Lab3 Object-Oriented Programming: Inheritance

Objectives

1. To create classes by inheriting from exiting classes.
2. The notations of base classes and derived classes and the relationships between them.
3. The order in which objects were constructed and destructed in inheritance hierarchies.
4. The initial in heritance.
5. The difference between public, protected and private member access specifier.
6. The difference between public, protected and private inheritance.
7. The inheritance, add and hide of class member functions.
8. The translation between base class and derived class.

Experiments

Ex 1:(The construction and destroying of objects in heritance)

1) To create a base class as following:

```
class MyBase1 {
public:
    MyBase1(){ cout << "…BaseClass1 Object is created!"<< end; }
    ~MyBase1(){ cout << "…BaseClass1 Object is destroyed!"<< end; }
}
```

2) To create a derived class from MyBase1 with public inheritance and analyze the result.

```
class Myderived1 : public MyBase1 {
public:
    MyDerived1()
    { cout << "…First layer derived Object is created!"<< end; }
    ~MyDerived1()
    { cout << "…First layer derived Object is Destroyed!"<< end; }
}
class Myderived11 : public MyDerived1 {
public:
    MyDerived11()
```

```
	{ cout << "…Second layer derived Object is created!"<< end; }
	~MyDerived11()
	{ cout << "…Second layer derived Object is destroyed!"<< end; }
}
int main()
{
	MyBase1 a;
	MyDerived1 b;
	MyDerived11 c;
}
```

3) To create a base class as following:

```
class MyBase2 {
	MyBase1 a1;
public:
	MyBase2()
	{ cout << "…BaseClass2 Object is created!"<< end; }
	~ MyBase2()
	{ cout << "…BaseClass2 Object is destroyed!"<< end; }
}
```

4) To create a derived class from MyBase2 with public inheritance and analyze the result.

```
class Myderived1 : public MyBase2 {
	MyBase1 a1;
public:
	MyDerived1()
	{ cout << "…First layer derived Object is created!"<< end; }
	~MyDerived1()
	{ cout << "…First layer derived Object is Destroyed!"<< end; }
}
class Myderived11 : public MyDerived1 {
public:
	MyDerived11()
	{ cout << "…Second layer derived Object is created!"<< end; }
	~MyDerived11()
	{ cout << "…Second layer derived Object is destroyed!"<< end; }
}
int main()
{
	MyBase2 a;
	MyDerived1 b;
	MyDerived11 c;
}
```

Ex 2: The initial of objects in heritance

1) To create two classes as following and analyze the result

```
class MyBase31 {
    int a, b, c;
public:
    MyBase31(int x, int y, int z) :a(x), b(y), c(z)
    {
        cout << "…BaseClass31 Object is created!"<< end;
        cout << a << " " << b << " " << c << endl;
    }
    ~ MyBase31(){ cout << "…BaseClass31 Object is destroyed!"<< end; }
}
class MyBase32 {
    int a, b, c;
public:
    MyBase32(int x, int y, int z)
    {
        cout << "…BaseClass32 Object is created!"<< end;
        cout << a << " " << b << " " << c << endl;
        a=x, b=y, c=z;
        cout << a << " " << b << " " << c << endl;
    }
    ~ MyBase32(){ cout << "…BaseClass32 Object is destroyed!"<< end; }
}
int main()
{
    MyBase31 a(1,2,3);
    MyBase32 b(4,5,6);
}
```

2) To create some derived classes as following and analyze the result

```
class MyDerived1 : public MyBase31 {
    MyBase31 a(5,6,7);
    int c;
public:
    MyDerived1(int x) : c(x), MyBase31(x,8,9)
    {
        cout << "…Base Object has been created!" << endl;
        cout << "…Member Object has been created! " << a.x << " " << a.y << " "
<< a.z << endl;
```

```
        cout << "…Derived Object is created! "<< c << endl;
    }
}
int main()
{
    MyDerived1 b(88);
}
```

Ex 3: The access properties in inheritance

1) To create a base class as following:

```
class MyBase3 {
    int x;
    fun1() { cout << “MyBase3---fun1()” << endl; }
protcted:
    int y;
    fun2() { cout << “MyBase3---fun2()” << endl; }
public:
    int z;
    MyBase(int a, int b, int c) {x=a; y=b; z=c;}
    int getX(){cout << “MyBase3---x:” << endl; return x;}
    int getY(){cout << “MyBase3---y:” << endl; return y;}
    int getZ(){cout << “MyBase3---z:” << endl; return z;}
    fun3() { cout << “MyBase3---fun3()” << endl; }
}
```

2) To create a derived classes from MyBase3 with public inheritance and analyze the result.

```
class MyDerived1 : public MyBase3 {
    int p;
public:
    MyDerived1(int a) : p(a)
        int getP(){cout << “MyDerived---p:” << endl; return p;}
    int disply()
    {
        cout << p << “ “ << x << “ “ << y << “ “ << z << “ “ << endl
            << fun1( ) << endl << fun2() << endl << fun3() << endl;
    }
}
int main()
{
    MyDerived1 a(3);
    a.disply();
    cout << a.x << “ “ << a.p << “ “ << a.y << “ “ << a.z << endl;
    cout << a.getX() << “ “ << a.getP() << “ “ << a.getY() << “ “ << a.getZ() <<
endl;
}
```

3) To create a derived classes from MyBase3 with private inheritance and analyze the result.

```
class MyDerived2 : private MyBase3 {
    int p;
public:
    MyDerived2(int a) : p(a)
        int getP(){cout << "MyDerived---p:" << endl; return p;}
    int disply()
    {
        cout << p << " " << x << " " << y << " " << z << " " << endl
            << fun1( ) << endl << fun2() << endl << fun3() << endl;
    }
}
class MyDerived21 : public MyDerived3 {
    int p;
public:
    MyDerived21(int a) : p(a)
        int getP(){cout << "MyDerived21---p:" << endl; return p;}
    int disply1()
    {
        cout << p << " " << x << " " << y << " " << z << " " << endl;
    }

}
int main()
{
    MyDerived2 a(3);
    MyDerived21 b(6);
    a.disply();
    cout << a.x << " " << a.p << " " << a.y << " " << a.z << endl;
    cout << a.getX() << " " << a.getP() << " " << a.getY() << " " << a.getZ() <<
endl;
    b.disply1();
}
```

4) To create a derived classes from MyBase3 with protected inheritance and analyze the result.

```
class MyDerived3 : protected MyBase3 {
    int p;
public:
    MyDerived3(int a) : p(a)
        int getP(){cout << "MyDerived---p:" << endl; return p;}
    int disply()
    {
        cout << p << " " << x << " " << y << " " << z << " " << endl
            << fun1( ) << endl << fun2() << endl << fun3() << endl;
    }
}
class MyDerived31 : public MyDerived3 {
    int p;
public:
    MyDerived31(int a) : p(a)
        int getP(){cout << "MyDerived31---p:" << endl; return p;}
    int disply1()
    {
        cout << p << " " << x << " " << y << " " << z << " " << endl;
    }
}
int main()
{
    MyDerived3 a(3);
    MyDerived31 b(6);
    a.disply();
    cout << a.x << " " << a.p << " " << a.y << " " << a.z << endl;
    cout << a.getX() << " " << a.getP() << " " << a.getY() << " " << a.getZ() <<
endl;
    b.disply1();
}
```

5) To analyze the result

```
class MyBase {
public:
    void f1(){ cout << "…MyBase f1-----!" << endl; }
    void f2(){ cout << "…MyBase f2-----!" << endl; }
}
class MyDerived : public MyBase {
public:
    void f2(){ cout << "…MyDerived f2-----!" << endl; }
    void f22(){ MyBase::f2(); cout << "…MyDerived f2-----!" << endl; }
    void f3(){ cout << "…MyDerived f3-----!" << endl; }
}
int main()
```

```
{
      MyDerived a;
      a.f1(); a.f2(); a.f3(); a.f22();
}
```

Ex 4: The translation between base class and derived class.

1)To create a base class as following:

```
class MyBase {
int x;
public:
MyBase(int a):x(a);
int getX(){ cout << “” << endl; return x; }
}
```

2)To create a derived class as following:

```
class MyDerived : public MyBase {
int y;
public:
MyDerived(int a):y(a),MyBase(a+4);
int getY(){ cout << “” << endl; return Y; }
}
```

3)To create a test program as following and analyze the result.

```
int main()
{
MyBase a(2), *p = a,;
MyDerived b(4), *q=b;
MyBase &c = a;
MyBase &d = b;
cout << a.getX() << “ ” << p-> getX() << endl;
cout << b.getY() << “ ” << q-> getY() << b.getX() << “ ” << q-> getX() << endl;
a = b;
cout << a.getX() << “ “ << a.getY() << endl;
p = q;
cout << p->getX() << “ “ << p->getY() << endl;
cout << c.getX() << “ “ << d.getX() << “ “ << d.getY() << endl;
b = a;
cout << b.getX() << “ ” << b.getY() << endl;
}
```

Ex 5: Construction and Composition

Implement Date class and FinalTest class, and make the main function output correctly. All data members should be private.
Tips:
(1) Data validation is not required.
(2) It is only required to implement the necessary member functions.
(3) Interface and implementation are not necessarily separated.

```
int main()
{
    FinalTest item1("C++ Test", Date(2014,6,2));
    item1.print();
    FinalTest item2("Java");
    item2.print();
    item2.setDue(Date(2014,6,10));
    item2.print();
}
```

```
Title: C++ Test
Test Date: 2014-6-2
Title: Java
Test Date: 2014-1-1
Title: Java
Test Date: 2014-6-10
```

Ex 6: Inheritance, Constructor and Initializer

Design and implement a hierarchical class structure, according to the following requirements.

- Shape is a base class.
- Classes Circle, Triangle and Rectangle are directly inherited from shape.
- Square is directly inherited from Rectangle.
- Provide constructors and destructor for each class.
- Each object must include at least one data member named id (string).
- Objects of derived classes should contain some necessary data members to determine their position and area, such as centerofcircle(circle), lefttop(circle), rightbottom(circle), radius etc.
- Objects of class Square have one special method named incircle. This method can create and return the inscribed circle obejcet(circle) of the corresponding Square object.
- Each object provides area()function to calculate the area of an shape object and print()function to display all information of an object such as radius, width, length, area and incircle.
- Use Initializers to initialize data members of base class and composition objects.

Lab4 Object-Oriented Programming: Polymorphism

Objectives

1. What polymorphism is, how it makes programming more convenient, and how it makes systems more extensible and maintainable.
2. To declare and use virtual functions to effect polymorphism.
3. The distinction between abstract and concrete classes.
4. To declare pure virtual functions to create abstract classes.
5. How C++ implements virtual functions and dynamic binding "under the hood."
6. How to use virtual destructors to ensure that all appropriate destructors run on an object.

Experiments

Ex 1:(12.12, Payroll System Modification)

1. Description of the Problem

(Payroll System Modification) Modify the payroll system of Figs. 12.9-12.17 to include private data member birthDate in class Employee. Use class Date from Figs. 10.6-10.7 to represent an employee's birthday. Assume that payroll is processed once per month. Create a vector of Employee references to store the various employee objects. In a loop, calculate the payroll for each Employee (polymorphically), and add a $100.00 bonus to the person's payroll amount if the current month is the month in which the Employee's birthday occurs.

2. Problem-Solving Tips

1) Tips: how to get system time

Method 1:

```
#include <windows.h>
int main()
{
  SYSTEMTIME systm;
  GetLocalTime(&systm);
  cout<<systm.wYear<<"-"<<systm.wMonth<<"-"<<systm.wDay<<" "<<
systm.wHour<<":"<<systm.wMinute<<":"<<systm.wSecond;
  return 0;
}
```

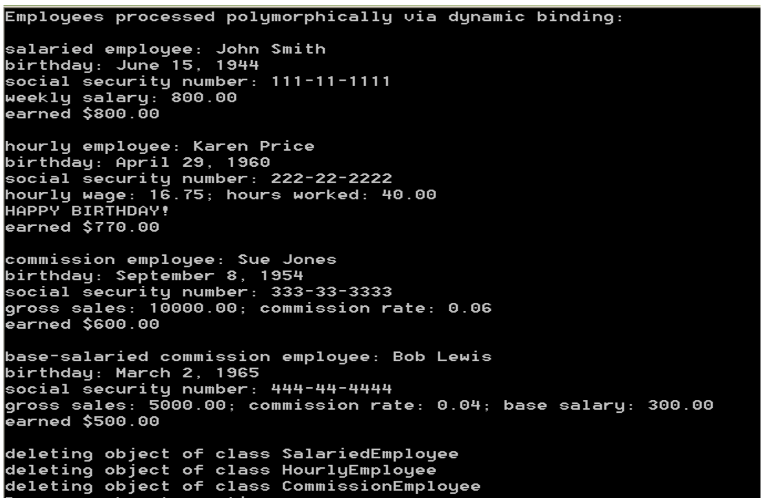
Method 2:

```
#include <iostream>
#include <ctime>
using namespace std;
int main()
{
    time_t nowtime;
    struct tm* ptm;
    time(&nowtime);
    ptm = localtime(&nowtime);
    cout<<ptm->tm_year + 1900<<"-"<<ptm->tm_mon + 1
    <<"-"<<ptm->tm_mday<<" "<<ptm->tm_hour<<":"
    <<ptm->tm_min<<":"<<ptm->tm_sec;
    return 0;
}
```

2) main function

Modify the main functions of Figs 12.17, 12.19.

3. Results example



```
Employees processed polymorphically via dynamic binding:

salaried employee: John Smith
birthday: June 15, 1944
social security number: 111-11-1111
weekly salary: 800.00
earned $800.00

hourly employee: Karen Price
birthday: April 29, 1960
social security number: 222-22-2222
hourly wage: 16.75; hours worked: 40.00
HAPPY BIRTHDAY!
earned $770.00

commission employee: Sue Jones
birthday: September 8, 1954
social security number: 333-33-3333
gross sales: 10000.00; commission rate: 0.06
earned $600.00

base-salaried commission employee: Bob Lewis
birthday: March 2, 1965
social security number: 444-44-4444
gross sales: 5000.00; commission rate: 0.04; base salary: 300.00
earned $500.00

deleting object of class SalariedEmployee
deleting object of class HourlyEmployee
deleting object of class CommissionEmployee
```

Ex 2:

1. Description of the Problem

(Shape Hierarchy) Implement the Shape hierarchy designed in Exercise 11.7 (which is based on the hierarchy in Fig. 11.3).

Each TwoDimensionalShape should contain function getArea to calculate the area of the two-dimensional shape. Each ThreeDimensionalShape should have member functions getArea and getVolume to calculate the surface area and volume of the three-dimensional shape, respectively.

Create a program that uses a vector of Shape pointers to objects of each concrete class in the hierarchy. The program should print the object to which each vector element points. Also, in the loop that processes all the shapes in the vector, determine whether each shape is a TwoDimensionalShape or a ThreeDimensionalShape. If a shape is a TwoDimensionalShape, display its area. If a shape is a ThreeDimensionalShape, display its area and volume.

2. Requirements

(1) Define at least 5 classes.

(2) Define member functions for each class, the following functions must be included: constructor, destructor, virtual getArea function

(3) main function:

vector declaration

test the getArea function

Ex 3:(12.14 Polymorphic Banking Program Using Account Hierarchy)

(Polymorphic Banking Program Using Account Hierarchy) Develop a polymorphic banking program using the Account hierarchy created in Exercise 11.10.

Create a vector of Account pointers to SavingsAccount and CheckingAccount objects. For each Account in the vector, allow the user to specify an amount of money to withdraw from the Account using member function debit and an amount of money to deposit into the Account using member function credit.

As you process each Account, determine its type. If an Account is a SavingsAccount, calculate the amount of interest owed to the Account using member function calculateInterest, then add the interest to the account balance using member function credit.

After processing an Account, print the updated account balance obtained by invoking baseclass member function getBalance.