Polymorphism

22 September 2024 20:45

Polymorphism is the ability of object to take on different forms or behave in different ways depending on the context they are used. e.g. if we want to learn driving then do you learn BMW, Maruti, Suzuki? No, we learn Car and thereafter we can manage to drive any car.

Runtime polymorphism

Runtime polymorphism (also known as **dynamic polymorphism**) is a concept in OOPs where the method to be invoked is determined at runtime, rather than at compile time. It allows for flexibility in how objects interact, enabling the same piece of code to handle different types of objects and behave differently depending on the object's actual type.

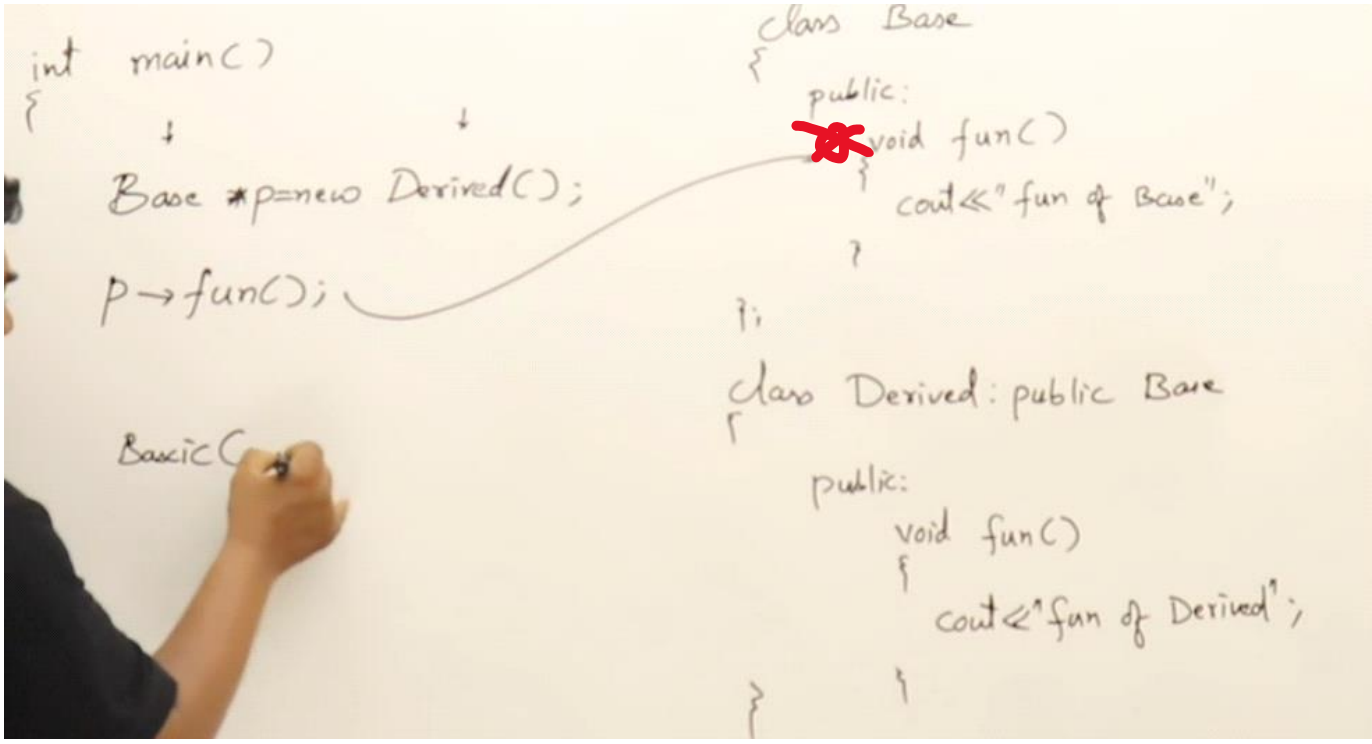
Function Overriding

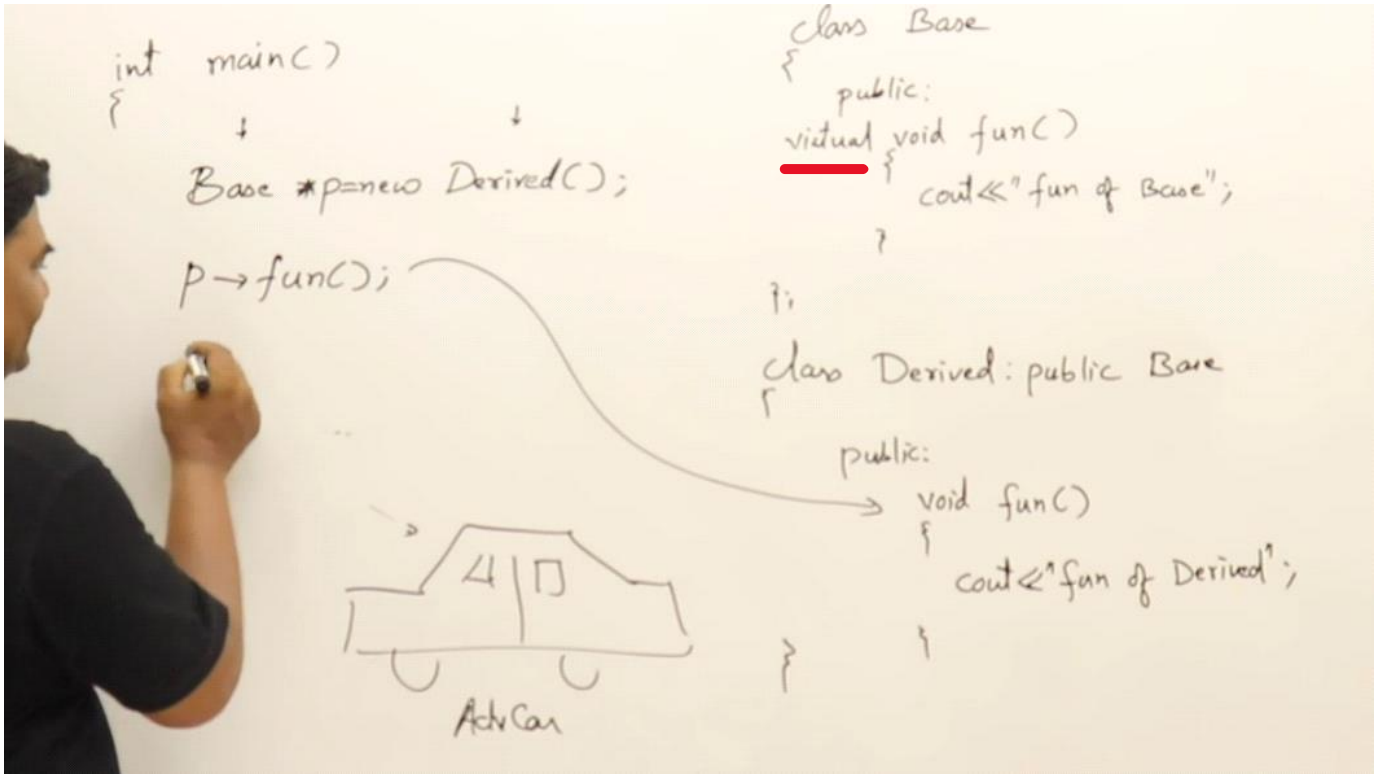
Function overriding occurs when a derived class provides a specific implementation for a method that is already defined in its base (or parent) class. This allows the derived class to customize or completely replace the behaviour of the inherited method.

```cpp
#include<iostream>
using namespace std;
class Base {
public:
    virtual void display(){
        cout<<"Base Class"<<endl;
    }
};
class Child1 : public Base {
};
class Child2 : public Base {
public:
    void display() override{
        cout<<"Child Class"<<endl;
    }
};
int main() {
    Base a;
    Child1 b;
    Child2 c;
    a.display();
    b.display();
    c.display();
    return 0;
}
```

Virtual Functions

A **virtual function** in C++ is a member function in a base class that can be overridden in derived classes. This feature enables **runtime polymorphism**.

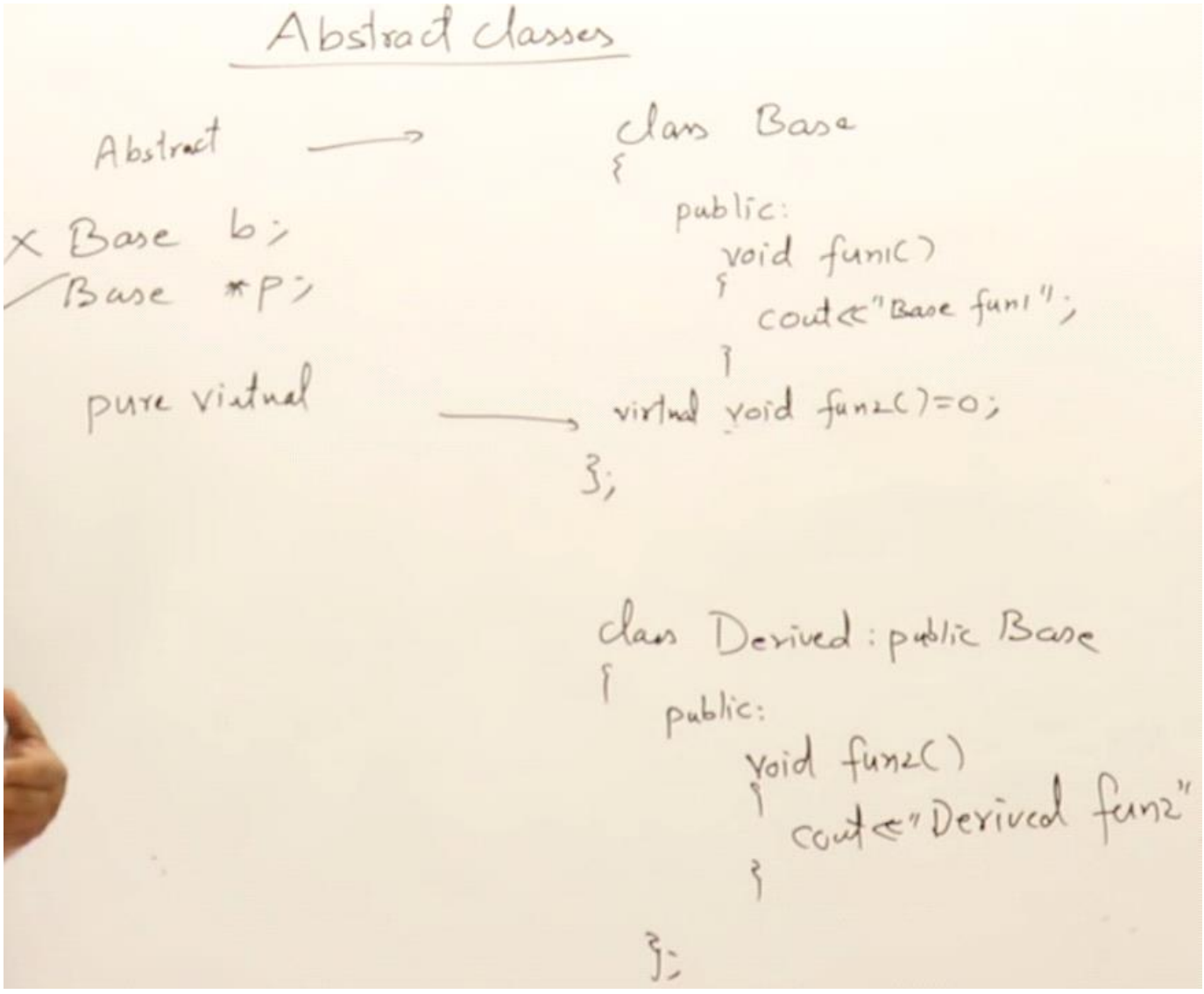


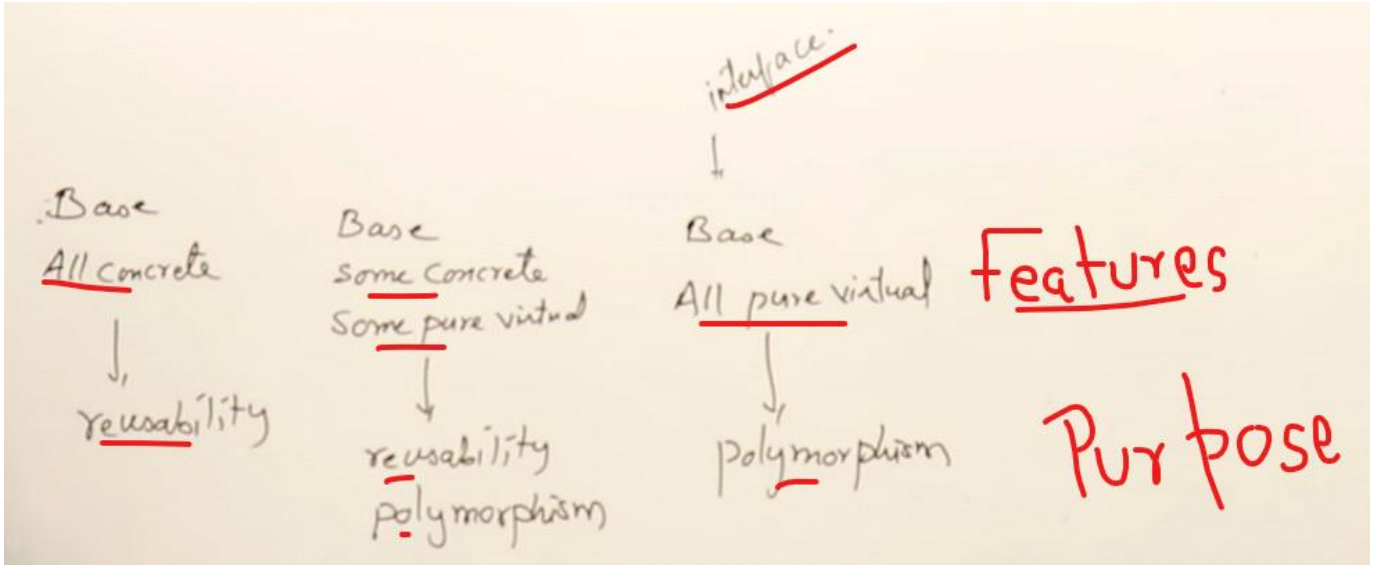


Abstract Classes

an abstract class is a class that contains at least one pure virtual function. A pure virtual function is a function that is declared in the base class and has no implementation in that class.

Classes that inherit from an abstract class must provide an implementation for all pure virtual functions, otherwise, they too become abstract classes.


Abstract classes
Abstract
class Base
{
public:
void func()
{
cout<<"Base func1";
}
Base b;
Base *p;
pure virtual
virtual void func2()=0;
};
class Derived : public Base
{
public:
void func2()
{
cout<<"Derived func2"
}
};


interface
Base
All concrete
reusability
Base
some concrete
Some pure virtual
reusability
polymorphism
Base
All pure virtual
polymorphism
Features
Purpose