Object-Oriented Programming (OOP)
Lecture No. 27

Class Collection

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

Class Set

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

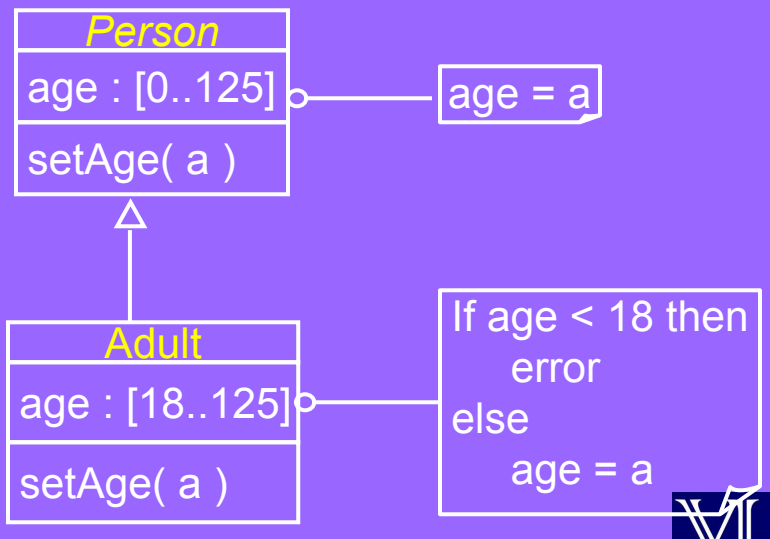
Specialization (Restriction)

- the derived class is behaviourally incompatible with the base class
- Behaviourally incompatible means that base class can't always be replaced by the derived class

Specialization (Restriction)

- Specialization (Restriction) can be implemented using private and protected inheritance

Example – Specialization (Restriction)



Example

```
class Person{
    …
protected:
    int age;
public:
    bool SetAge(int _age){
        if (_age >=0 && _age <= 125) {
            age = _age;
            return true;
        }
        return false;
    }
};
```

Example

```
class Adult : private Person {
    public:
    bool SetAge(int _age){
        if (_age >=18 && _age <= 125) {
            age = _age;
            return true;
        }
        return false;
    }
};
```

Private Inheritance

- Only member functions and friend functions of a derived class can convert pointer or reference of derived object to that of parent object

Example

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


int main(){
    Child cobj;
    Parent *pptr = & cobj;          //Error
    return 0;
}
```

Example

```
void DoSomething(const Parent &);

Child::Child(){
   Parent & pPtr =
               static_cast<Parent &>(*this);
   DoSomething(pPtr);
   //   DoSomething(*this);
}
```

Private Inheritance

- The child class object has an anonymous object of parent class object
- The default constructor and copy constructor of parent class are called when needed

Example

```
class Parent{
public:
    Parent(){
        cout << "Parent Constructor";
    }

    Parent(const Parent & prhs){
      cout << "Parent Copy Constructor";
    }
};
```

Example

```
class Child: private Parent{
public:
     Child(){
          cout << "Child Constructor";
    }

     Child(const Child & crhs)
                              :Parent(crhs){
          cout << "Child Copy Constructor";
    }
};
```

Example

```
int main() {
   Child cobj1;
   Child cobj2 = cobj1;
   //Child cobj2(cobj1);
   return 0;
}
```

Example

- Output:

```
Parent Constructor
Child Constructor
Parent Copy Constructor
Child Copy Constructor
```

Private Inheritance

- The base class that is more then one level down the hierarchy cannot access the member function of parent class, if we are using private inheritance

Class Hierarchy

```
class GrandParent{
public :
   void DoSomething();
};

class Parent: private GrandParent{
   void SomeFunction(){
        DoSomething();
   }
};
```

Example

```
class Child: private Parent
{
public:
   Child() {
        DoSomething();              //Error
   }
};
```

Private Inheritance

- The base class that is more then one level down the hierarchy cannot convert the pointer or reference to child object to that of parent, if we are using private inheritance

Class Hierarchy

```
void DoSomething(GrandParent&);
class GrandParent{
};
class Parent: private GrandParent{
public:
   Parent() {DoSomething(*this);}
};
```

Example

```
class Child: private Parent {
public:
   Child()
   {
       DoSomething(*this);              //Error
   }
};
```

Protected Inheritance

- Use protected inheritance if you want to build class hierarchy using "implemented in terms of"

Protected Inheritance

- If B is a protected base and D is derived class then public and protected members of B can be used by member functions and friends of classes derived from D

Class Hierarchy

```
class GrandParent{
public :
   void DoSomething();
};

class Parent: protected GrandParent{
   void SomeFunction(){
        DoSomething();
   }
};
```

Example

```
class Child: protected Parent
{
public:
   Child()
   {
        DoSomething();
   }
};
```

Protected Inheritance

- If B is a protected base and D is derived class then only friends and members of D and friends and members of class derived from D can convert D* to B* or D& to B&

Class Hierarchy

```
void DoSomething(GrandParent&);

class GrandParent{
};

class Parent: protected GrandParent{
};
```

Example

```
class Child: protected Parent {
public:
   Child()
   {
       DoSomething(*this);
   }
};
```