Inheritance

Chapter: 7, Teach Yourself C++

Example

```
class base{
private: int y;
public:
   base (int n){cout<<"Constructor in base class"<<endl;
                 y=n;}
   ~base (){cout<<"Destructor in base class"<<endl;}  };
class derived: public base{
   private: int a;
public:
   derived (int x){
      cout<<"Constructor in derived class"<<endl;
      a=x;  }
   ~derived(){cout<<"Destructor in derived class"<<endl;}
   void showa() { cout<<a<<endl;}   };
int main()
{ derived ob(100);
}
```

Example

```
class base{
private: int y;
public:
   base (int n){cout<<"Constructor in base class"<<endl;
                 y=n;}
   ~base (){cout<<"Destructor in base class"<<endl;}  };
class derived: public base{
   private: int a;
public:
   derived (int x){
      cout<<"Constructor in derived class"<<endl;
      a=x;  }
   ~derived(){cout<<"Destructor in derived class"<<endl;}
   void showa() { cout<<a<<endl;}   };
int main()
{ derived ob(100);
}
```

ERROR!!

Example

```
class base{
private: int y;
public:
   base (int n){cout<<"Constructor in base class"<<endl;
                 y=n;}
   ~base (){cout<<"Destructor in base class"<<endl;}  };
class derived: public base{
   private: int a;
public:
   derived (int x){
      cout<<"Constructor in derived class"<<endl;
      a=x;  }
   ~derived(){cout<<"Destructor in derived class"<<endl;}
   void showa() { cout<<a<<endl;}   };
int main()
{ derived ob(100);
}
```

ERROR!!
Because there is no matching function to call the constructor function of the base class.

Example

```
class base{
private: int y;
public:
   base (int n){cout<<"Constructor in base class"<<endl;
                y=n;}
   ~base (){cout<<"Destructor in base class"<<endl;}  };
class derived: public base{
   private: int a;
public:
    derived (int x, int y) : base (y) {
      cout<<"Constructor in derived class"<<endl;
      a=x;  }
   ~derived(){cout<<"Destructor in derived class"<<endl;}
   void showa() { cout<<a<<endl;}   };
int main()
{ derived ob(100, 200);
}
```

Example

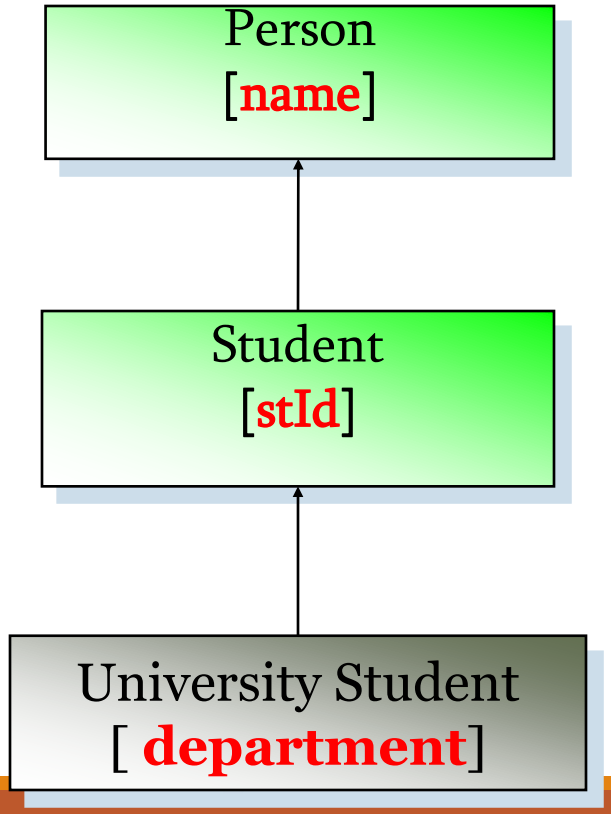
```
class base{
private: int y;
public:
   base (int n){cout<<"Constructor in base class"<<endl;
                y=n;}
   ~base (){cout<<"Destructor in base class"<<endl;}  };
class derived: public base{
   private: int a;
public:
    derived (int y) : base (y) {
      cout<<"Constructor in derived class"<<endl;
      a=0;  }
   ~derived(){cout<<"Destructor in derived class"<<endl;}
   void showa() { cout<<a<<endl;}   };
int main()
{ derived ob(100);
}
```

Multilevel Inheritance

1. Multilevel Class Hierarchy
   - A derived class can be used as base class of another derived class.

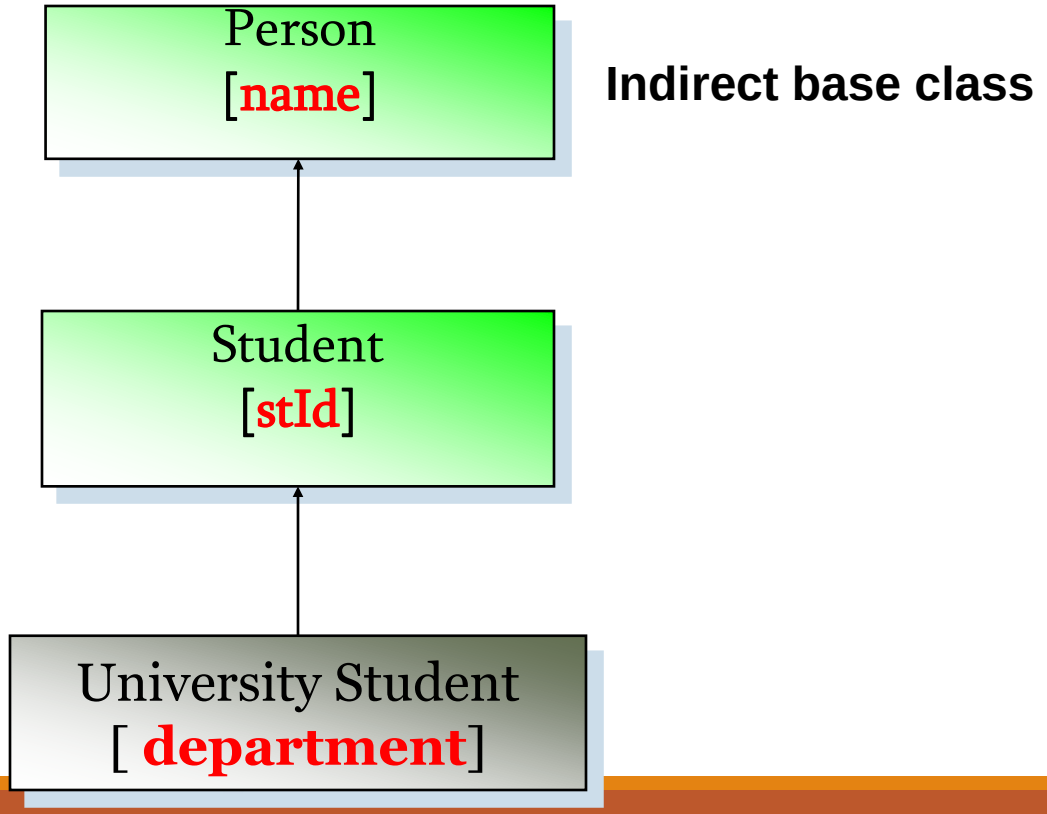
Multilevel Inheritance

1. Multilevel Class Hierarchy
   - A derived class can be used as base class of another derived class.



Multilevel Inheritance

1. Multilevel Class Hierarchy
   - A derived class can be used as base class of another derived class.



Multilevel Inheritance

- Constructor functions of all three classes are called in order of derivation.
  - That is, Person's constructor is called first
  - Then student's constructor is called
  - And then, university student's constructor is called.

- Destructor functions are called in reverse order.

Multilevel Inheritance

```
class Person{
 Private: char name[20];
 Public: int contactNo;
 Protected: char gender;
 // here goes member functions
}
 class Student : private  Person{
      private:
         int id;
      public:
       // here goes member functions
 };
```

```
class UnivStudent : public Student{
      private:
         int dept;
      public:
void setVal (int d, int i, char* nm, int
      contact, char g){
              dept = d;
              id = i;
              strcpy (name,nm);
              contactNo = contact;
              gender = g;
        }
};
```

Multilevel Inheritance

```
class Person{
 Private: char name[20];
 Public: int contactNo;
 Protected: char gender;
 // here goes member functions
}
 class Student : private  Person{
      private:
         int id;
      public:
       // here goes member functions
 };
```

ERROR

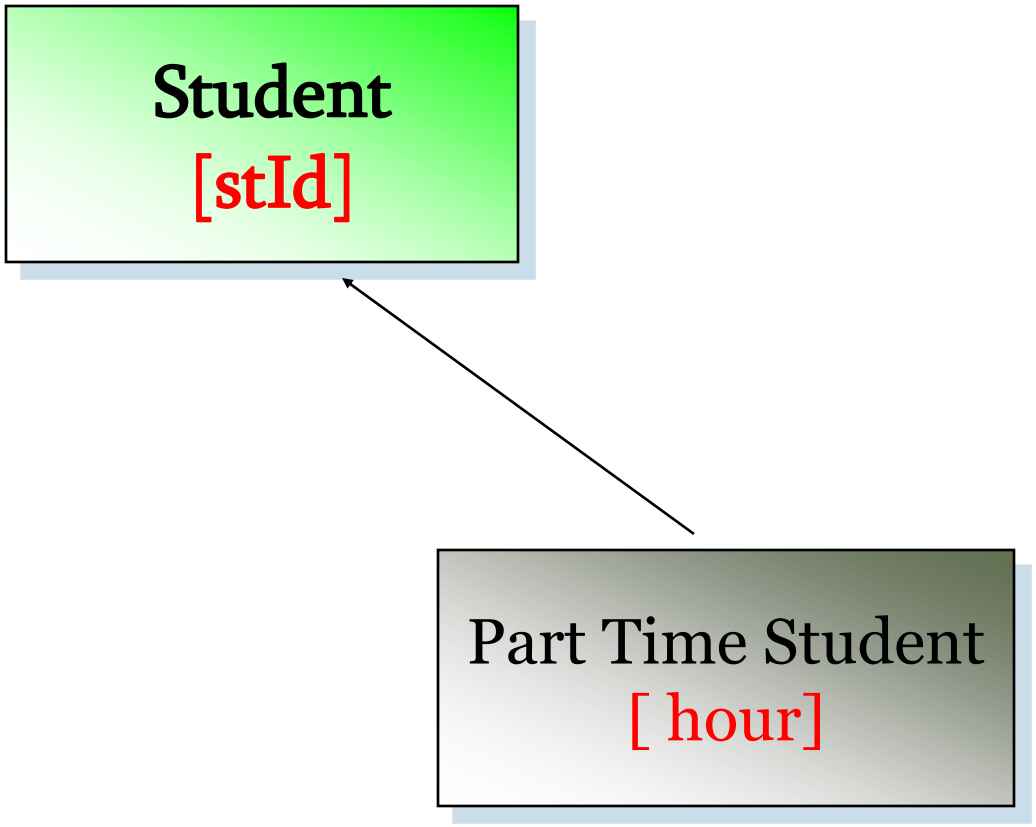
```
class UnivStudent : public Student{
     private:
          dept;
          ic:
 void setVal (int d, int i, char* nm, int
     contact, char g){
          dept = d;
          id = i;
          strcpy (name,nm);
          contactNo = contact;
          gender = g;
     }
};
```

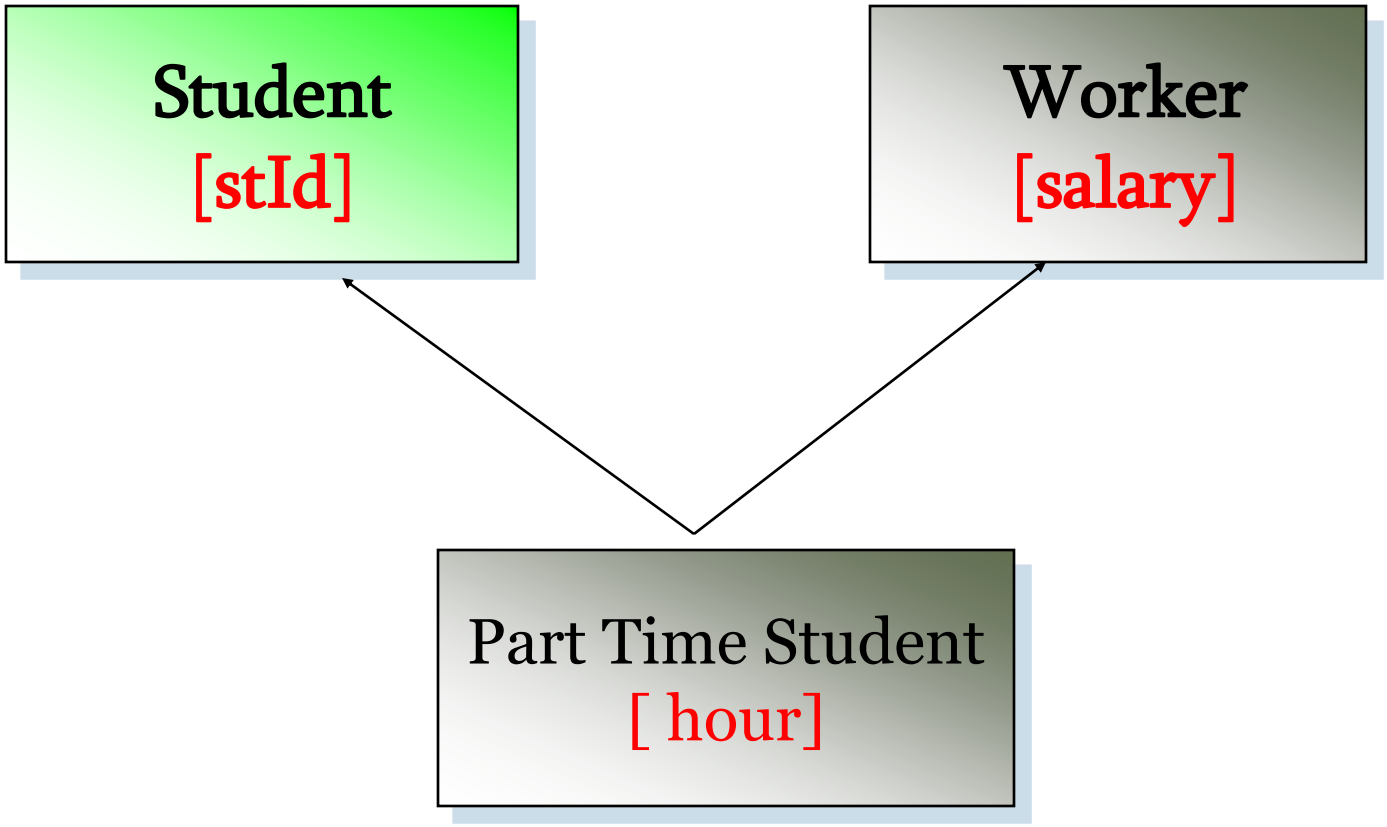
Multilevel Inheritance

```
class base
{
   int a;
public:
   base(int x)
   {cout<<"Contructor in base"<<endl;
   a=x;}
   int geta(){return a;}
};
class derived1 : public base
{
   int b;
public:
   derived1(int x, int y):base(x)
    {cout<<"Contructor in derived1"<<endl;
    b=y;}
   int getb(){return b;}
};
```

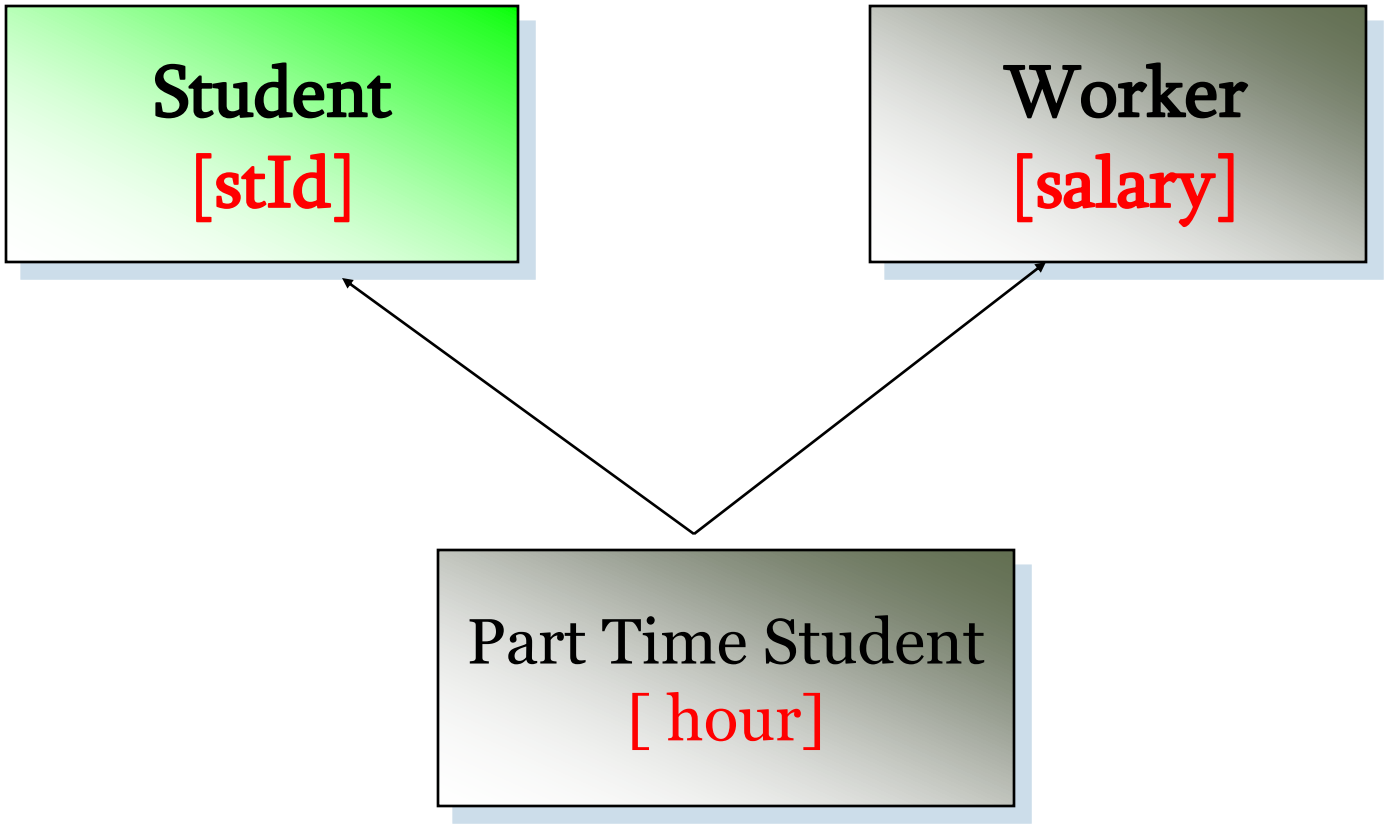
```
class derived2: public derived1
{
   int c;
public:
   derived2(int x, int y, int z):derived1(x, y)
    {cout<<"Contructor in derived2"<<endl;
    c=z;}
   void show(){cout<<geta()<<" "<<getb()<<“
 "<<c<<endl;}
};

int main()
{
   derived2 ob(1,2,3);
   ob.show();
   cout<<ob.geta()<<" "<<ob.getb()<<" "<<endl;
}
```

Part Time Student
[hour]


Student
[stId]
Part Time Student
[hour]


Student
[stId]
Worker
[salary]
Part Time Student
[hour]

Multiple Inheritance



Multilevel Inheritance

- Multiple inheritance allows a derived class to be derived from more than one base class. It syntax is:

```
class Base1 { };

class Base2 { };

class Derived : (acc-spec) Base1, (acc-spec) Base2

{ … };
```

Multilevel Inheritance: Example

```
class Student{
        int id ;
        public:
        Student (int i){
            id = i;
            cout <<"C : Student\n";
        }
};

class Worker{
        int salary ;
        public:
        Worker (int sal){
            salary = sal;
            cout << "C : Worker\n";
        }
};
```

```
class PTstudent : public Student, public
 Worker
{
        int hour ;
        public:
        PTstudent (int h, int sal, int id)
: Student (id), Worker (sal) {
            hour = h;
            cout << "C : PTstudent\n";
        }
};

int main()
{
   PTstudent p1(6,2000,3);
}
```

Multilevel Inheritance: Example

```
class Student{
        int id ;
        public:
        Student (int i){
            id = i;
            cout <<"C : Student\n";
        }
};

class Worker{
        int salary ;
        public:
        Worker (int sal){
            salary = sal;
            cout << "C : Worker\n";
        }
};
```

```
class PTstudent : public Student, public
 Worker
{
        int hour ;
        public:
        PTstudent (int h, int sal, int id)
: Student (id), Worker (sal) {
            hour = h;
            cout << "C : PTstudent\n";
        }
};

int main()
{
   PTstudent p1(6,2000,3);
}
```

C: Student
C: Worker
C: PTstudent

Multilevel Inheritance: Example

```
class base1
{
public:
   base1(){cout<<"Constructor in base
1"<<endl;}

   ~base1() {cout<<"Destructor in base 1"<<endl;}
};
class base2
{
public:
   base2(){cout<<"Constructor in base 2"<<endl;}
   ~base2() {cout<<"Destructor in base 2"<<endl;}
};
```

```
class derived: public base1, public
base2
{
public:
   derived(){cout<<"Constructor in
derived"<<endl;}
   ~derived() {cout<<"Destructor in
derived"<<endl;}
};
int main()
{ derived ob; }
```

Multilevel Inheritance: Example

```
class base1
{
public:
   base1(){cout<<"Constructor in base
1"<<endl;}

   ~base1() {cout<<"Destructor in base 1"<<endl;}
};
class base2
{
public:
   base2(){cout<<"Constructor in base 2"<<endl;}
   ~base2() {cout<<"Destructor in base 2"<<endl;}
};
```

```
class derived: public base1, public
base2
{
public:
   derived(){cout<<"Constructor in
derived"<<endl;}
   ~derived() {cout<<"Destructor in
derived"<<endl;}
};
int main()
{ derived ob; }
```

Constructor in base 1
Constructor in base 2
Constructor in derived
Destructor in derived
Destructor in base 2
Destructor in base 1

Multilevel Inheritance: Example

```
class Student{
        int id ;
        public:
        Student (int i){id = i;}
        void show()
        {cout<<id<<endl;}
};
class Worker{
        int salary ;
        public:
        Worker (int sal)
        {salary = sal;}
        void show()
        {cout<<salary<<endl;}
};
```

```
class PTstudent : public Student,
public Worker
{
        int hour ;
        public:
        PTstudent (int h, int sal, int
id) : Student (id), Worker (sal)
         { hour = h;}
        void showp()
        { show(); }
};
int main()
{
   PTstudent p1(6,2000,3);
}
```

Multilevel Inheritance: Example

```
class Student{
        int id ;
        public:
        Student (int i){id = i;}
        void show()
        {cout<<id<<endl;}
};
class Worker{
        int salary ;
        public:
        Worker (int sal)
        {salary = sal;}
        void show()
        {cout<<salary<<endl;}
};
```

```
class PTstudent : public Student,
public Worker
{
        int hour ;
        public:
        PTstudent (int h, int sal, int
id) : Stude                    (sal)
        {
        void                   ()
        { show(); }
};
int main()
{
   PTstudent p1(6,2000,3);
}
```

ERROR!!!

Solution:

```
class Student{
        int id ;
        public:
        Student (int i){id = i;}
        void show()
       {cout<<“ID:”<<“ ”<<id<<endl;}
};
class Worker{
        int salary ;
        public:
        Worker (int sal)
       {salary = sal;}
        void show()
       {cout<<“Salary:”<<“ ”salary<<endl;}
};
```

```
class PTstudent : public Student,
public Worker
{
        int hour ;
        public:
        PTstudent (int h, int sal, int
id) : Student (id), Worker (sal)
         { hour = h;}
        void showp()
        { Student::show();
            Worker::show();
    cout<<“Hour:”<<“ ”<<hour<<endl;}
};
int main()
{
   PTstudent p1(6,2000,3);
   p1.showp();
}
```

Solution:

```
class Student{
        int id ;
        public:
        Student (int i){id = i;}
        void show()
       {cout<<“ID:”<<“ ”<<id<<endl;}
};
class Worker{
        int salary ;
        public:
        Worker (int sal)
       {salary = sal;}
        void show()
       {cout<<“Salary:”<<“ ”salary<<endl;}
};
```

```
class PTstudent : public Student,
public Worker
{
        int hour ;
        public:
        PTstudent (int h, int sal, int
id) : Student (id), Worker (sal)
         { hour = h;}
        void showp()
        { Student::show();
            Worker::show();
    cout<<“Hour:”<<“ ”<<hour<<endl;}
};
int main()
{
   PTstudent p1(6,2000,3);
   p1.showp();
}
```

ID:3
Salary: 2000
Hour: 6

Practice

Create a car and a truck class. Both of these classes will inherit another class called vehicle. Create car() and truck() constructor functions. Have each pass along appropriate arguments to vehicle() constructor.

- **class vahicle:**

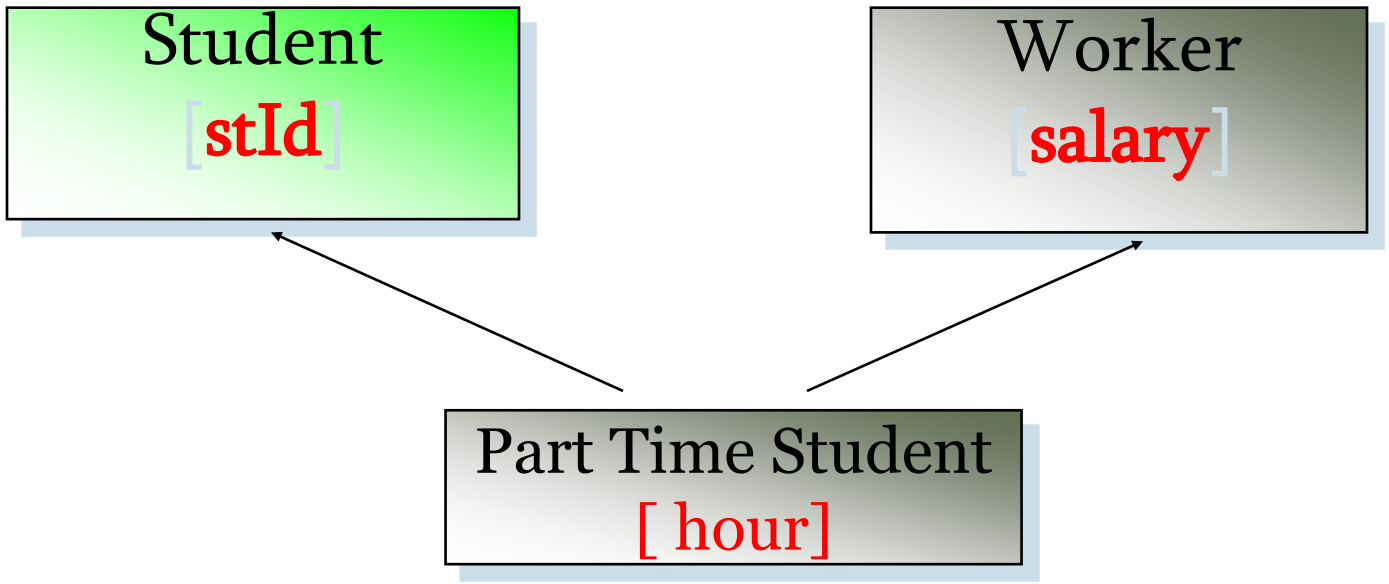
private: int num_wheels, range;
public: vehicle(int n, int w)
void showv()

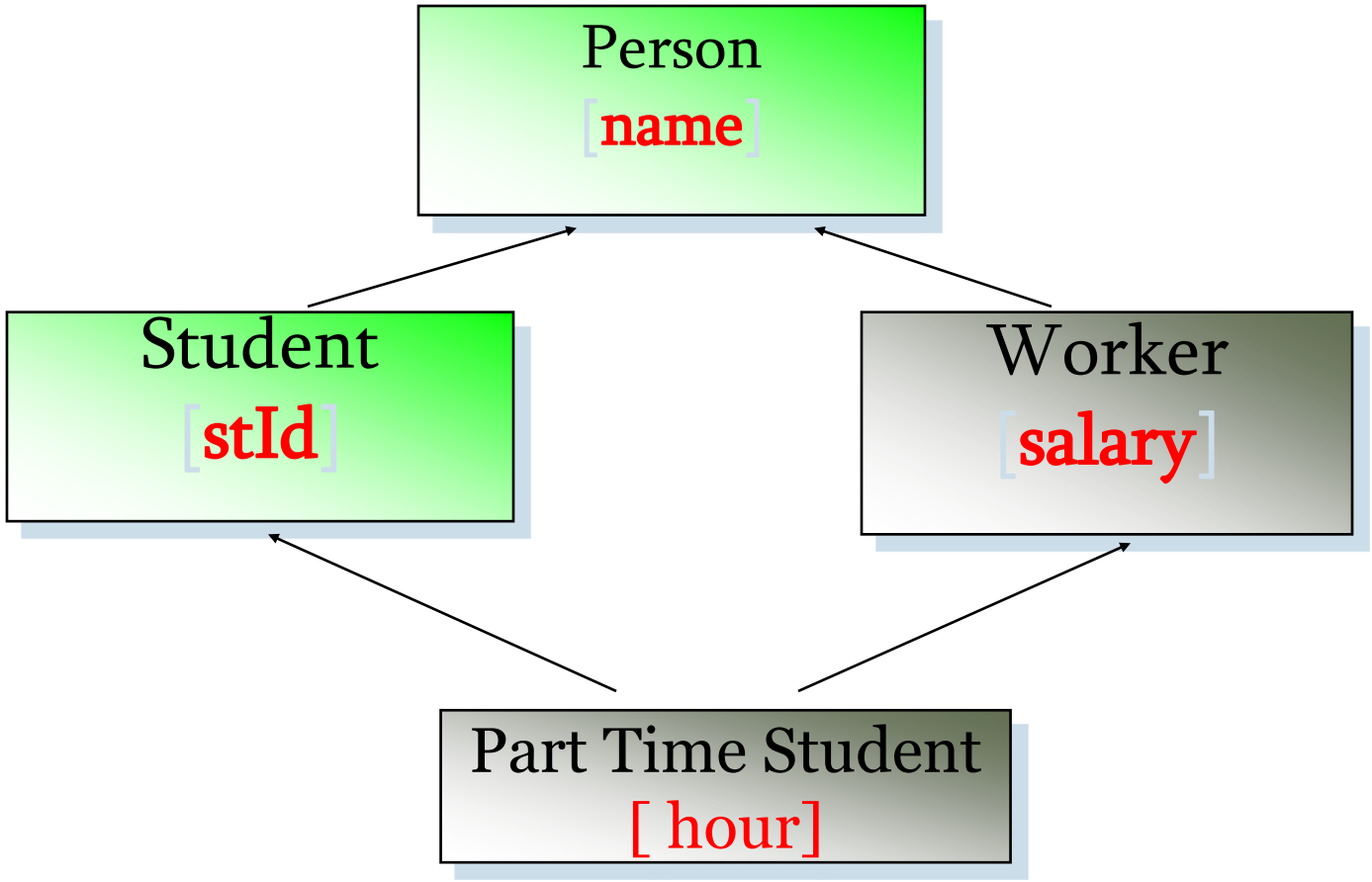
- **class car:**

priavte: int passengers;
public: car(int p, int n, int w)
void show()

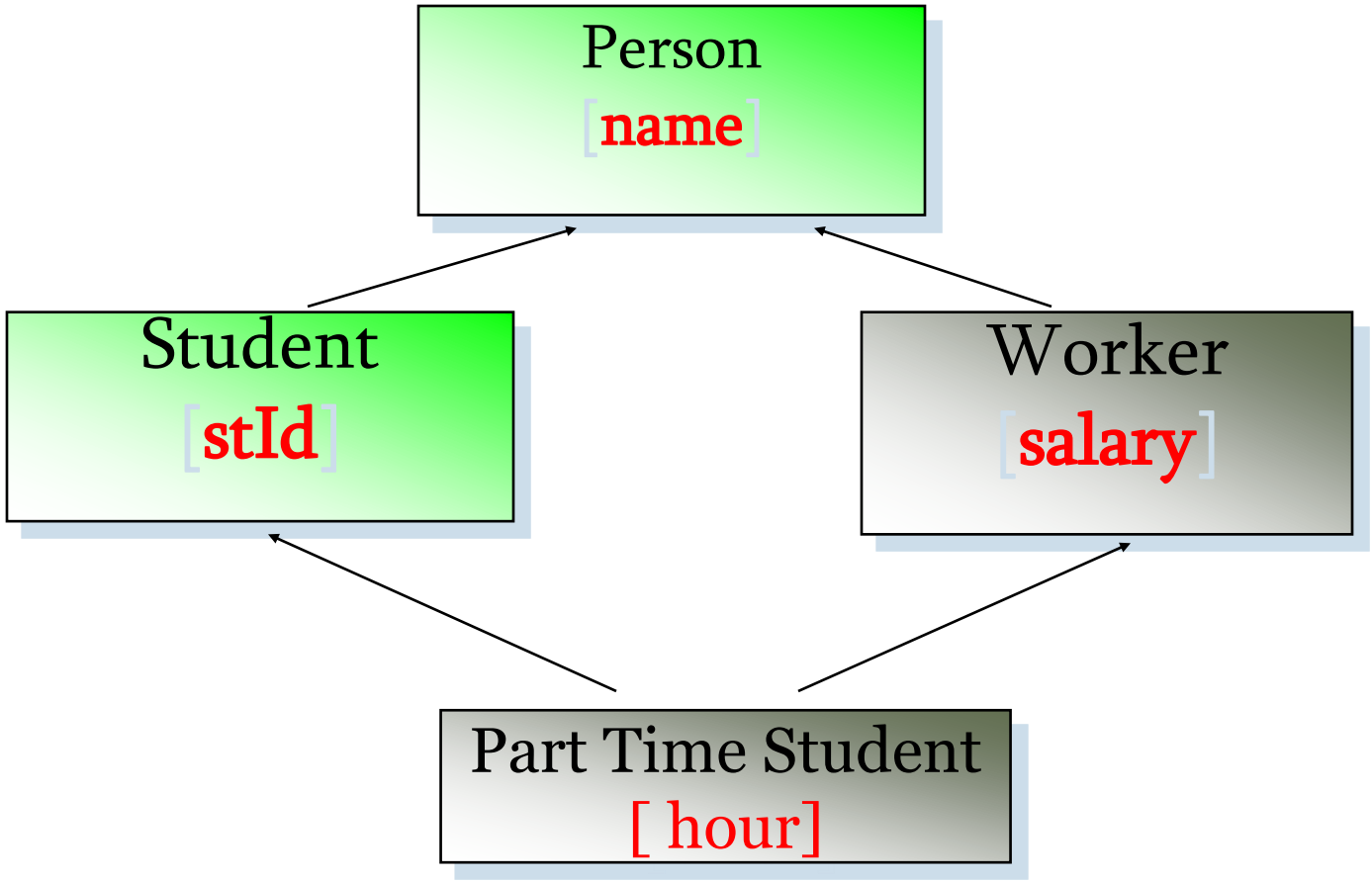
- **class truck:**

private: int loadlimit
public: truck(int l, int n, int w)
void show()


Student
[stId]
Worker
[salary]
Part Time Student
[hour]


Person
[name]
Student
[stId]
Worker
[salary]
Part Time Student
[hour]

Multilevel Multiple Inheritance



Multiple Inheritance – Multiple Copy

```
class Person{
 public : char name[20];
 … … …
};
class Student : public Person{
 … … …
};
class Worker : public Person{
 … … …
};
```

```
class PTstudent : public Student,
                  public Worker {
 public:
        PTstudent (char* n)
        {  strcpy (name,n); }
};
int main(){
 PTstudent PTstudent("ABCD");
}
```

Output:

Multiple Inheritance – Multiple Copy

```
class Person{
 public : char name[20];
 … … …
};
class Student : public Person{
 … … …
};
class Worker : public Person{
 … … …
};
```

```
class PTstudent : public Student,
                  public Worker {
 public:
        PTstudent (char* n)
        {  strcpy (name,n); }
};
int main(){
 PTstudent PTstudent("ABCD");
}
```

Output:

Compilation Error

Multiple Inheritance – Multiple Copy

```
class Person{
 public : char name[20];
 … … …
};
class Student : public Person{
 … … …
};
class Worker : public Person{
 … … …
};
```

```
class PTstudent : public S
                  public W
 public:
        PTstudent (cha  n)
        {  strcpy (name,n); }
};
int main(){
 PTstudent PTstudent("ABCD");
}
```

Output:

Which name to use?

Compilation Error

Virtual base class

- Here, base class Person is inherited by both Student and worker.
- PTStudent inherits Person twice.
  - First, it is inherited by Student
  - Again, it is inherited by Worker
- This causes ambiguity when a member of Person is used by PTStudent
- That's why, C++ includes a mechanism by which only one copy of base class will be included.
- This feature is called virtual base class.

Solution:

```
class Person{
 public : char name[20];
 … … …
};

class Student : virtual public Person{
 … … …
};

class Worker : virtual public Person{
 … … …
};
```

```
class PTstudent : public Student,
                  public Worker {
 public:
PTstudent (char* n){
        strcpy (name,n); }
};

int main(){
 PTstudent PTstudent("ABCD");
}
```

Solution:

```
class Person{
 public : char name[20];
  … … …
};

class Student : virtual public Person{
  … … …
};

class Worker : virtual public Person{
  … … …
};
```

```
class PTstudent : p
                  P
 public:
PTstudent (char* n){
            strcpy (name,n); }
};

int main(){
 PTstudent PTstudent("ABCD");
}
```

OK
only on copy
of name
is included

Virtual base class Example:

```
class base
{
   public: int i;
};
class derived1: virtual public base
{
   public: int j;
};
class derived2: virtual public base
{
   public: int k;
};
```

```
class derived3: public derived1, public
derived2
{
public:
   int product(){return i*j*k;}
};
int main()
{
   derived3 ob;
   ob.i=1;
   ob.j=2;
   ob.k=3;
   cout<<ob.product()<<endl;
}
```

Virtual base class

- When a base class is inherited as virtual by a derived class, the base class still exists within that derived class.
- The only difference between a normal base class and virtual base class occurs when an object inherits the same class more than once.
- In previous example, it perfectly works:

derived1 ob1;

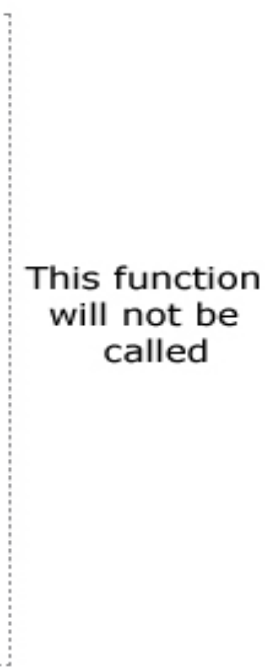
ob1.i=100;

Function Overriding

- Suppose, both base class and derived class have a member function with same name and arguments (number and type of arguments).
- If you create an object of the derived class and call the member function which exists in both classes (base and derived), the member function of the derived class is invoked and the function of the base class is ignored.

Function Overriding: Example

```
class Base
{
... .. ...
public:
  void getData();
  {
    ... .. ...
  }
};

class Derived: public Base
{
  ... .. ...
  public:
    void getData();
    {
    ... .. ...
    }
};

int main()
{
  Derived obj;
  obj.getData();
}
```



Function Overriding

How to access the overridden function in the base class from the derived class?- To access the overridden function of the base class from the derived class, scope resolution operator :: is used.