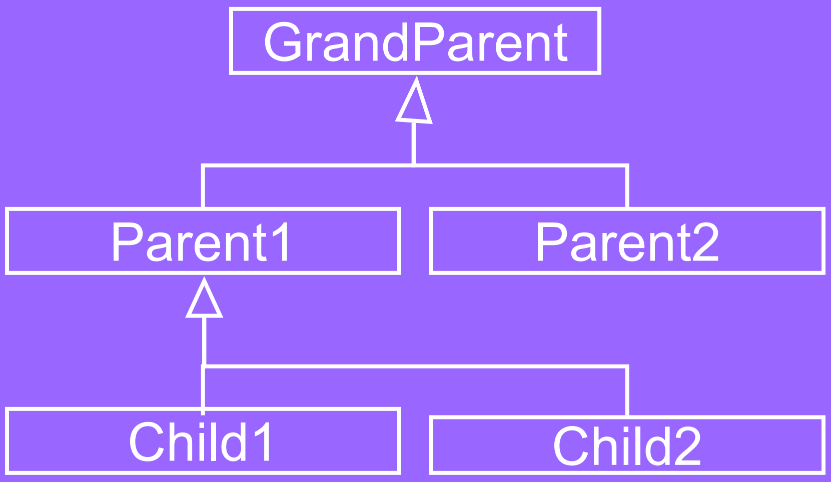
Object-Oriented Programming (OOP)
Lecture No. 26

Hierarchy of Inheritance

- We represent the classes involved in inheritance relation in tree like hierarchy

Example



Direct Base Class

- A direct base class is explicitly listed in a derived class's header with a colon (:)

```
class Child1:public Parent1
...
```

Indirect Base Class

- An indirect base class is not explicitly listed in a derived class's header with a colon (:)
- It is inherited from two or more levels up the hierarchy of inheritance

```
class GrandParent{};
class Parent1:
          public GrandParent {};
class Child1:public Parent1{};
```

Base Initialization

- The child can only perform the initialization of direct base class through *base class initialization list*
- The child can not perform the initialization of an indirect base class through *base class initialization list*

Example

```
class GrandParent{
   int gpData;
public:
   GrandParent() : gpData(0){...}
   GrandParent(int i) : gpData(i){...}
   void Print() const;
};
```

Example

```
class Parent1: public GrandParent{
   int pData;
public:
   Parent1() : GrandParent(),
        pData(0) {...}
};
```

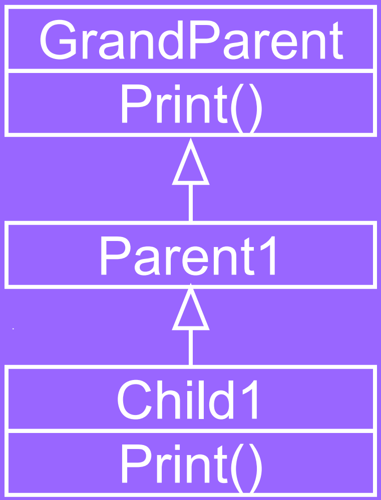
Example

```
class Child1 : public Parent1 {
public:
   Child1() : Parent1()      {...}
   Child1(int i) : GrandParent (i) //Error
   {...}
   void Print() const;
};
```

Overriding

- Child class can override the function of GrandParent class

Example



Example

```
void GrandParent::Print() {
     cout      << "GrandParent::Print"
               << endl;
}

void Child1::Print() {
     cout << "Child1::Print" << endl;
}
```

Example

```
int main(){
   Child1 obj;
   obj.Print();
   obj.Parent1::Print();
   obj.GrandParent::Print();
   return 0;
}
```

Output

- Output is as follows

```
Child1::Print
GrandParent::Print
GrandParent::Print
```

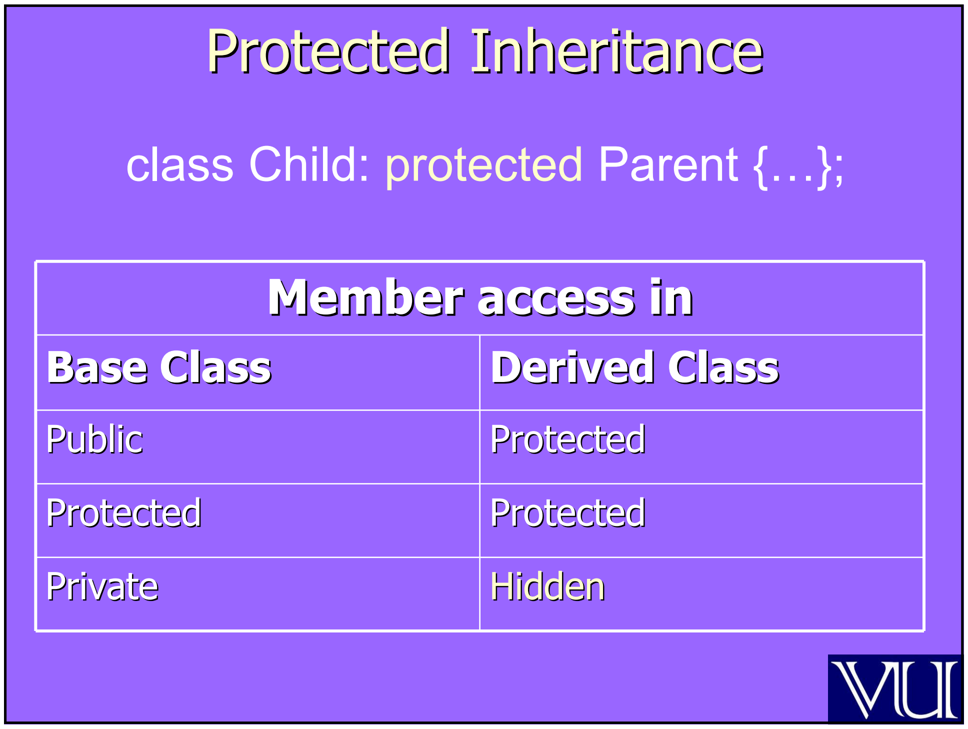
Types of Inheritance

- There are three types of inheritance
  - Public
  - Protected
  - Private
- Use keyword public, private or protected to specify the type of inheritance

Public Inheritance

class Child: public Parent {…};

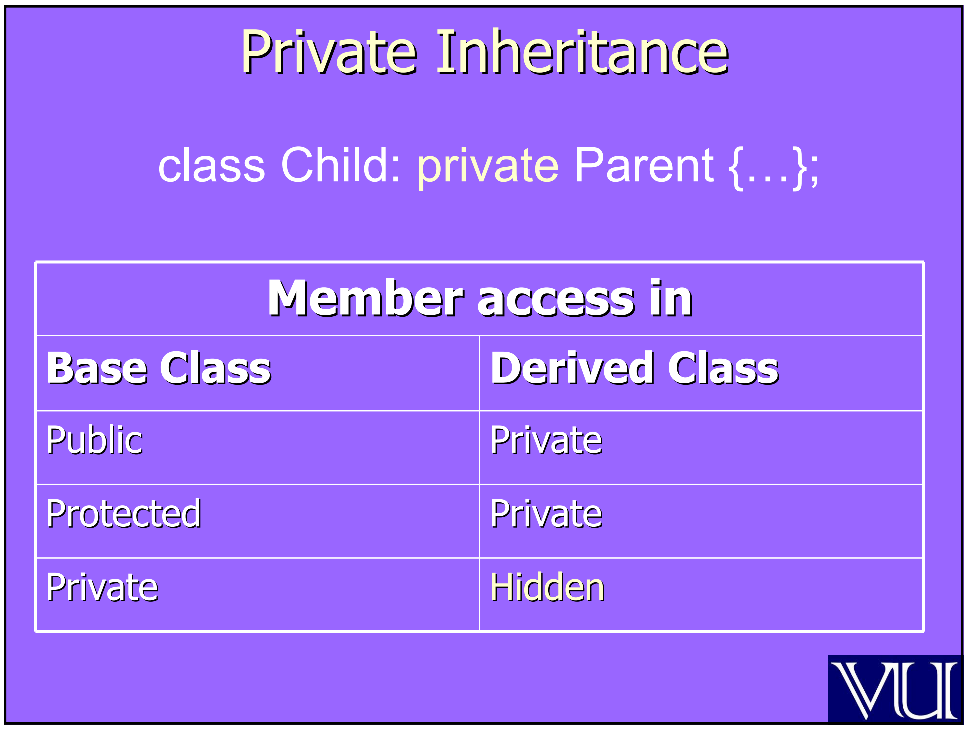
Member access in	
Base Class	**Derived Class**
Public	Public
Protected	Protected
Private	Hidden



Protected Inheritance

class Child: protected Parent {…};

Member access in	
Base Class	**Derived Class**
Public	Protected
Protected	Protected
Private	Hidden



Private Inheritance

class Child: private Parent {…};

Member access in	
Base Class	**Derived Class**
Public	Private
Protected	Private
Private	Hidden

Private Inheritance

- If the user does not specifies the type of inheritance then the default type is private inheritance

```
class Child: private Parent {…}
```

is equivalent to

```
class Child: Parent {…}
```

Private Inheritance

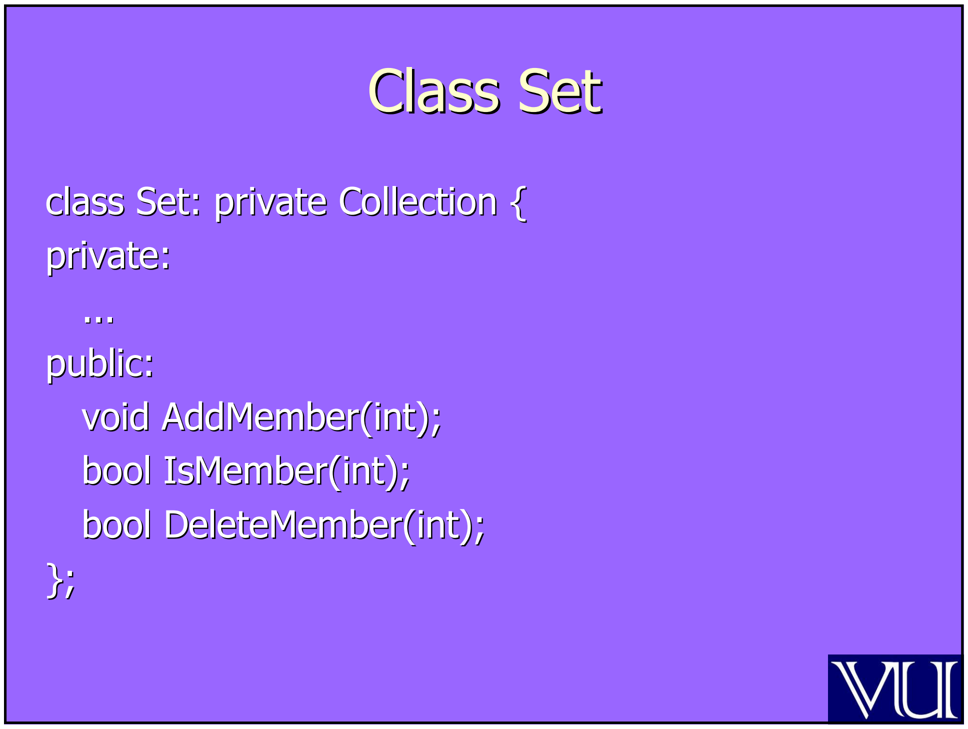
- We use private inheritance when we want to reuse code of some class
- Private Inheritance is used to model "Implemented in terms of" relationship

Example

```
class Collection {
…
public:
   void AddElement(int);
   bool SearchElement(int);
   bool SearchElementAgain(int);
   bool DeleteElement(int);
};
```

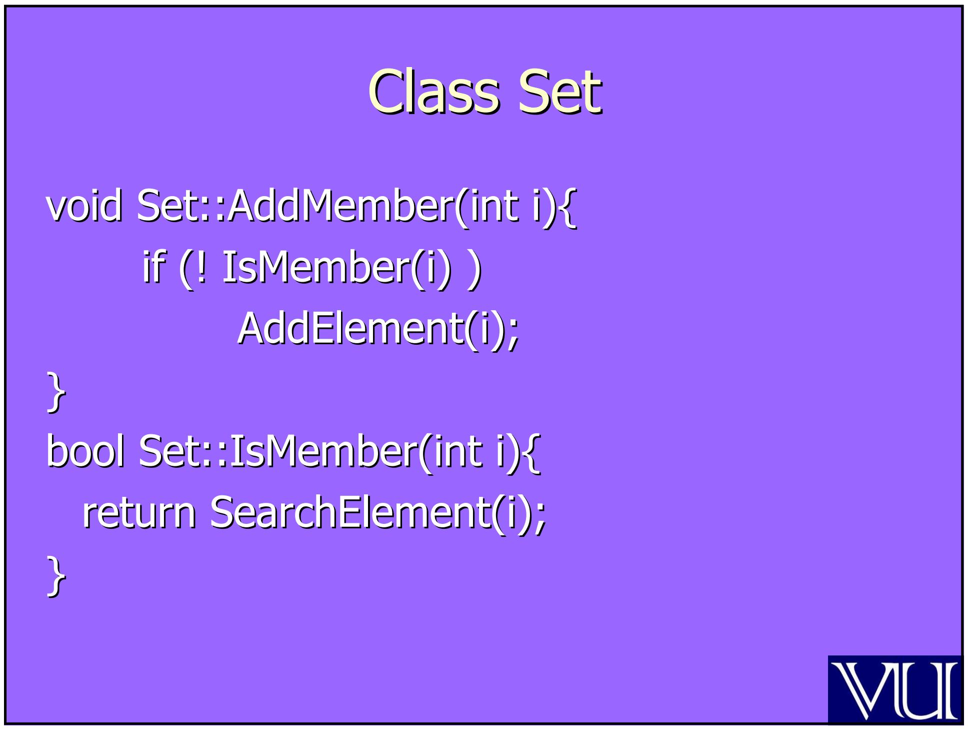
Example

- If element is not found in the Collection the function SearchElement will return false
- SearchElementAgain finds the second instance of element in the collection



Class Set

```
class Set: private Collection {
private:
   ...
public:
   void AddMember(int);
   bool IsMember(int);
   bool DeleteMember(int);
};
```



Class Set

```
void Set::AddMember(int i){
        if (! IsMember(i) )
                AddElement(i);
}
bool Set::IsMember(int i){
   return SearchElement(i);
}
```