Containment and Inheritance

Contents

- Define a containment and state its rules.
- Implement containment
- Need of member initializer list.
- Define inheritance.
- Implement inheritance.
- Define function overriding.

Containment

- Containment represents ‘ has ’ or ‘ is a part of ’ relationship.
- One object added as part of another object.
- It is not compulsory in containment that object should be of same type.
- For e.g.

  car has engine.

  or

  engine is a part of car.

Implement containment

```
class cEmployee
{
        protected:
                int eid;
                float esalary;
                cString name;   //object of class cString
                cDate dob;      //object of class cDate
        public:
                cEmployee()
                {
                        id=1;
                        salary=10000;
                }
};
```

Implement parameterized constructor

```
cEmployee :: cEmployee(int id , int sal , char* nm , int  d ,
                                                    int m , int y)
{
        eid=id;
        esalary=sal;
        name=cString(nm);
        date=cDate(d,m,y);
}
int main()
{
        cEmployee  e1(1 , 40000 , “Ajit” , 14 , 09 , 1994 );
}
```

Constructor calling for cEmployee

- cEmployee is invoked first.
- cString default constructor.
- cDate default constructor.
- cString parameterized constructor.
- cDate parameterized constructor.

- Finally cEmployee is executed.
- So here total constructors are called.

Need of member initializer list

- Here unnecessary default contractors are called.
- To reduce calling of default constructors we need to use member initializer list.
- By using this only parameterized constructors are called.

Need of member initializer list: example

```
cEmployee :: cEmployee(int id , float  sal , char *nm , int d , int m , int y ) : name(nm) , dob(d , m , y)
{
        eid=id;
        esalary=sal;
}
```

Constructors calling Sequence:

cString( ) -> cDate( ) -> cEmplyee( )

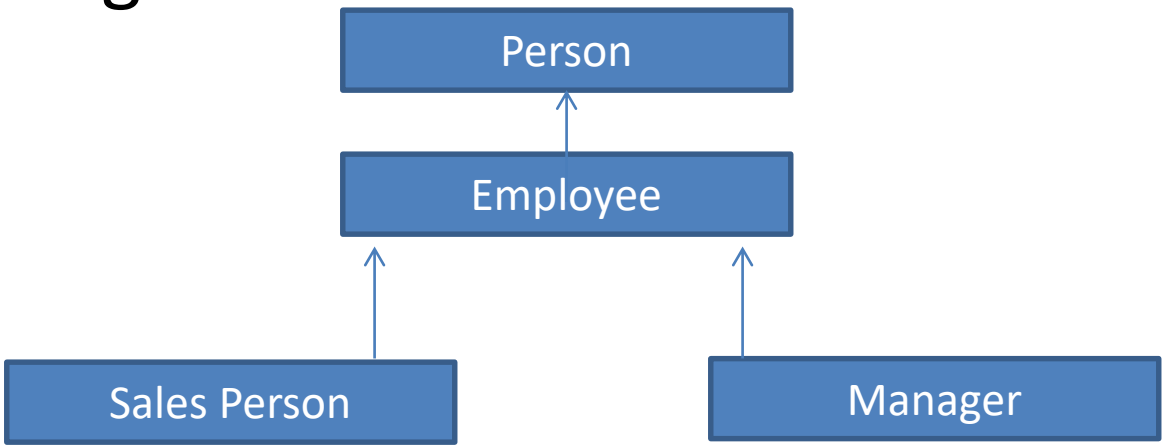
cEmployee accept()

```
void cEmployee :: accept()
{
        cout<<“Enter the id\n”;
        cin>>eid;
        cout<<“Enter the salary\n”;
        cin>>esalary;

        name.accept();
        dob.accept();
}
```

Inheritance

- One object can extends or acquires properties form another object.
- Represents ‘ is a ’ relationship.
- Provides reusability and extensibility.
- For e.g.



Inheritance Syntax

```
class base-class-name
{
        //body of base class
};
class derivedClass : accessSpecifier baseClassName
{
        //body of derived class
};
```

Implementing cSalesPerson class

```
class cSalesPerson : public cEmployee
{
        float sales;
        float comm;
        public:
        cSalesPerson();
        cSalesPerson(int ,float , char* , int , int , int , float , float);
        void accept();
        void display();
};
```

Constructors for cSalesPerson class

```
cSalesPerson :: cSalesPerson()   //default
{
        sales=0;
        comm=0;
}

cSalesPerson :: cSalesPerson(int id , float sal , int d , int m , int y
,float s , float c) : cEmployee( id , sal , d , m , y )
{
        sales=2000;
        comm=1000;
}
```

Accept function for cSalesPerson class

```
void cSalesPerson :: accept( )
{
        cEmployee::accept();     //called base class fun

        cout<<“Enter the sales and comm\n”;
        cin>>sales>>comm;
}
```

Lab Assignments

- We have already implemented cEmployee class. Now we have to implement derived classes for this class
    1. cSalesPerson class
        a. data members: sales and comm.
        b. Constructors.
        c. accept() and display() functions.
    2. cManager class
        a. data members: petrol and food allowance.
        b. Constructors.
        c. accept() and display() functions.