



Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

200
1822 - 2022

Inheritance

Man Geni Addysg Uwch yng Nghymru
Where Higher Education Began in Wales

www.uwtsd.ac.uk



Objective:

- Understand the concept of inheritance in C++.
- Learn the types of inheritance.
- scenario-based tutorials.



Recap

<https://forms.office.com/e/u7MAJwnjpX>

Man Geni Addysg Uwch yng Nghymru
Where Higher Education Began in Wales

www.uwtsd.ac.uk



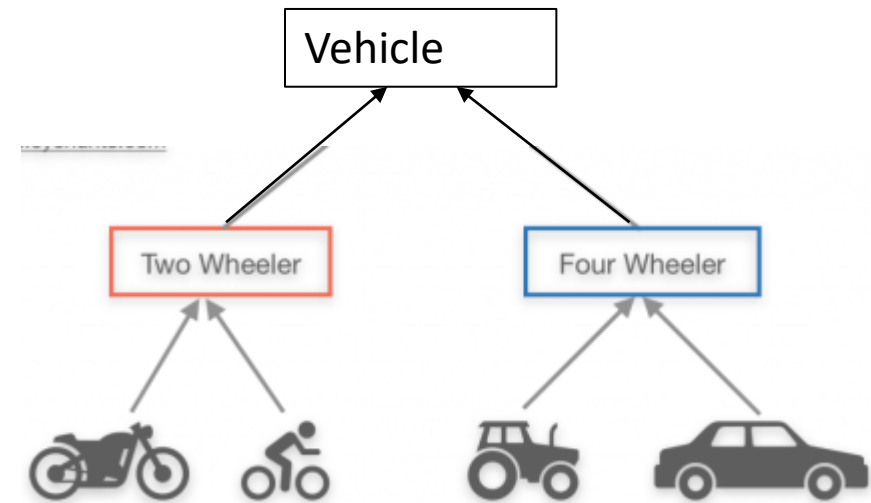
Introduction to Inheritance

- allows a new class to inherit properties and behaviours (attributes and methods) from an existing class.
- It facilitates code reuse and promotes a hierarchical relationship between classes.
- derived class (child) - the class that inherits from another class
- base class (parent) - the class being inherited from

Syntax of Inheritance:

- Use the **:** symbol after the derived class name followed by the access specifier (public, private, or protected) and the name of the base class.

```
class Base {  
    // Base class definition  
};  
  
class Derived : accessSpecifier Base {  
    // Derived class inheriting from Base  
};
```



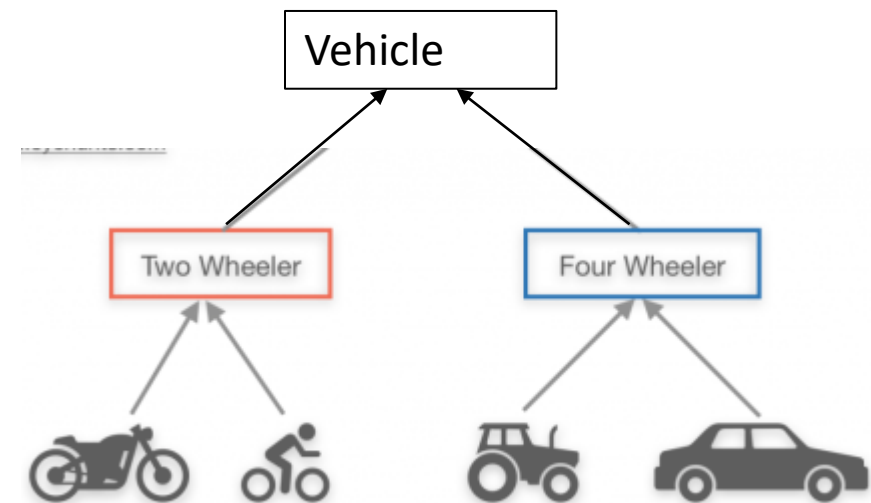
Common attributes and methods of base class

Attributes:

1. Make/Brand
2. Model
3. Year:
4. Engine Type/Size
5. Number of Wheels

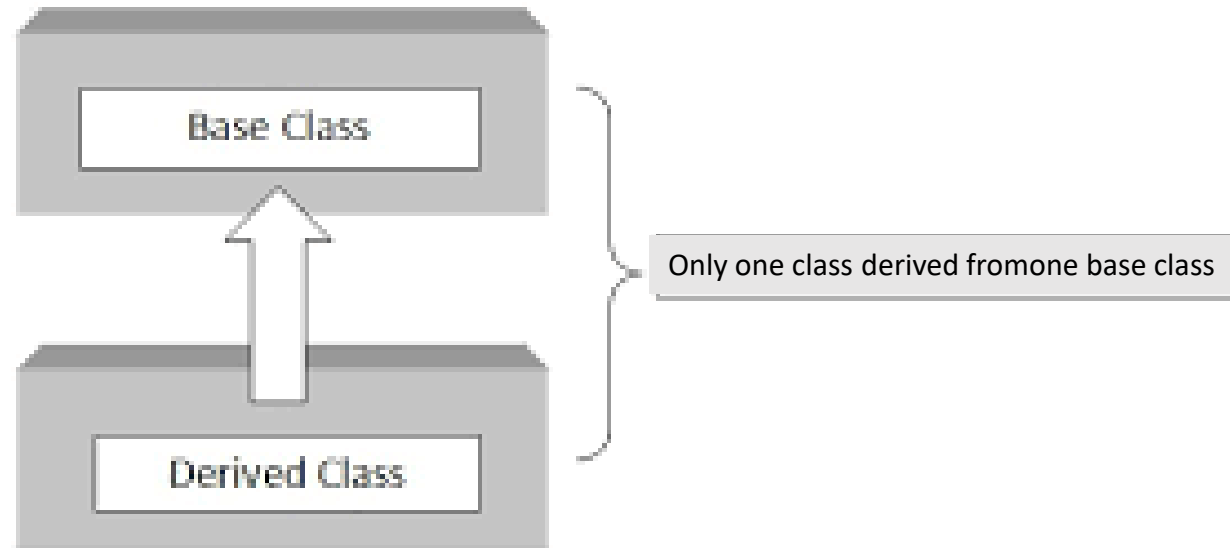
Behaviors/Methods:

1. Start():
2. Stop()
3. Accelerate(speed)
4. Brake()
5. GetInfo()



Types of Inheritance in C++

- **Single Inheritance**



Single inheritance

```
// Base class
class Vehicle {
public:
    string brand = "Toyota";
    void start() {
        cout << "Vehicle engine started." << endl;
    }
};
```

```
// Derived class
class Car: public Vehicle {
public:
    string model = "Corolla";
};
```

```
int main() {
    Car myCar;
    myCar.start();
    cout << myCar.brand + " " + myCar.model;
    return 0;
}
```

Syntax:

```
class subclass_name : access_mode base_class
{
    // body of subclass
};
```

Tutorial

- Consider a scenario involving two classes: **Employee** and **Manager**. The **Employee** class serves as the base class with attributes representing employee details, while the **Manager** class is derived from the **Employee** class, extending it with additional attributes specific to managerial roles.

Solution:

```
#include <iostream>
#include <string>

class Employee {
protected:
    std::string name;
    int id;

public:
    void setDetails(const std::string& empName, int empID) {
        name = empName;
        id = empID;
    }

    void displayDetails() {
        std::cout << "Employee Name: " << name << std::endl;
        std::cout << "Employee ID: " << id << std::endl;
    }
};
```

```
class Manager : public Employee {
private:
    std::string department;

public:
    void setManagerDetails(const std::string& mgrName, int mgrID, const std::string& dept) {
        name = mgrName; // Accessing the protected member of base class
        id = mgrID; // Accessing the protected member of base class
        department = dept;
    }

    void displayDetails() {
        std::cout << "Manager Name: " << name << std::endl;
        std::cout << "Manager ID: " << id << std::endl;
        std::cout << "Department: " << department << std::endl;
    }
};
```

```
int main() {  
  
    // Creating an Employee object and setting details  
    Employee emp;  
    emp.setDetails("John Doe", 1001);  
  
    std::cout << "Employee Details:" << std::endl;  
    emp.displayDetails();  
    std::cout << std::endl;  
  
    // Creating a Manager object and setting manager-specific details  
    Manager mgr;  
    mgr.setManagerDetails("Jane Smith", 2001, "Operations");  
  
    std::cout << "Manager Details:" << std::endl;  
    mgr.displayDetails();  
  
    return 0;  
}
```

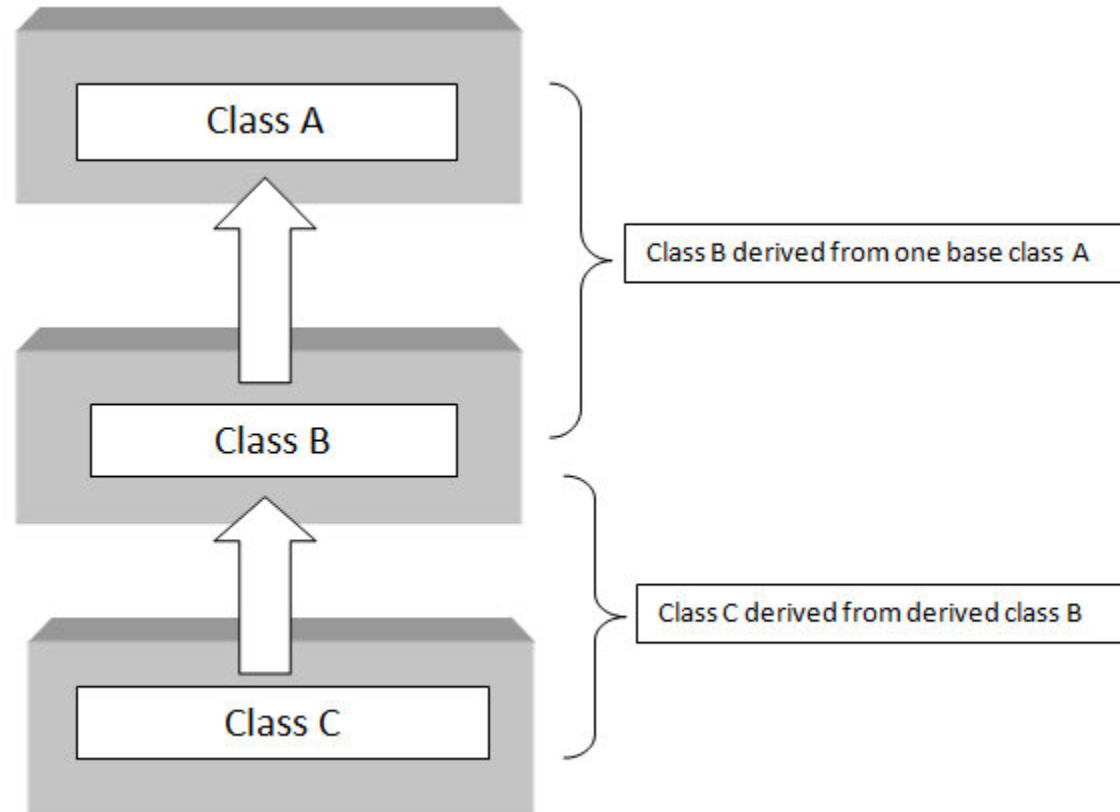
Output:

Employee Details:
Employee Name: John Doe
Employee ID: 1001

Output:

Manager Details:
Manager Name: Jane Smith
Manager ID: 2001
Department: Operations

Multilevel Inheritance



syntax

```
class A // base class
```

```
{
```

```
.....
```

```
};
```

```
class B : access_specifier A // derived class
```

```
{
```

```
.....
```

```
};
```

```
class C : access_specifier B // derived from derived class B
```

```
{
```

```
.....
```

```
};
```



Multilevel Inheritance

- A class can also be derived from one class, which is already derived

```
// Base class (parent)
class MyClass {
    public:
        void myFunction() {
            cout << "Some content in
parent class." ;
        }
};
```

```
// Derived class (grandchild)
class MyGrandChild: public MyChild
{
};
```

```
// Derived class (child)
class MyChild: public MyClass {
};
```

```
int main() {
    MyGrandChild myObj;
    myObj.myFunction();
    return 0;
}
```

```
class base //single base class
```

```
{  
    public:  
    int x;  
    void getdata()  
    {  
        cout << "Enter value of x= ";  
        cin >> x;  
    }  
};
```

```
class derive1 : public base
```

```
{  
    public:  
    int y;  
    void readdata()  
    {  
        cout << "\nEnter value of y= ";  
        cin >> y;  
    }  
};
```

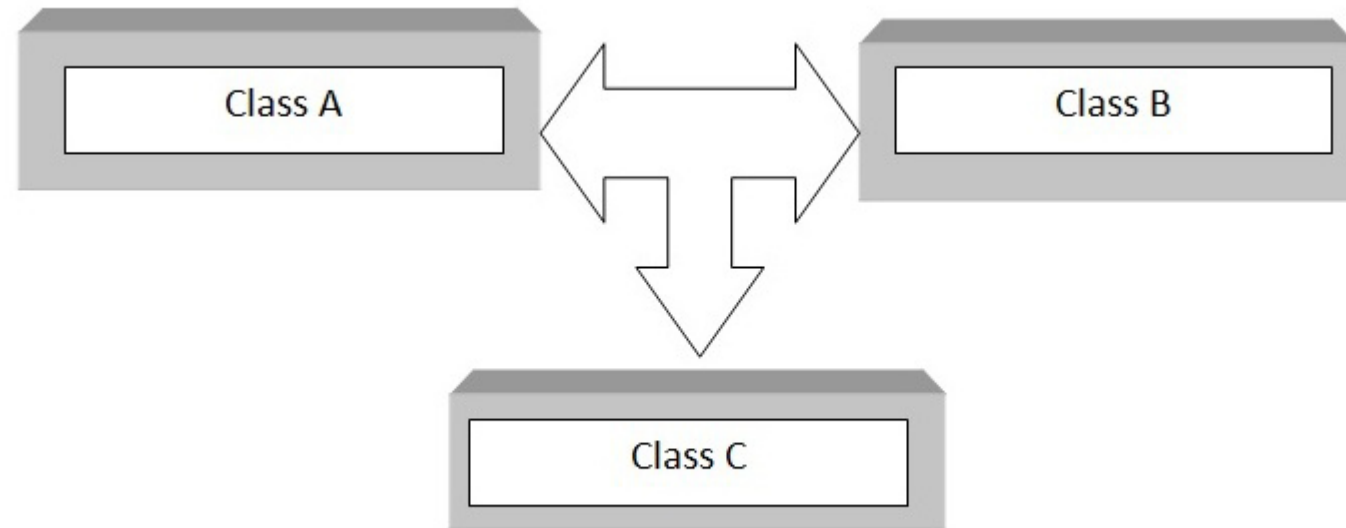
```
class derive2 : public derive1
```

```
{  
    private:  
    int z;  
    public:  
    void indata()  
    {  
        cout << "\nEnter value of z= ";  
        cin >> z;  
    }  
    void product()  
    {  
        cout << "\nProduct= " << x * y * z;  
    } };  
int main()  
{  
    derive2 a;    //object of derived class  
    a.getdata();  
    a.readdata();  
    a.indata();  
    a.product();  
    return 0;  
}
```

Output:

```
Enter value of x= 2  
Enter value of y= 3  
Enter value of z= 3  
Product= 18
```

Multiple Inheritance



// C++ program to explain multiple inheritance

```
class A
{
    public:
    int x;
    void getx()
    {
        cout << "enter value of x: "; cin >> x;
    }
};

class B
{
    public:
    int y;
    void gety()
    {
        cout << "enter value of y: "; cin >> y;
    }
};
```

```
class C : public A, public B
{
    public:
    void sum()
    {
        cout << "Sum = " << x + y;
    }
};

int main()
{
    C obj1;
    obj1.getx();
    obj1.gety();
    obj1.sum();
    return 0;
}
```

Output:

```
enter value of x: 5
enter value of y: 4
Sum = 9
```

Tutorial

- Consider a scenario involving multiple classes: **Musician**, **Writer**, and **Artist**. The **Musician** class represents individuals skilled in playing musical instruments, the **Writer** class represents those adept at creating written works, and the **Artist** class represents individuals with artistic drawing abilities.

Create a scenario where a class, **MultitalentedPerson**, inherits from both **Musician** and **Writer**, showcasing how multiple inheritance can be utilized.

Solution:

```
#include <iostream>
#include <string>

class Musician {
protected:
    std::string instrument;

public:
    void playInstrument(const std::string& instr)
    {
        instrument = instr;
        std::cout << "Playing " << instrument ;
    }
};
```

```
class Writer {
protected:
    std::string genre;

public:
    void writeGenre(const std::string& gen) {
        genre = gen;
        std::cout << "Writing in " << genre << " genre.\n";
    }
};
```

```
class MultitalentedPerson : public Musician, public Writer {
public:
    void displayCombinedSkills() {
        std::cout << "Multitalented person can play " << instrument << " and
write in " << genre << " genre.\n";
    }
};
```

```
int main() {
    MultitalentedPerson person;

    person.playInstrument("Piano");
    person.writeGenre("Fiction");

    std::cout << "Combined Skills:\n";
    person.displayCombinedSkills();

    return 0;
```



- When deriving a class from a base class, the base class may be inherited through public, protected or private inheritance

Modes of inheritance

- **Public Mode:**

- public members of the base class remain public in the derived class, protected members remain protected, and the base class's private members are not accessible in the derived class.

- **Protected Mode:**

Subclass derived from protected base class – both public & protected members of base class are protected in derived class

- **Private Mode:**

Subclass derived from private base class- both public & protected members of base class are private in derived class

C++ Implementation to show that a derived class doesn't inherit access to private data members.

```
class A {  
public:  
    int x;  
  
protected:  
    int y;  
  
private:  
    int z;  
};  
  
class B : public A {  
    // x is public  
    // y is protected  
    // z is not accessible from B  
};
```

```
class C : protected A {  
    // x is protected  
    // y is protected  
    // z is not accessible from C  
};  
  
class D : private A // 'private' is default  
for classes  
{  
    // x is private  
    // y is private  
    // z is not accessible from D  
};
```



End of session

Next Session
- Polymorphism