Section 16 : Polymorphism

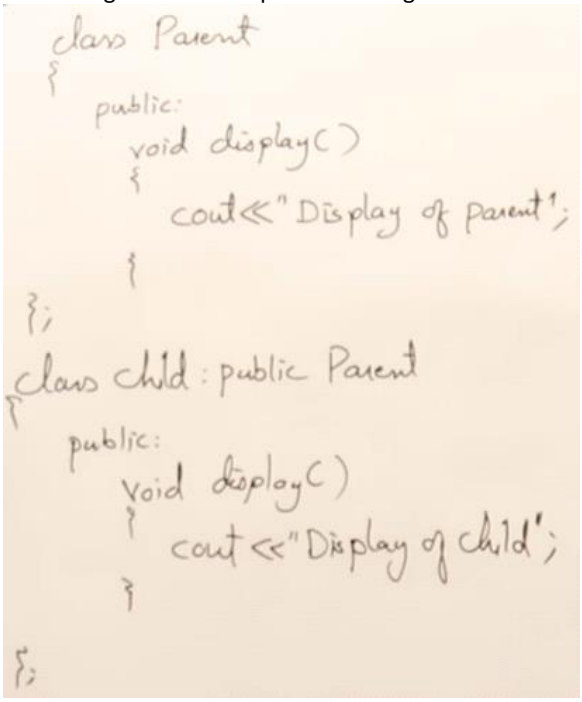
15 April 2022 16:49

© Rajat Kumar
https://www.linkedin.com/in/imRajat/
https://github.com/im-Rajat

Section 16 : Polymorphism

Function Overriding :

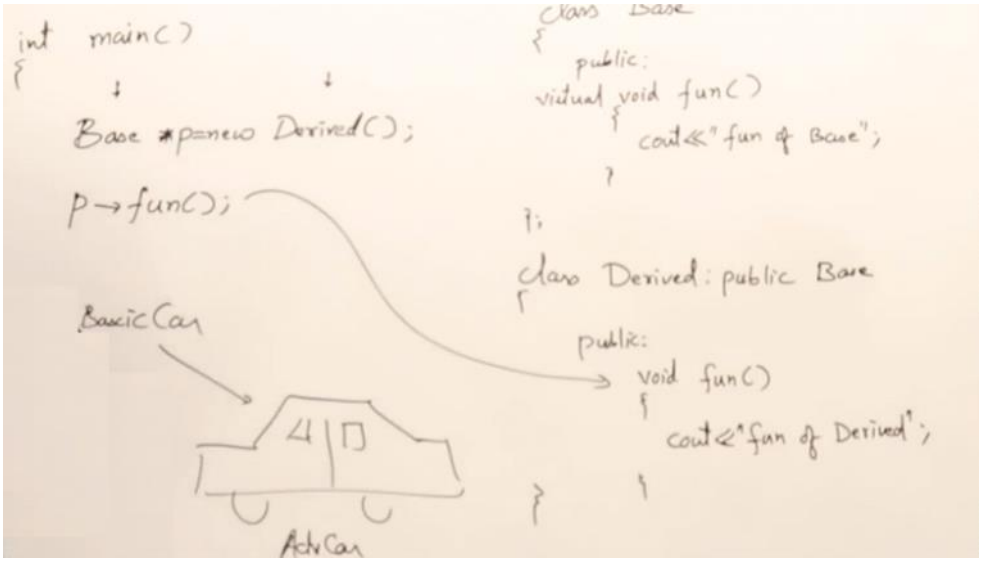
- A function redefine in child class which is already define in parent class.
- Redefining a function of parent class again in child class.



- Function overriding means the prototype of a function must be as it is same. It cannot have any variation at all.
- So when we do function overriding, make sure that the signature or the prototype of function is as it is same otherwise it is not taken as function overriding, it will become function overloading.

Virtual Functions :

- When we create a pointer of base class and point it to object of Derived class. And there is a function of same name (function overloading) in both base and derived class.
- And we call the function then base class function will be called, so to execute the correct function (of that object) we have virtual functions.
- So in base class function we can add virtual before it, so now if we call the object of derived class using pointer of base class, it will call function of derived class not of base class.



- Example :

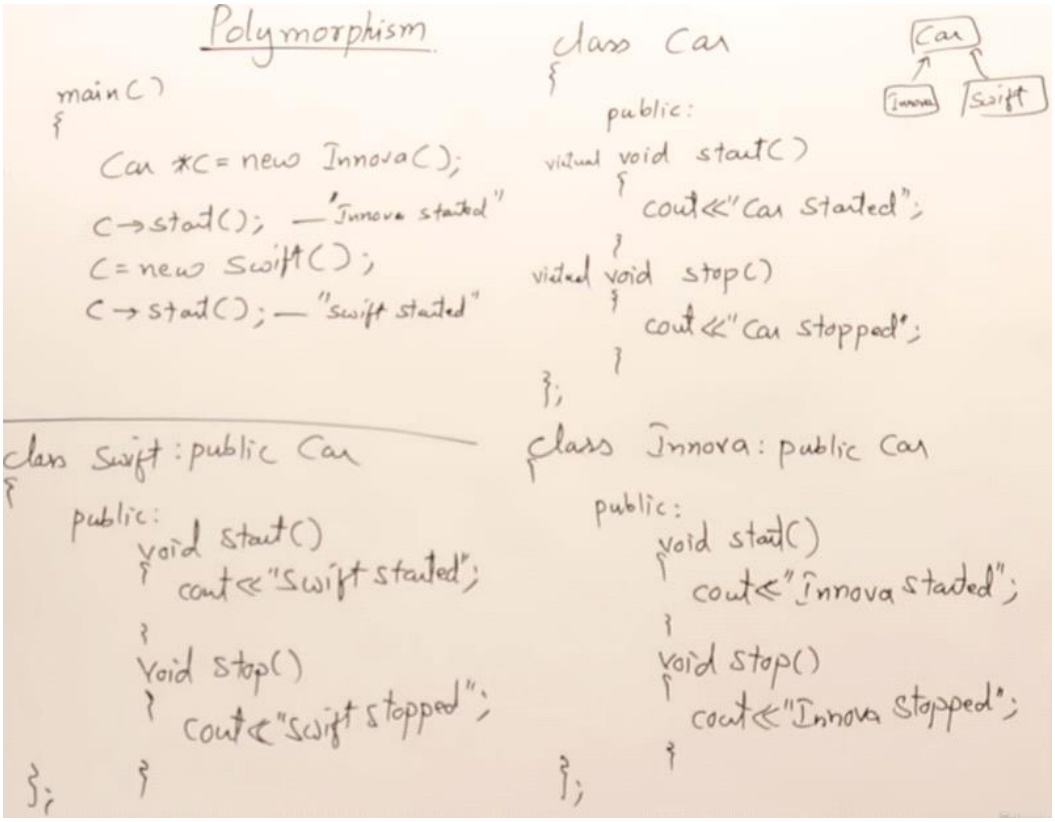
```cpp
class Base {
    public:
        void fun() {
            cout<<"Fun of Base\n";
        }
        virtual void dance() {
            cout<<"Dance of Base\n";
        }
};

class Derived : public Base {
    public:
        void fun() {
            cout<<"Fun of Derived\n";
        }
        void dance() {
            cout<<"Dance of Derived\n";
        }
};

int main()
{
    Base *p = new Derived();
    p->fun();       // print Fun of Base
    p->dance();     // print Dance of Derived

    return 0;
}
```

- If we have a pointer of base class and it's pointing to object of Derived class, It's in hands of programmers,
    - don't make virtual - base class function will be called
    - make it virtual - Derived class function will be called.

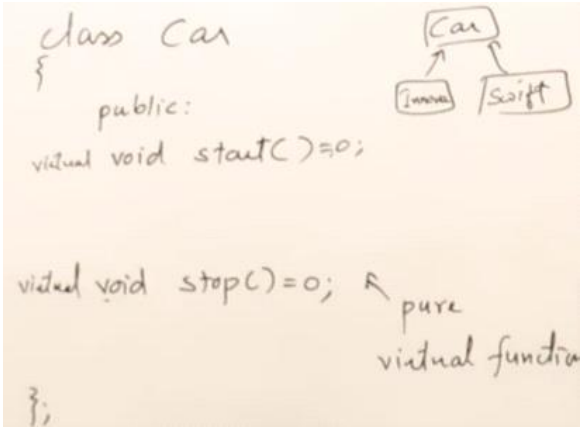
Runtime Polymorphism :

- What this polymorphism in above example:
    - Base class pointer pointing to a derive class object, and an overrated function is called
    - Then the function of derived class that is based on the object it will be called if the base class function is declared as virtual.
    - This is nothing but a **runtime polymorphism**.
    - So using virtual function and overriding function and base class pointer pointing to derived class object, we can achieve runtime polymorphism.
    - We need these three things to achieve runtime polymorphism :
        - Base class pointer pointing to derived class object
        - Overriding function
        - Virtual function

Polymorphism :



- Same statement but the function calls are different, this is polymorphism.
- We don't need to implement start and stop function in car class.
- That is for just achieving polymorphism because we want those functions must be implemented by subclasses.
- So we want that any class inherit from parent class must override those two functions. To make them compulsory we need to make them pure virtual functions by assigning virtual functions to 0.



- The above functions are called **pure virtual functions**.

Virtual Function Example :

```
class Car {
    public :
        void start() { cout<<"Car started\n"; }
        virtual void stop() { cout<<"Car stopped\n"; }
};

class Innova : public Car {
    public :
        void start() { cout<<"Innova started\n"; }
        void stop() { cout<<"Innova stopped\n"; }
};

class Swift : public Car {
    public :
        void start() { cout<<"Swift started\n"; }
        void stop() { cout<<"Swift stopped\n"; }
};

int main()
{
    Car *ptr;

    ptr = new Car();
    ptr->start();           // output : Car started

    ptr = new Innova();
    ptr->start();           // output : Car started
    ptr->stop();           // output : Innova started

    ptr = new Swift();
    ptr->start();           // output : Car started
    ptr->stop();            // output : Swift stopped

    return 0;
}
```

Pure Virtual Function :

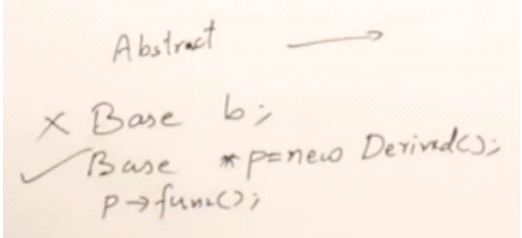
- When we assign virtual function to zero it's a pure virtual function.

- Those functions must be operated by derive classes otherwise the derive class will become abstract and that class car is actually abstract only. It's an **abstract**.
- We cannot create the object of that class because it is **abstract**.
- When a class inherited from that one, then this also become abstract if it is not overrating, so it must overwrite.
- So we have said that those functions must be override.
- So pure virtual function must be operated by derived classes. And the purpose of pure virtual function is to achieve polymorphism.

- We cannot create the object of car plus but we can create a reference pointer.

Abstract Classes :

- If a class is having pure virtual function then it's called as abstract class.
- We can't create it's object, but we can create it's pointer to achieve polymorphism.



- It must have pure virtual functions :
    - Virtual void start() = 0;
    - Virtual void stop() = 0;
- The parent class will just have the declaration of the function with the derived classes must implement those functions.

- Redefining a function of base class in a child class that is derived class is **Function overriding**. So when we are saying we want to redefine it means we want to achieve **polymorphism**.
- So for achieving polymorphism, we make the functions in the base class as virtual, saying that they are not real, they are just for

namesake. And the real function is in the child class, which is override.

- We also don't need to body of virtual function in base class, so assign it with 0 :
  - Virtual void start() = 0; // this is called as pure virtual function.
- **Pure virtual functions** are useful for defining **interface** (can be called as interface).

Example : (Basically car class is for generalization)

```
class Car {
    public :
        virtual void start() = 0;
};

class Innova : public Car {
    public :
        void start() { cout<<"Innova started\n"; }
};

class Swift : public Car {
    public :
        void start() { cout<<"Swift started\n"; }
};

int main()
{
    //Car c;       // error : cannot create abstract class object
    Car *ptr;

    ptr = new Innova();      // output : Innova started
    ptr->start();

    ptr = new Swift();       // output : Swift started
    ptr->start();

    return 0;
}
```

Purpose of Inheritance :

- Reusability :
  - The derived class is getting that function borrowed from base class.
- To achieve polymorphism :
  - There is a pure one shall function, so this class must override that function.

Concrete Function : Function define and implement in the class itself.

Categorize of classes in C++ based on purpose : (In Base Class)

- All Concreate functions/class (Reusability)
- Some Concrete and come pure virtual (Reusability + Polymorphism) - called as **abstract**
- All pure virtual functions (polymorphism) - called as abstract or Such class we can also call it as **Interface**. (In java it's called only as interface, its's a keyword in java)

- So if a class is inheriting from abstract class, it will also become abstract if we don’t override the pure virtual functions.

- Function Overriding is for achieving Polymorphism.
- Base class function must be virtual to achieve polymorphism.
- Example of Abstract Class :

```
class Demo
{
    public:
        vitrualvoid fun1()=0;
};
```

- Concreate classes are perfectly useful for Reusability.
- Runtime Polymorphism is achieved using Base class Pointer to derived class object and override method is called.