Inheritance

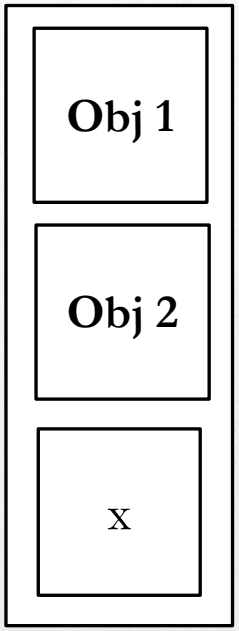
- Inheritance
- Types of Inheritance
- Access Specifier and Inheritance

Containment

- Containment represents “has a” relationship
- Containment Relationship means the use of an object of a class as a member of another class.
- Ex. Birth_Date or joining date as a part of Employee class
- The container relationship brings reusability of code.
- Ex. Already written Date class can be used in Class Employee.

"HAS-A" Type of Relationship

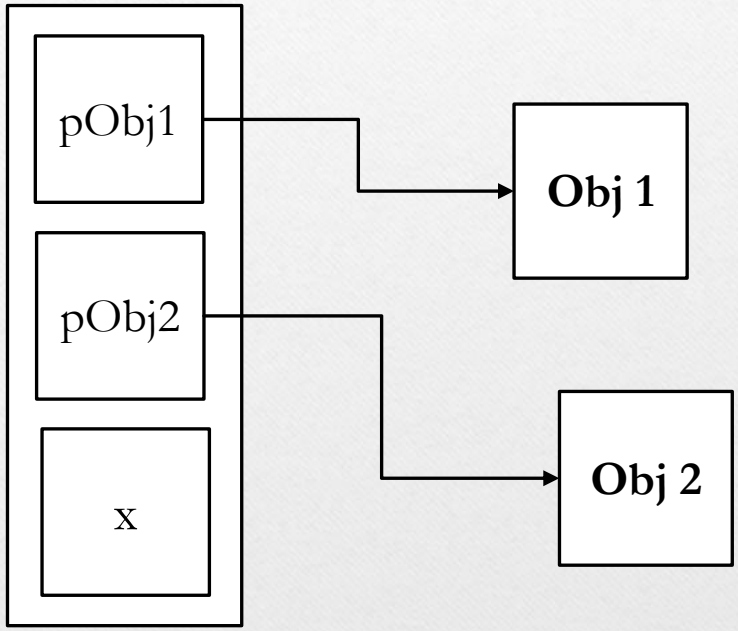
Physical containment



- Also called as tight coupling
- Example

  Car-Engine

Logical containment



- Also called as lose coupling
- Example

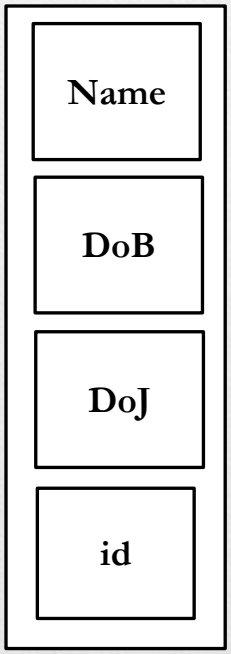
  Car-Documents

"Has-A" Type of Relationship

- Facilitates code reuse
- Already written classes can be used as members of another class
- Example:
  - Already written classes like String and Date can be used as date members of class Person, Employee or Student

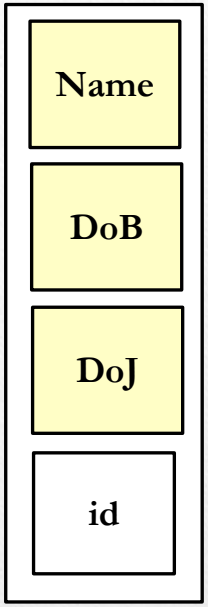
```
class Employee
{
    CString Name;
    Date DoB;
    Date DoJ;
    int Id;
public:
    Employee();
    Employee(char*,int,int,int,
             int,int,int,int);

    ~Employee();
    void Display();
};
```



Object Creation and Destruction

```
class Employee
{
    CString Name;
    Date DoB;
    Date DoJ;
    int Id;
public:
    Employee();
    Employee(char*,int,int,int,
             int,int,int,int);
    void Display();
};
```



Sequence of Constructor

1. CString ()::CString()
2. Date ::Date() for DoB
3. Date ::Date() for DoJ
4. Employee ::Employee()

Sequence of Destructor

1. Employee ::~Employee()
2. Date ::~Date() for DoJ
3. Date ::~Date() for DoB
4. CString ()::~CString()

- Contained objects created first
- Order of creation decided by declarations in container class
- Default constructor gets called
- Object destruction takes place in reverse order

Member Initialization List

```
class Employee
{
    CString Name;
    Date DoB;
    Date DoJ;
    int Id;
public:
    Employee();
    Employee(char*,int,int,int,
             int,int,int,int);

    ~Employee();
    void Display();
};
```

```
Employee::Employee(char* p, int a,int b,int c,
                   int d,int e,int f,int g)
:Name(p),DoB(a,b,c),DoJ(d,e,f),Id(g)
{
}
```

```
Date::Date(int a,int b,int c)
    :d(a),m(b),y(c)
{
}
```

```
CString::CString(char* q)
{
    len=strlen(q);
    p= new char[len+1];
    strcpy(p,q);
}
```

```
void main()
{
    Employee e1;
    e1.Display();
    Employee e2("SomeOne",10,10,1980,10,10,2000,100);
    e2.Display();
}
```

Member Initialization List

- Container object gets all the data from user and distributes it to appropriate contained object and to itself as well
- If not specified, all contained objects get created using default constructor. User input is reassigned later using member functions
- Call appropriate constructor right at the time of creation. This improves performance
- Sequence of member initialization list can not override that of class declaration
- Even built-in data members can be initialized in this list
- Reference members can be initialized only in this list

Delegation

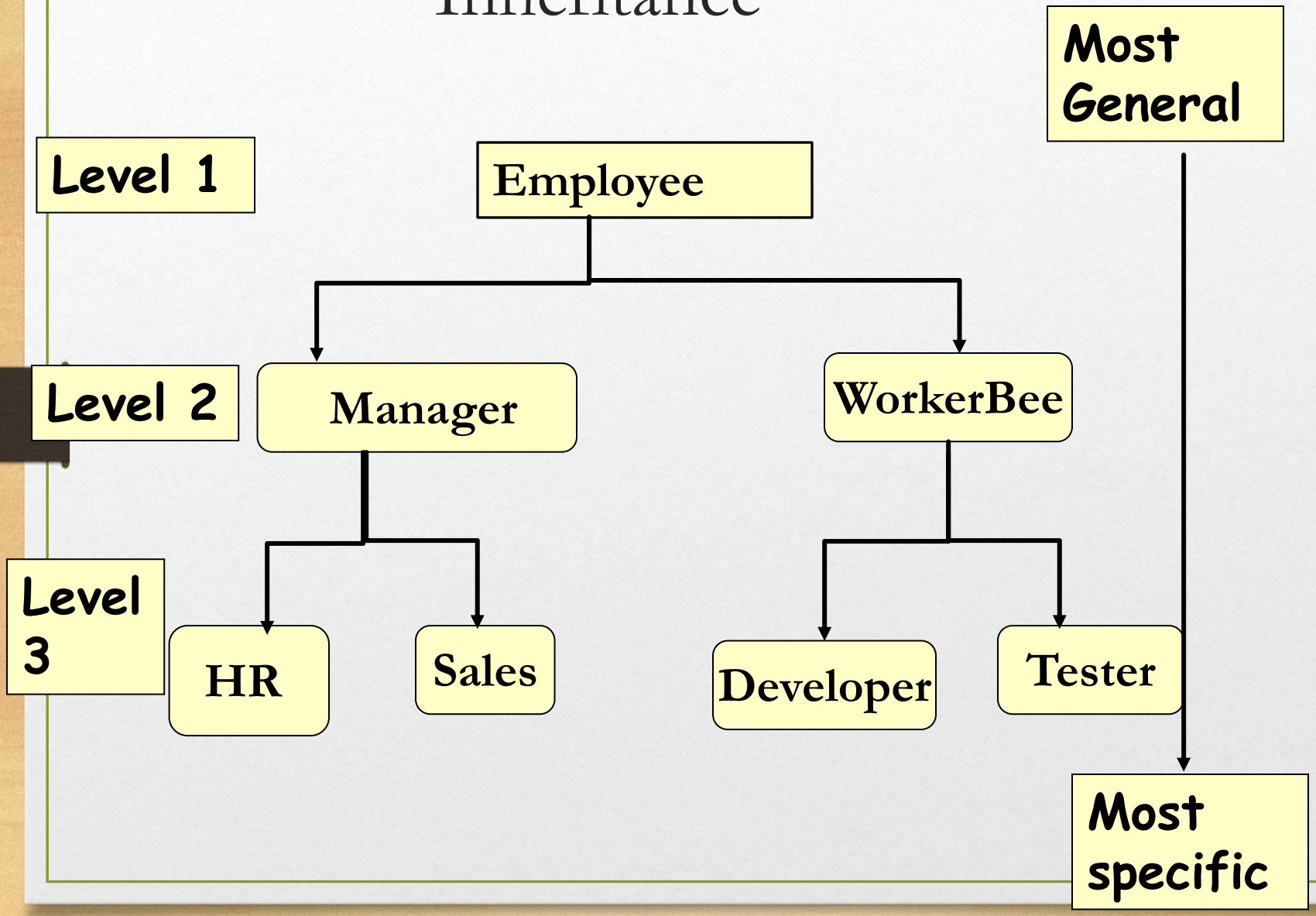
- Container delegates a responsibility to the contained objects by calling appropriate member function

```
void Employee::Display()
{
    Name.Display();
    DoB.Display();
    DoJ.Display();
}
```

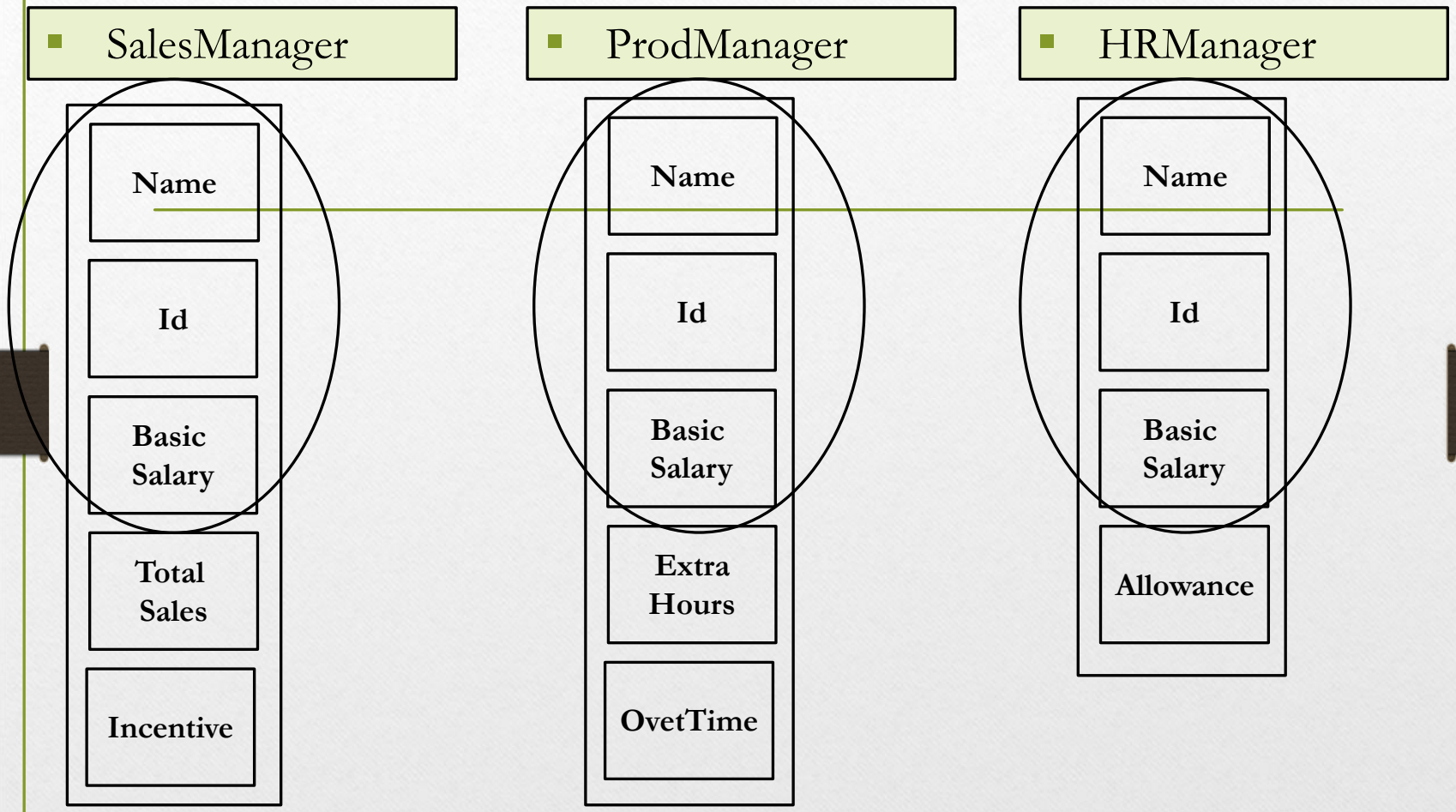
Inheritance

- Extending the feature of a existing class into a new class.
- Inheritance is where one class (child) inherits the members of another class (parent).
- The benefit of inheritance is that the child class doesn't have to redeclare and redefine all the members which it inherits from the parent class. It is therefore a way to re-use code.
- Subclass can have additional specific attributes and methods
- Establishes ‘is-a’ kind of relationship
- Moves down from generalization to specialization.
- Most general at top to most specific at the bottom of inheritance chain

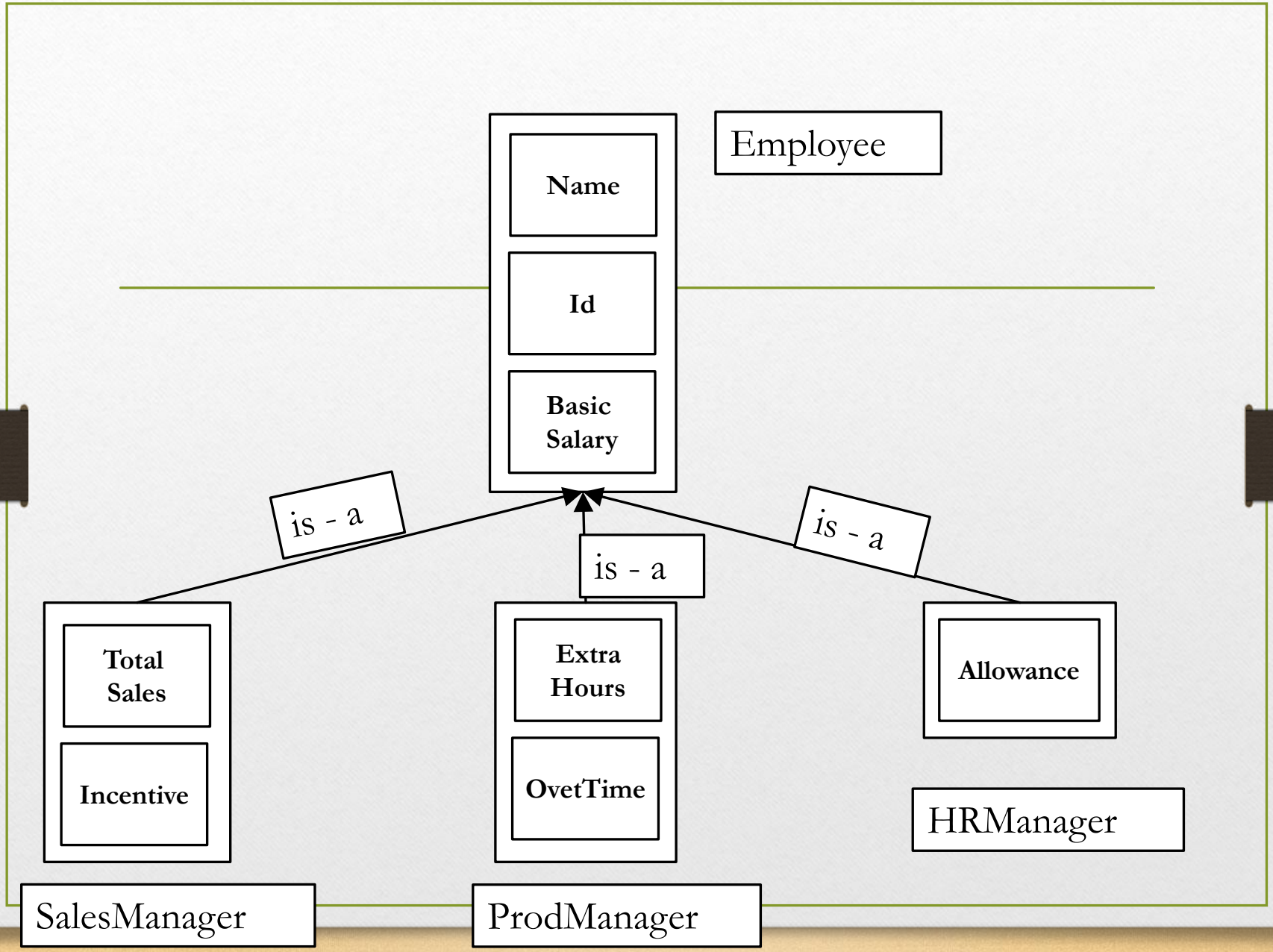
Inheritance



"IS-A" Type of Relationship



"Is-A" Type of Relationship



Syntax For Inheritance

- Syntax:

  class *DerivedClassName* : access-level *BaseClassName*

  where

  - access-level specifies the type of derivation
    - private by default, or
    - public
    - protected
- Any class can serve as a base class
  - Thus a derived class can also be a base class

"IS-A" Type of Relationship

```
class Employee
{
    char Name[50];
    int Id;
    float BasicSal;
public:
    Employee();
    Employee(char*,int,float);

    ~Employee();
    void Display();
};
```

```
class SalesManager: public Employee
{
    float TotalSales;
    float Incentive;
public:
    SalesManager();
    SalesManager(char*,int,float,
                 float,float);
    ~SalesManager();

    void Display();
};
```

```
class ProdManager: public Employee
{
    float ExtraHrs;
    float OverTime;
public:
    ProdManager();
    ProdManager(char*,int,float,
                float,float);
    ~ProdManager();

    void Display();
};
```

```
class HrManager: public Employee
{
    float Allowance;
public:
    HrManager();
    HrManager(char*,int,float,
              float);
    ~HrManager();

    void Display();
};
```

Creation and Destruction

- Base class part is created first
- Order of creation decided by declarations in case of multiple inheritance
- If not specified, default constructor of base class gets called
- Object destruction takes place in reverse order

Code Reuse

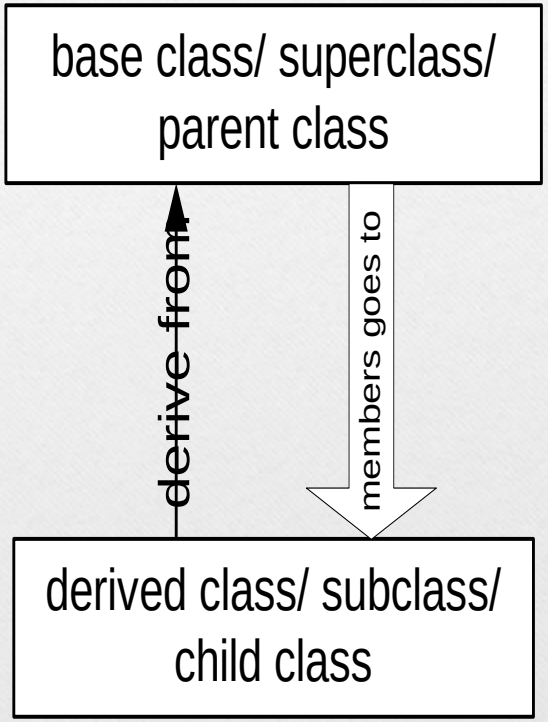
- Data as well as functions of base class available to derived class
- Inherited function can be hidden by redefining it in derived class. This is called as function hiding
- Derived class definition can call base class version of the same function using scope resolution operator

```
void Employee::Display()
{
    cout<<Name<<"\n"<<Id<<"\n"<<BasicSal<<endl;
}
```

```
void SalesManager::Display()
{
    Employee::Display();
    cout<<TotalSales<<"\n"<<Incentive<<endl;
}
```

What to Inherit??

- In principle, every member of a base class is inherited by a derived class
  - just with different access permission



- Two levels of access control over class members
  - class definition
  - inheritance type

Access Specifier & Inheritance

Base class member access specifier	Type of Inheritance		
	Public inheritance	**Protected Inheritance**	**Private Inheritance**
public	Public in derived class. Can be accessed directly by any non-static member functions, friend functions, and non-member functions.	Protected in derived class. Can be accessed directly by any non-static member functions, friend functions.	Private in derived class. Can be accessed directly by any non-static member functions, friend functions.
Protected	Protected in derived class. Can be accessed directly by any non-static member functions, friend functions.	Protected in derived class. Can be accessed directly by any non-static member functions, friend functions.	Private in derived class. Can be accessed directly by any non-static member functions, friend functions.
private	Hidden in derived class. Can be accessed directly by non-static member functions, friend functions through public or protected member function of base class.	Hidden in derived class. Can be accessed directly by non-static member functions, friend functions through public or protected member function of base class.	Hidden in derived class. Can be accessed directly by non-static member functions, friend functions through public or protected member function of base class.