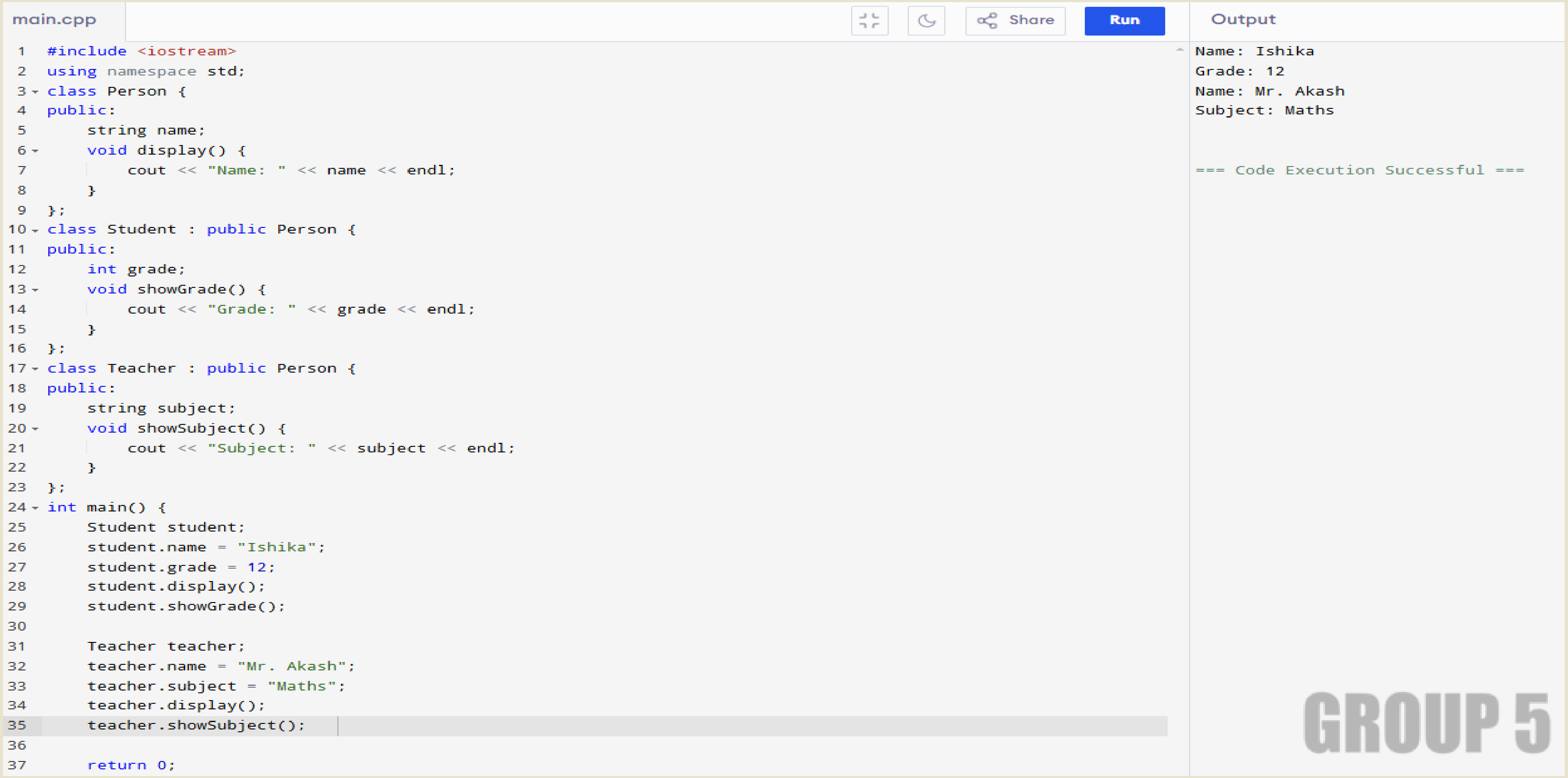
- Multiple classes inherit from the same parent class.



```
      Parent Class
        /     \
       /       \
  Child1        Child2
```

GROUP 5

CODE:



```cpp
#include <iostream>
using namespace std;
class Person {
public:
    string name;
    void display() {
        cout << "Name: " << name << endl;
    }
};
class Student : public Person {
public:
    int grade;
    void showGrade() {
        cout << "Grade: " << grade << endl;
    }
};
class Teacher : public Person {
public:
    string subject;
    void showSubject() {
        cout << "Subject: " << subject << endl;
    }
};
int main() {
    Student student;
    student.name = "Ishika";
    student.grade = 12;
    student.display();
    student.showGrade();

    Teacher teacher;
    teacher.name = "Mr. Akash";
    teacher.subject = "Maths";
    teacher.display();
    teacher.showSubject();

    return 0;
```

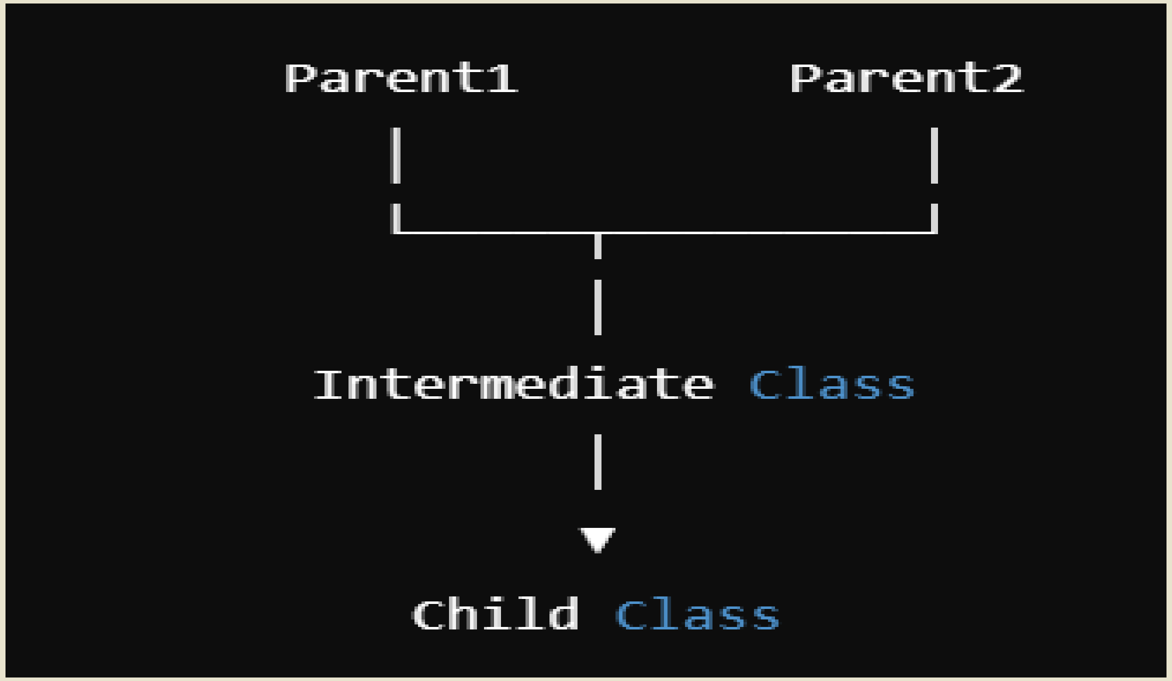
```
Name: Ishika
Grade: 12
Name: Mr. Akash
Subject: Maths


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

GROUP 5

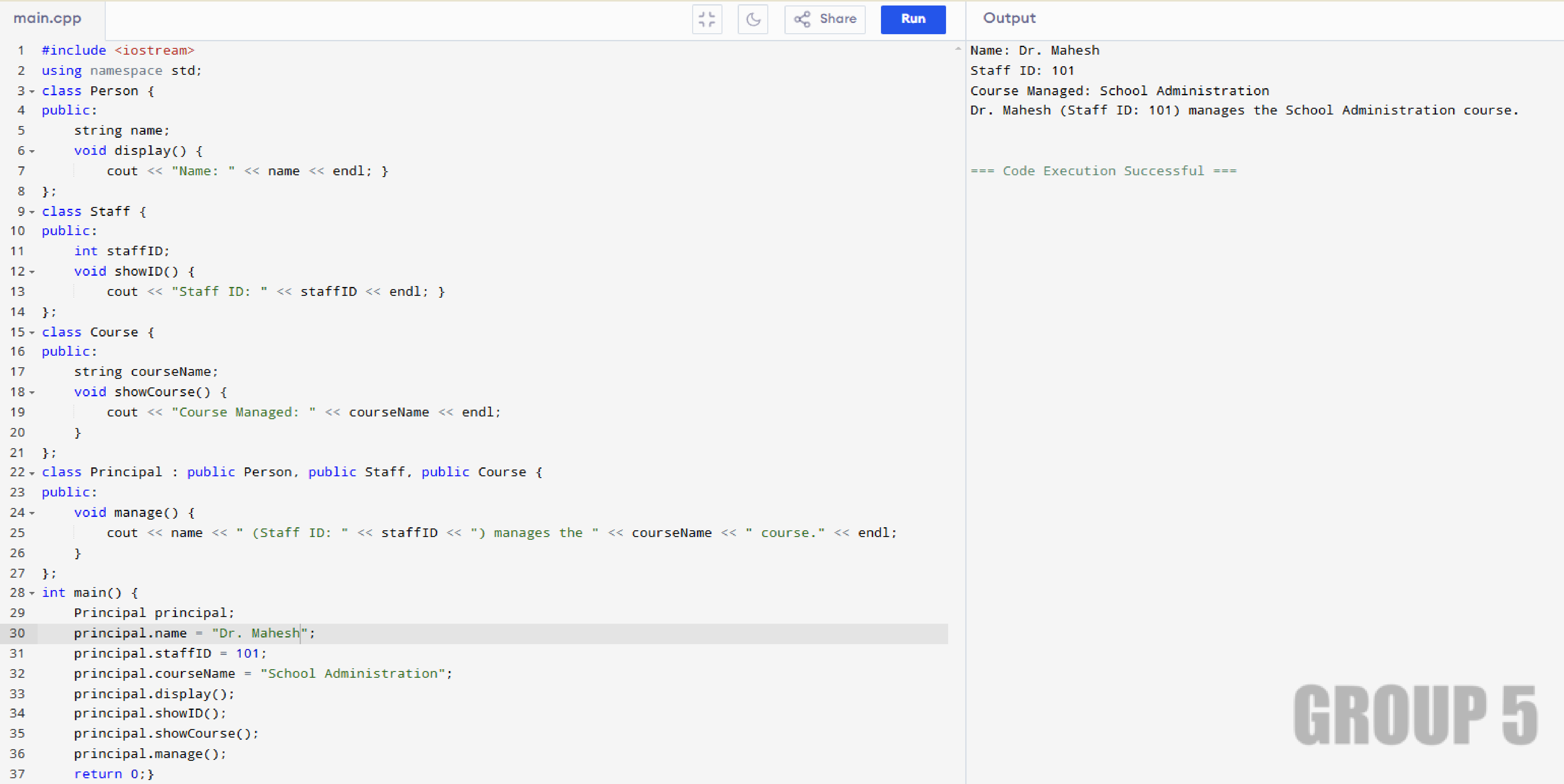
5.HYBRID INHERITANCE

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



GROUP 5

CODE:



```cpp
#include <iostream>
using namespace std;
class Person {
public:
    string name;
    void display() {
        cout << "Name: " << name << endl; }
};
class Staff {
public:
    int staffID;
    void showID() {
        cout << "Staff ID: " << staffID << endl; }
};
class Course {
public:
    string courseName;
    void showCourse() {
        cout << "Course Managed: " << courseName << endl;
    }
};
class Principal : public Person, public Staff, public Course {
public:
    void manage() {
        cout << name << " (Staff ID: " << staffID << ") manages the " << courseName << " course." << endl;
    }
};
int main() {
    Principal principal;
    principal.name = "Dr. Mahesh";
    principal.staffID = 101;
    principal.courseName = "School Administration";
    principal.display();
    principal.showID();
    principal.showCourse();
    principal.manage();
    return 0;}
```

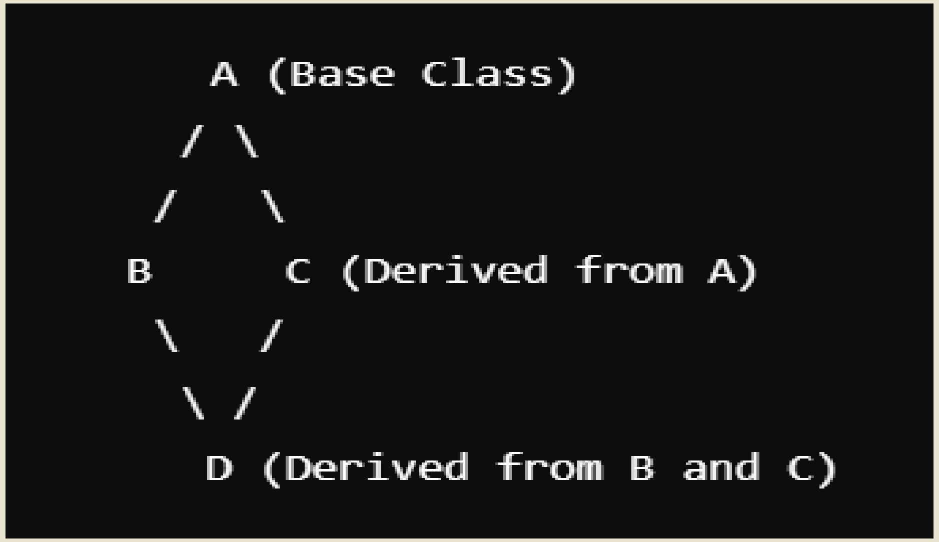
```
Name: Dr. Mahesh
Staff ID: 101
Course Managed: School Administration
Dr. Mahesh (Staff ID: 101) manages the School Administration course.


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

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.

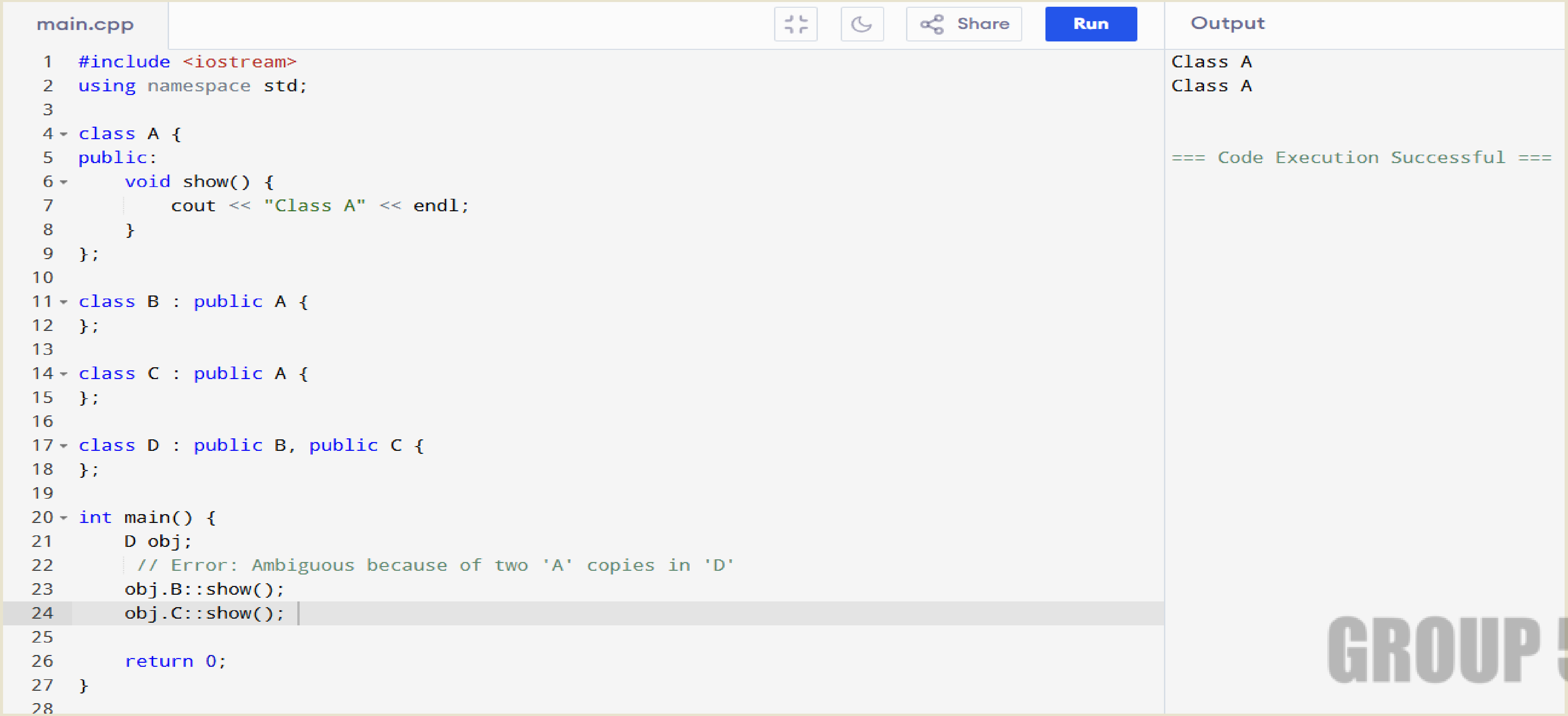


```
       A (Base Class)
      / \
     /   \
    B     C (Derived from A)
     \   /
      \ /
       D (Derived from B and C)
```

GROUP 5

CODE:

Without virtual inheritance



```cpp
#include <iostream>
using namespace std;

class A {
public:
    void show() {
        cout << "Class A" << endl;
    }
};

class B : public A {
};

class C : public A {
};

class D : public B, public C {
};

int main() {
    D obj;
     // Error: Ambiguous because of two 'A' copies in 'D'
    obj.B::show();
    obj.C::show();

    return 0;
}
```

```
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).

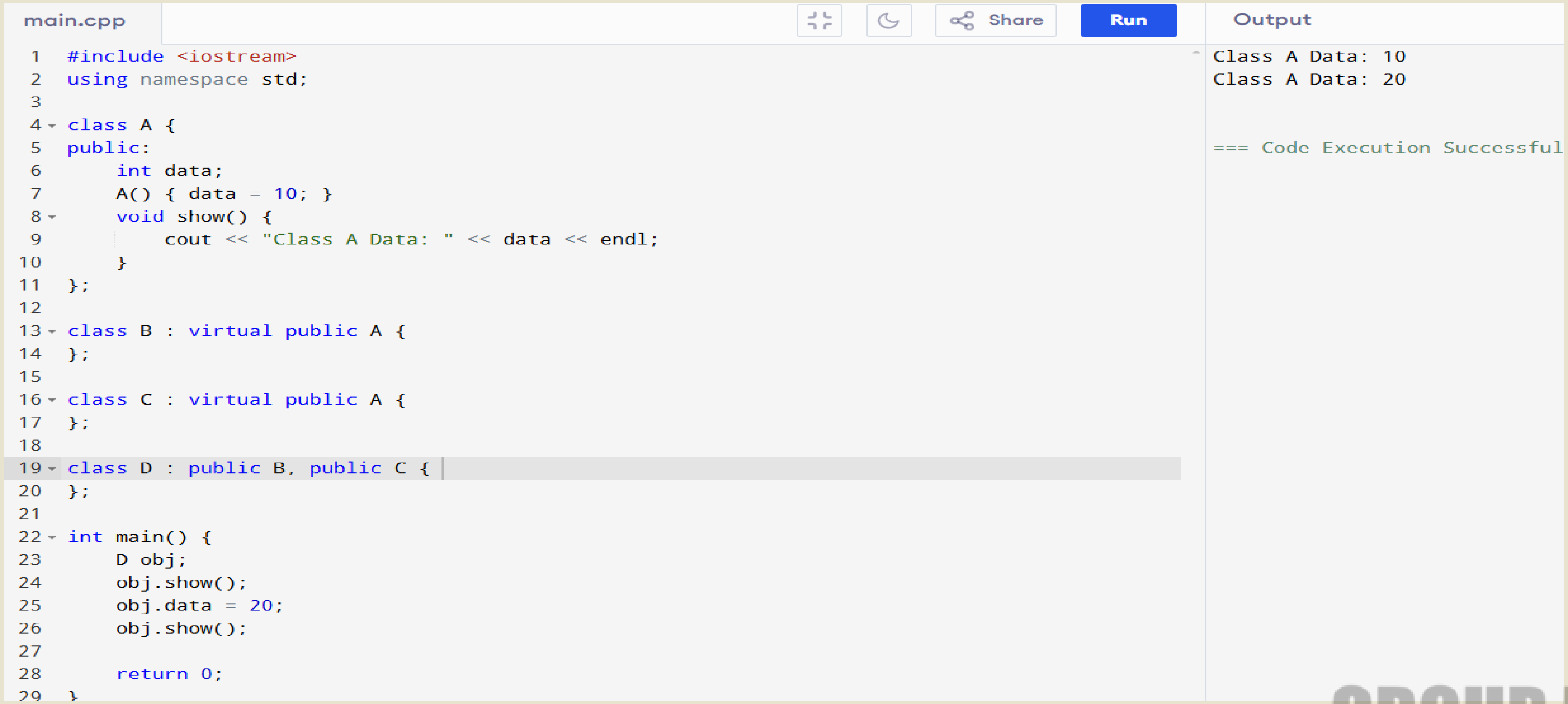
- **Why Use Virtual Base Classes?**

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

```cpp
#include <iostream>
using namespace std;

class A {
public:
    int data;
    A() { data = 10; }
    void show() {
        cout << "Class A Data: " << data << endl;
    }
};

class B : virtual public A {
};

class C : virtual public A {
};

class D : public B, public C {
};

int main() {
    D obj;
    obj.show();
    obj.data = 20;
    obj.show();

    return 0;
}
```

```
Class A Data: 10
Class A Data: 20


=== Code Execution Successful
```

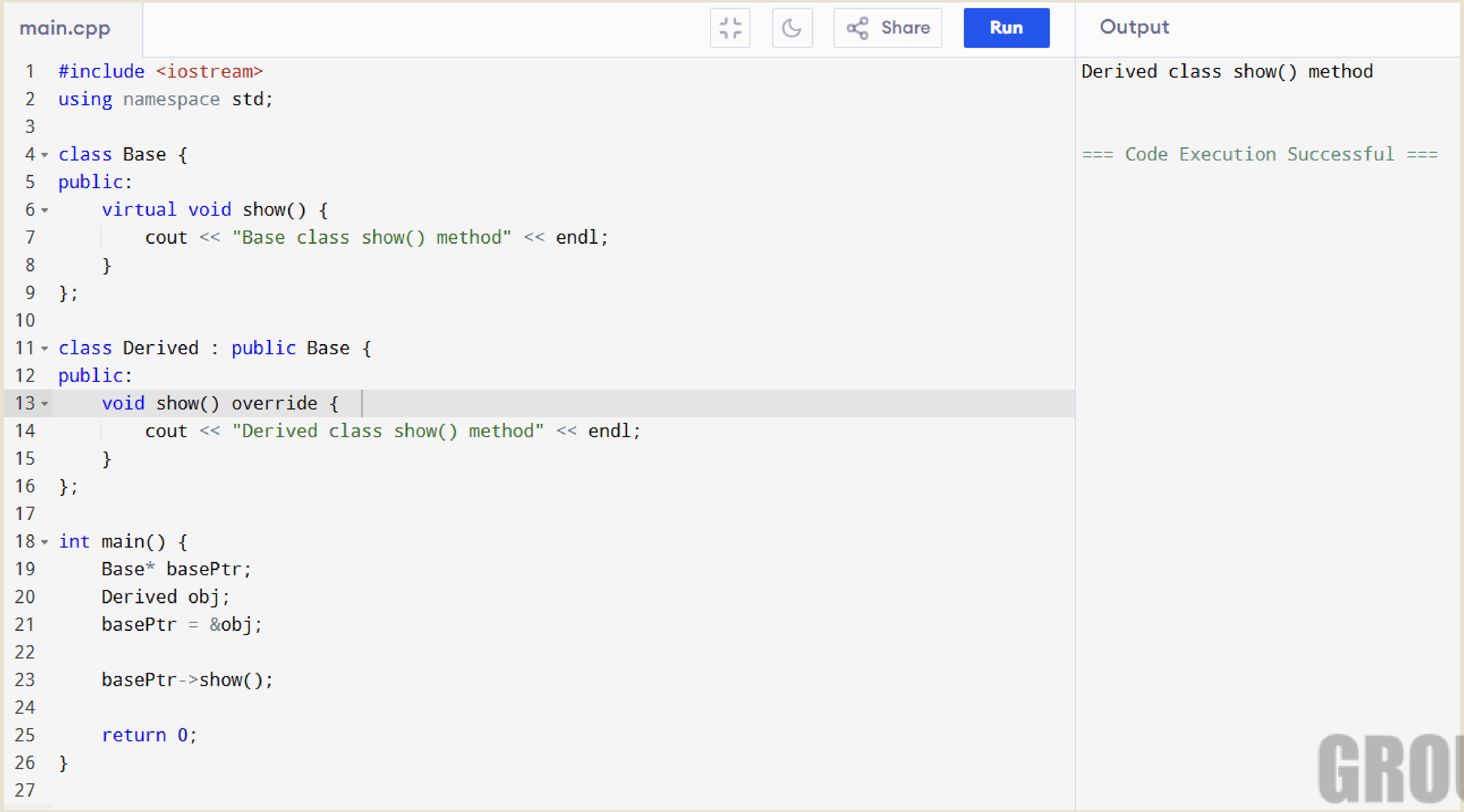
GROUP 5

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:-



```cpp
#include <iostream>
using namespace std;

class Base {
public:
    virtual void show() {
        cout << "Base class show() method" << endl;
    }
};

class Derived : public Base {
public:
    void show() override {
        cout << "Derived class show() method" << endl;
    }
};

int main() {
    Base* basePtr;
    Derived obj;
    basePtr = &obj;

    basePtr->show();

    return 0;
}
```

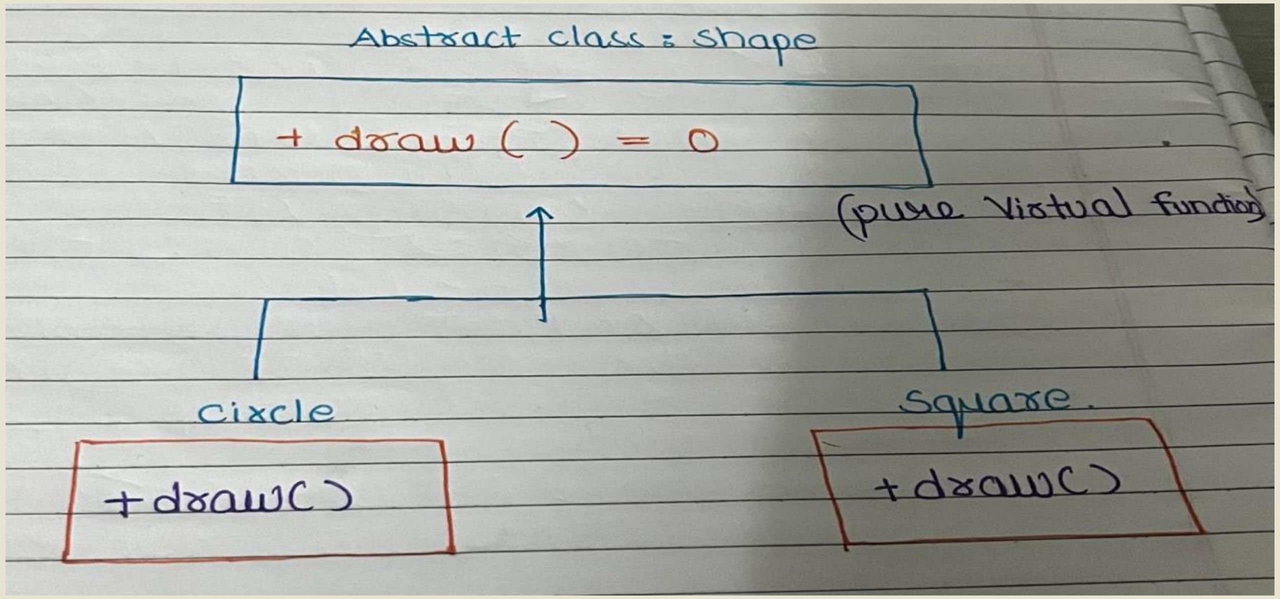
```
Derived class show() method


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

GROUP 5

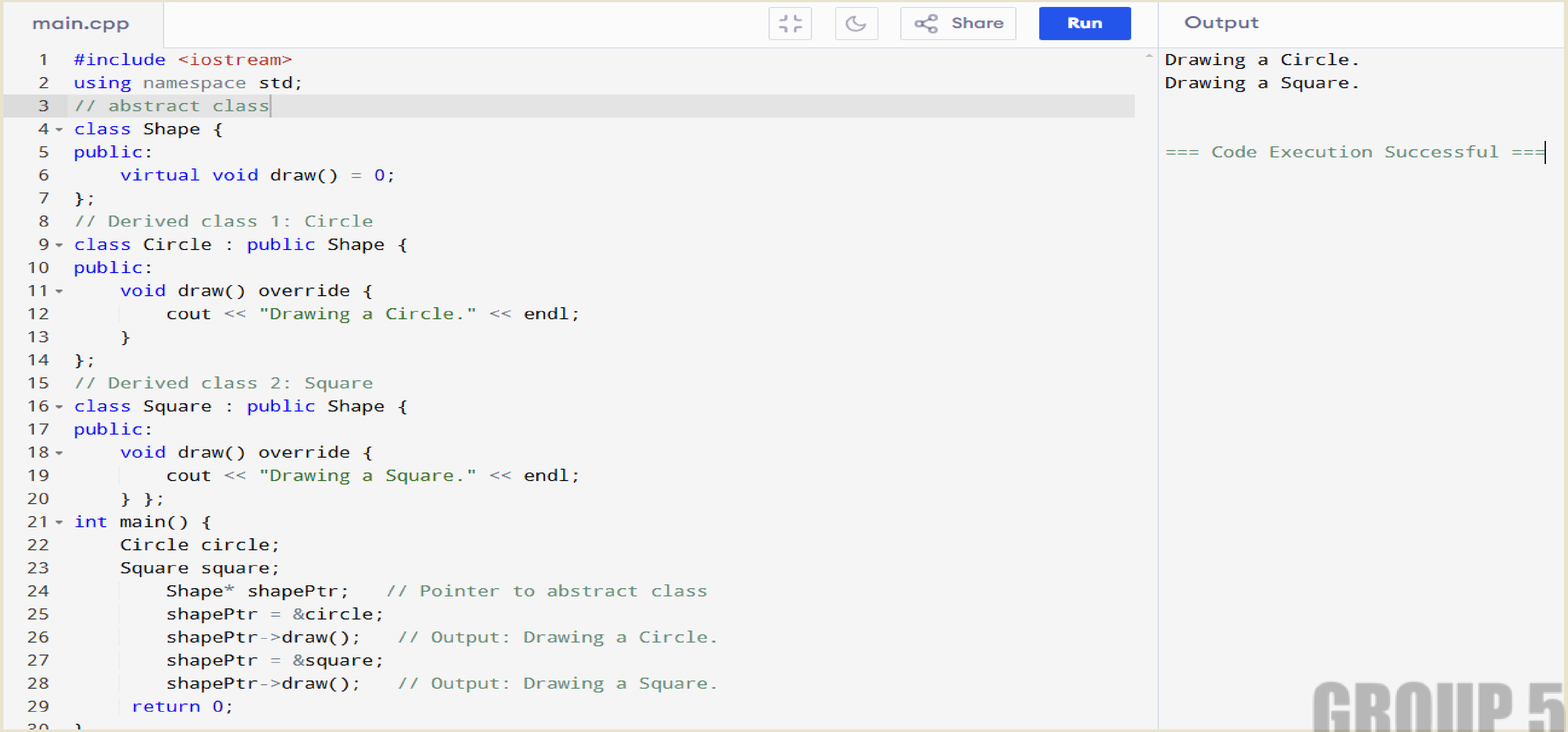
ABSTRACT CLASS AND PURE VIRTUAL CLASS

- 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:-



```cpp
#include <iostream>
using namespace std;
// abstract class
class Shape {
public:
    virtual void draw() = 0;
};
// Derived class 1: Circle
class Circle : public Shape {
public:
    void draw() override {
        cout << "Drawing a Circle." << endl;
    }
};
// Derived class 2: Square
class Square : public Shape {
public:
    void draw() override {
        cout << "Drawing a Square." << endl;
    } };
int main() {
    Circle circle;
    Square square;
        Shape* shapePtr;    // Pointer to abstract class
        shapePtr = &circle;
        shapePtr->draw();    // Output: Drawing a Circle.
        shapePtr = &square;
        shapePtr->draw();    // Output: Drawing a Square.
     return 0;
}
```

```
Drawing a Circle.
Drawing a Square.


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

GROUP 5