30 August 2023 20:28

OOPs Part 2.

- → Access Specifiers
- → Inheritance
- → Types of Inheritance
- → Encapsulation
- → Abstraction
- → Polymorphism

Access Specifiers

- → private (By Default)
- → public
- → protected

```
#include<iostream> //Header File
using namespace std;
class Student{
      int rollno; //Private Attribute
      void print(){ //Private Function
         cout<<"RollNo:"<<rollno<<endl;
      }
};
int main(){ //Start of the main function
   Student s1;
   s1.print(); //Error
   s1.rollno=10; //Error
}
```

```
#include<iostream> //Header File
using namespace std;
class Student{
      int rollno; //Private Attribute
      public:
      void print(){ //Public Function
         cout<<"RollNo:"<<rollno<<endl;
      }
};
int main(){ //Start of the main function
   Student s1;
   s1.print();
//   s1.rollno=10; //Error
}
```

```
#include<iostream> //Header File
using namespace std;
class Student{
```

```
    public:
        int rollno; //Public Attribute
        void print(){ //Public Function
            cout<<"RollNo:"<<rollno<<endl;
        }
};
int main(){ //Start of the main function
    Student s1;
    s1.rollno=10;
    s1.print();
}
```

Private → Only inside class

Public → Anywhere

Protected → After inheritance.

Inheritance :

Reusability

class A

int x
int y
fun 1

class B

int x
int y
fun 1
int z ✓
fun 2 ✓

Parent Class — Child Class

Base Class — Derived Class

Super Class — Sub Class

class A {

};

class B : mode of inheritance A {

Private, Public, Protected.

3

```
#include<iostream> //Header File
using namespace std;
class A{
   private: //Only in Class A
      int x;
   public: //Anywhere
      int y;
   protected: //In class A and child classes of A
      int z;
};
class B: public A{
   void fun(){
    //  x=10; //Error
      y=20;
      z=30;
   }
};
int main(){ //Start of the main function
   A obj;
  // obj.x=1;  //Error
  obj.y=2; //NO ERROR
 // obj.z=3; //Error
}
```

```
#include<iostream> //Header File
using namespace std;
class A{
   private: //Only in Class A
      int x;
   public: //Anywhere
      int y;
   protected: //In class A and child classes of A
      int z;
};
class B: public A{ //All the access specifiers remain as such
   void fun(){
    //  x=10; //Error (PRIVATE)
      y=20; //(PUBLIC)
      z=30; //(PROTECTED)
   }
};
class C: public B{
   void fun2(){
      z=50;
   }
};
int main(){ //Start of the main function
   A obj;
  // obj.x=1;  //Error
  obj.y=2; //NO ERROR
 // obj.z=3; //Error
}
```

```
#include<iostream> //Header File
using namespace std;
class A{
   private: //Only in Class A
      int x;
   public: //Anywhere
      int y;
   protected: //In class A and child classes of A
      int z;
};
class B: private A{ //All the access specifiers change to private
   void fun(){
    //  x=10; //Error (PRIVATE)
      y=20; //(PRIVATE)
      z=30; //(PRIVATE)
   }
};
class C: public B{
   void fun2(){
      z=100; //Error
   }
}
int main(){ //Start of the main function
```

```
   B obj;
   obj.x=1;  //Error
  obj.y=2; //Error
  obj.z=3; //Error
}
```

```
#include<iostream> //Header File
using namespace std;
class A{
   private: //Only in Class A
      int x;
   public: //Anywhere
      int y;
   protected: //In class A and child classes of A
      int z;
};
class B: protected A{ //Private rewmains as such, Others change to protected
   void fun(){
    //  x=10; //Error (PRIVATE)
      y=20; //(PROTECTED)
      z=30; //(PROTECTED)
   }
};
int main(){ //Start of the main function
   B obj;
   obj.x=1;  //Error
  obj.y=2; //Error
  obj.z=3; //Error
}
```

```
#include<iostream> //Header File
using namespace std;
class A{
   private: //Only in Class A
      int x;
   public: //Anywhere
      int y;
   protected: //In class A and child classes of A
      int z;
};
class B: protected A{ //Private rewmains as such, Others change to protected
   void fun(){
    //  x=10; //Error (PRIVATE)
      y=20; //(PROTECTED)
      z=30; //(PROTECTED)
   }
};
class C: public B{
   void fun2(){
      z=100; //No Error
   }
}
int main(){ //Start of the main function
   B obj;
   obj.x=1;  //Error
  obj.y=2; //Error
  obj.z=3; //Error
}
```

Access Specifiers

Public → anywhere

Private → Only inside class

Protected → In class and child classes

Visibility modes ↓ / Access Specifiers →	private	protected	public
private	Private	private	private

private	Private	private	private
protected	Private	protected	protected
public	Private	Protected	public

Private > Protected > Public

Types of inheritance

① Single inheritance

One Parent Class
One Child Class

```
#include<iostream> //Header File
using namespace std;
class Animal{
   public:
   void eat(){
      cout<<"Animals can eat"<<endl;
   }
};
class Dog: public Animal{
   public:
   void bark(){
      cout<<"Dogs can bark"<<endl;
   }
};
int main(){ //Start of the main function
   Animal a;
   a.eat();
   //a.bark(); //Error
   Dog d;
   d.eat();
   d.bark();
}
```

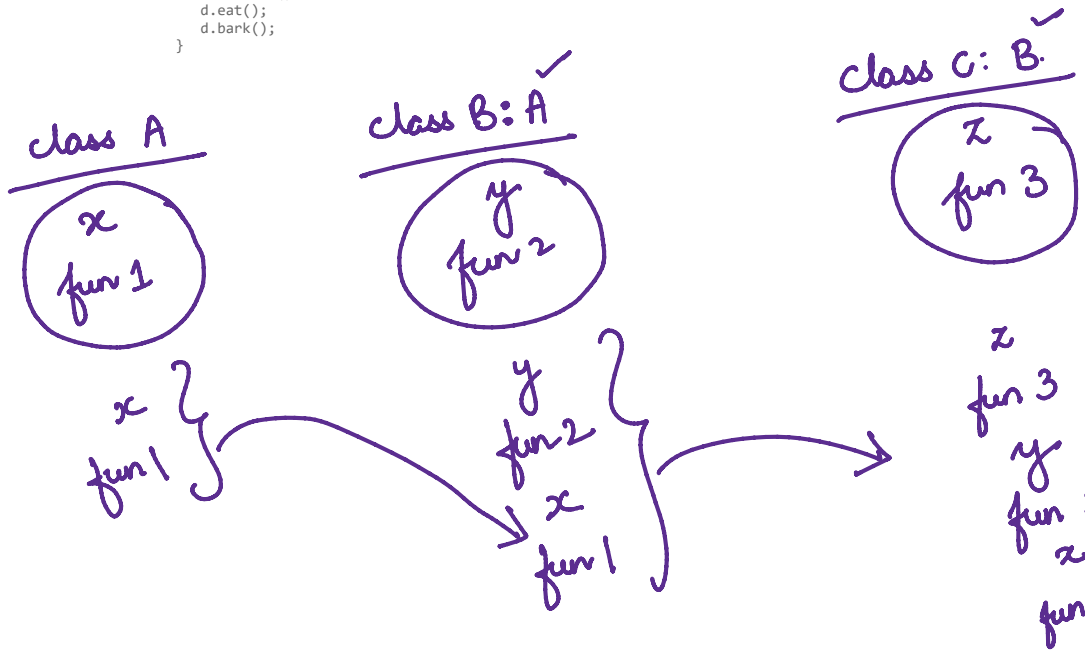
② Multilevel Inheritance

A parent
↓
B child
↓
C

parent
child

child L C

```
#include<iostream> //Header File
using namespace std;
class Mammal{
   public:
   void breathe(){
      cout<<"Mammals can breathe"<<endl;
   }
};
class Animal: public Mammal{
   public:
   void eat(){
      cout<<"Animals can eat"<<endl;
   }
};
class Dog: public Animal{
   public:
   void bark(){
      cout<<"Dogs can bark"<<endl;
   }
};
int main(){ //Start of the main function
   Mammal m;
   m.breathe();
   // m.eat(); //Error
   // m.bark(); //Error
   Animal a;
   a.breathe();
   a.eat();
  // a.bark(); //Error
   Dog d;
   d.breathe();
   d.eat();
   d.bark();
}
```



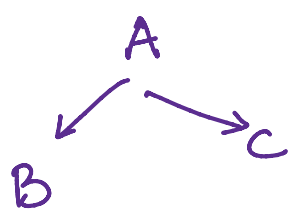
③ Hierarichal Inheritance

TREE

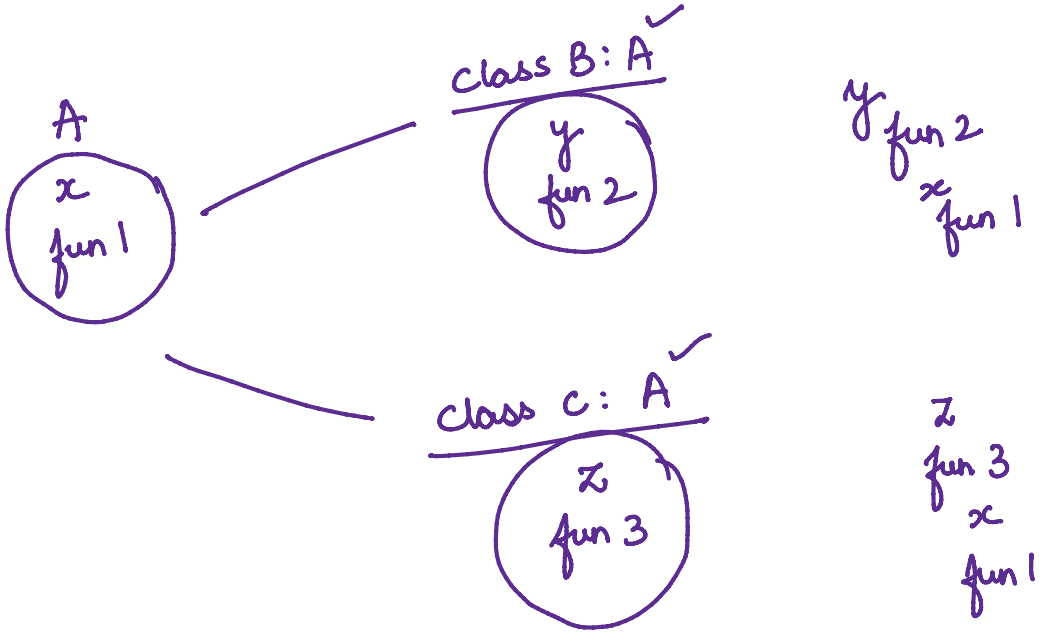
One Base Class

Multiple Child Classes

>1

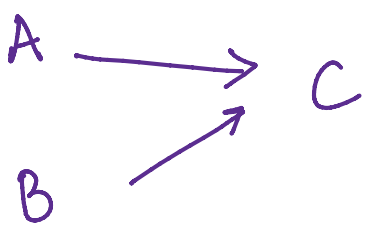


class B: A



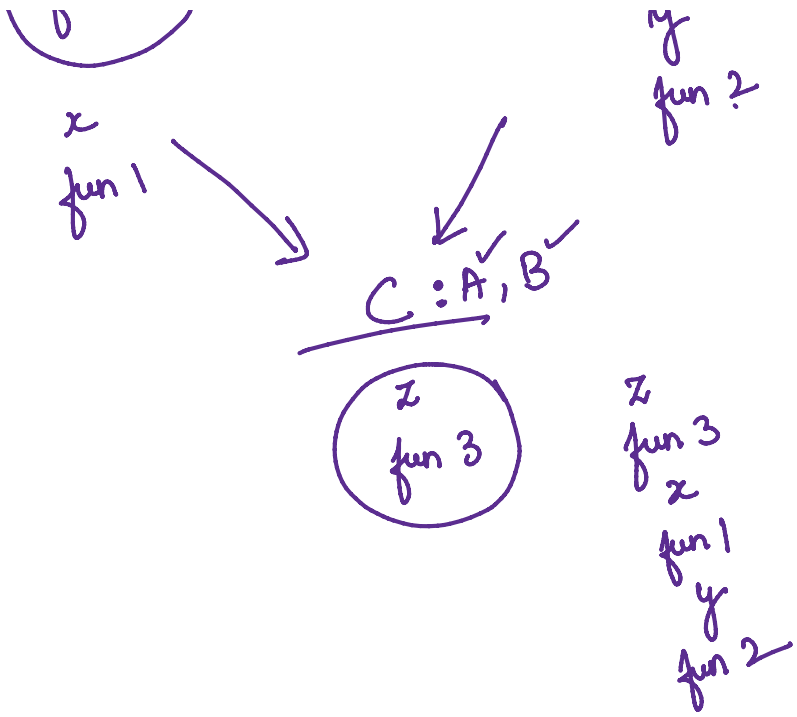
④ Multiple Inheritance

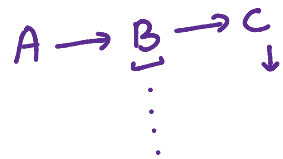
multiple base class
One child class



class C : public A , public B {

}






```
#include<iostream> //Header File
using namespace std;
class A{
   public:
   void fun1(){
      cout<<"In Class A Function 1"<<endl;
   }
   void fun2(){
      cout<<"In Class A Function 2"<<endl;
   }
};
class B{
   public:
   void fun1(){
      cout<<"In Class B Function 1"<<endl;
   }
   void fun3(){
      cout<<"In Class B Function 3"<<endl;
   }
};
class C: public A, public B{
};
int main(){ //Start of the main function
   C obj;
   obj.fun2(); //In Class A Function 2
   obj.fun3(); //In Class B Function 3
  // obj.fun1(); //Error (Ambiguity)
  obj.A::fun1(); //In Class A Function 1
  obj.B::fun1(); //In Class B Function 1
}
```

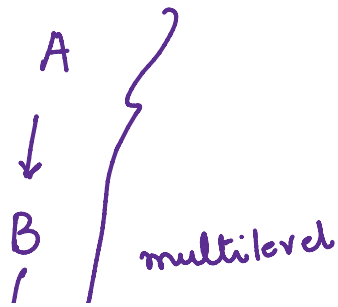
Class name : : Function name

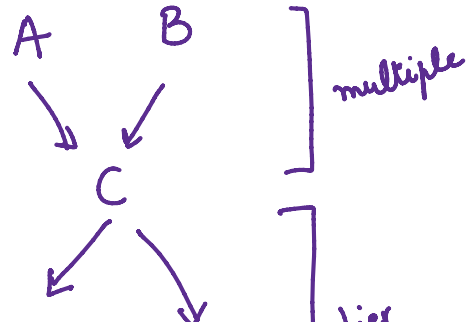
↓

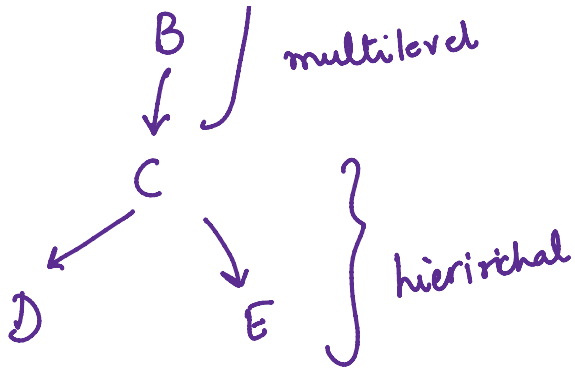
Scope Resolution Operator

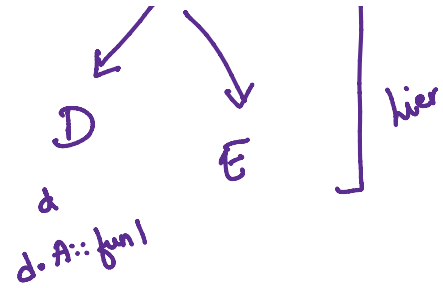
5.) Hybrid Inheritance

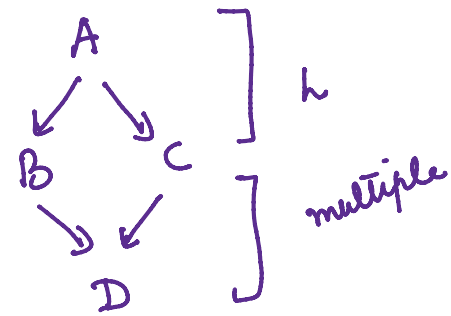
mixture












```
#include<iostream> //Header File
using namespace std;
class A{
   public:
   A(){
      cout<<"Constructor of A"<<endl;
   }
};
class B:public A{
   public:
   B(){
      cout<<"Constructor of B"<<endl;
   }
};
int main(){ //Start of the main function
      A obj; //Constructor of A
      B obj; //Constructor of A, Constructor of B
}
```

```
#include<iostream> //Header File
using namespace std;
class A{
   public:
   int x;
   A(){
      cout<<"Default Constructor of A"<<endl;
   }
   A(int x){
      this->x=x;
      cout<<"Parametrized Constructor of A"<<endl;
   }
};
class B:public A{
   public:
   int y;
   B(){
      cout<<"Default Constructor of B"<<endl;
   }
   B(int y){
      this->y=y;
      cout<<"Parametrized Constructor of B"<<endl;
   }
};
int main(){ //Start of the main function
   B obj1;  //Default Constructor of A,Default Constructor of B
}
```

```
#include<iostream> //Header File
using namespace std;
class A{
```

```
   public:
   int x;
   A(){
      cout<<"Default Constructor of A"<<endl;
   }
   A(int x){
      this->x=x;
      cout<<"Parametrized Constructor of A"<<endl;
   }
};
class B:public A{
   public:
   int y;
   B(){
      cout<<"Default Constructor of B"<<endl;
   }
   B(int y){
      this->y=y;
      cout<<"Parametrized Constructor of B"<<endl;
   }
};
int main(){ //Start of the main function
   B obj(20); //Default Constructor of A,Parametrized Constructor of B
}
```

•

```
#include<iostream> //Header File
using namespace std;
class A{
   public:
   int x;
   A(){
      cout<<"Default Constructor of A"<<endl;
   }
   A(int x){
      this->x=x;
      cout<<"Parametrized Constructor of A"<<endl;
   }
};
class B:public A{
   public:
   int y;
   B(){
      cout<<"Default Constructor of B"<<endl;
   }
   B(int y):A(y){
      this->y=y;
      cout<<"Parametrized Constructor of B"<<endl;
   }
};
int main(){ //Start of the main function
   B obj(20); //Parametrized Constructor of A, Parametrized Constructor of B
}
```

•

```
#include<iostream> //Header File
using namespace std;
class A{
   public:
   int x;
   A(){
      cout<<"Default Constructor of A"<<endl;
   }
   A(int x){
      this->x=x;
      cout<<"Parametrized Constructor of A"<<endl;
   }
};
class B:public A{
   public:
   int y;
   B(){
      cout<<"Default Constructor of B"<<endl;
   }
   B(int x,int y):A(x){
      this->y=y;
      cout<<"Parametrized Constructor of B"<<endl;
   }
};
int main(){ //Start of the main function
   B obj(20,30); //Parametrized Constructor of A, Parametrized Constructor of B
}
```

•

```
#include<iostream> //Header File
using namespace std;
class A{
   public:
   void fun(){
      cout<<"Function  of Class A"<<endl;
   }
```

Function Overriding

↓

function

```
using namespace std;
class A{
   public:
   void fun(){
      cout<<"Function  of Class A"<<endl;
   }
};
class B:public A{
   public:
    void fun(){ //function Overriding
      cout<<"Function  of Class B"<<endl;
   }
};
int main(){ //Start of the main function
   B obj;
   obj.fun(); //Output: Function  of Class B
   obj.A::fun(); //Output: Function  of Class A
}
```

Function ... 0

↓

Child class function overrides the parent class function.

Poly morphism → many forms

eg:
- Function Overloading
- Function Overriding

Ability to have more than 1 form

✓ **Encapsulation** → Bind variables & functions together.

capsule

class A{
- variables
- functions

}

Abstraction hide data from users

↓

variables private
functions public

User → functions → Variables

User main function

OOPs

4 main Pillars

- → Encapsulation
- → Abstraction
- → Inheritance
- → Polymorphism
  - → Run Time
    - eg. Function Overriding
  - → Compile Time
    - eg. Function Overloading