Object-Oriented Programming (OOP)
Lecture No. 23

Date Class

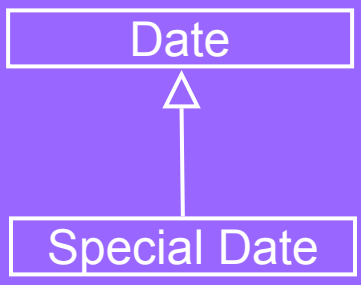
```
class Date{
   int day, month, year;
   static Date defaultDate;
public:
   void SetDay(int aDay);
   int GetDay() const;
   void AddDay(int x);
   …
    static void SetDefaultDate(
        int aDay,int aMonth, int aYear);
```

Date Class

```
    ...
private:
    bool IsLeapYear();
};

int main(){
    Date aDate;
    aDate.IsLeapYear(); //Error
    return 0;
}
```

Creating SpecialDate Class



Creating SpecialDate Class

```
class SpecialDate: public Date{
    …
public:
    void AddSpecialYear(int i){
            …
            if(day == 29 && month == 2
            &&      !IsLeapyear(year+i)){ //ERROR!
                    …
            }
    }
};
```

Modify Access Specifier

- We can modify access specifier "IsLeapYear" from private to public

Modified Date Class

```
class Date{
public:
   ...
   bool IsLeapYear();
};
```

Modified AddSpecialYear

```
void SpecialDate :: AddSpecialYear
                                        (int i){
   ...
   if(day == 29 && month == 2
        &&    !IsLeapyear(year+i)){
        ...
   }
}
```

Protected members

- Protected members can not be accessed outside the class
- Protected members of base class become protected member of derived class in Public inheritance

Modified Date Class

```
class Date{
    …
protected:
        bool IsLeapYear();
};

int main(){
    Date aDate;
    aDate.IsLeapYear(); //Error
    return 0;
}
```

Modified AddSpecialYear

```
void SpecialDate :: AddSpecialYear
                                        (int i){
  ...
  if(day == 29 && month == 2
      &&    !IsLeapyear(year+i)){
      ...
  }
}
```

Disadvantages

- Breaks encapsulation
  - The protected member is part of base class's implementation as well as derived class's implementation

"IS A" Relationship

- Public inheritance models the "IS A" relationship
- Derived object IS A kind of base object

Example

```
class Person {
   char * name;
public: ...
   const char * GetName();
};
class Student: public Person{
   int rollNo;
public: ...
   int GetRollNo();
};
```

Example

```
int main()
{
   Student sobj;
   cout << sobj.GetName();
   cout << sobj.GetRollNo();
   return 0;
}
```

"IS A" Relationship

- The base class pointer can point towards an object of derived class

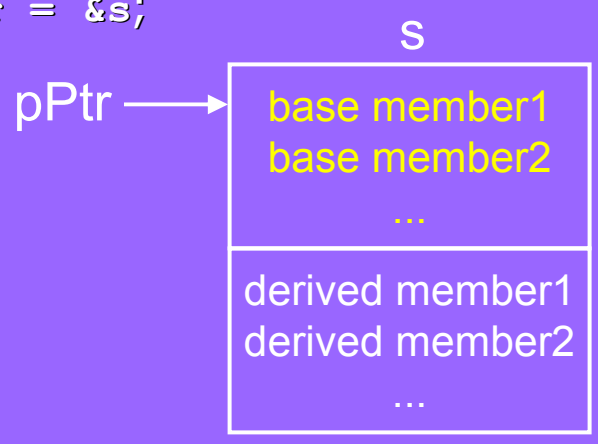
Example

```
int main(){
  Person * pPtr = 0;
  Student s;
  pPtr = &s;
  cout << pPtr->GetName();

  return 0;
}
```

Example

```
pPtr = &s;
```



Example

```
int main(){
  Person * pPtr = 0;
  Student s;
  pPtr = &s;
  //Error
  cout << pPtr->GetRollNo();
  return 0;
}
```

Static Type

- The type that is used to declare a reference or pointer is called its static type
  - The static type of pPtr is Person
  - The static type of s is Student

Member Access

- The access to members is determined by static type
- The static type of pPtr is Person

- Following call is erroneous

```
pPtr->GetRollNo();
```

"IS A" Relationship

- We can use a reference of derived object where the reference of base object is required

Example

```
int main(){
   Person p;
   Student s;
   Person & refp = s;
   cout << refp.GetName();
   cout << refp.GetRollNo(); //Error
   return 0;
}
```

Example

```
void Play(const Person& p){
   cout << p.GetName()
        << " is playing";
}
void Study(const Student& s){
   cout << s.GetRollNo()
        << " is Studying";
}
```

Example

```
int main(){
   Person p;
   Student s;
   Play(p);
   Play(s);
   return 0;
}
```