Computer Programming
Chapter 12: Inheritance - Part 1

Hung-Yu Wei
Department of Electrical Engineering
National Taiwan University

[Concept] Inheritance

- Create classes by inheriting from existing classes
  - Base class
  - Derived class
- Expand usage of existing classes

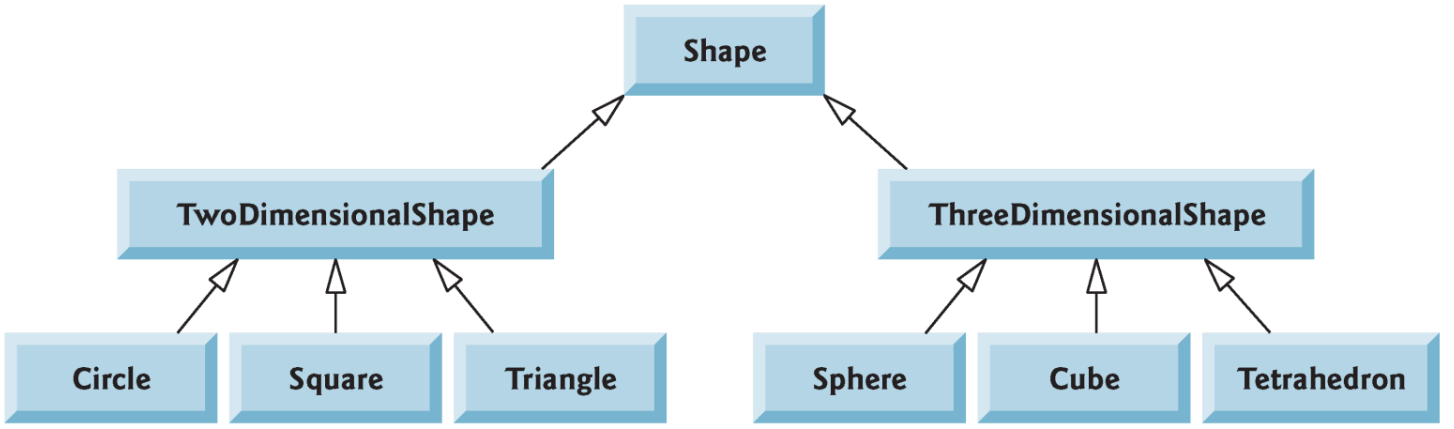

Fig. 12.3 | Inheritance hierarchy for Shapes.

Private, Public, Protected members

- We have learned
  - Private and public
- Protected
  - Between public and private access
  - Can be accessed by
    - Derived class
    - Friend of derived class

Syntax

- Syntax
  - class DerivedClassName : inheritanceType BaseClassName
- Inheritance types
  - Public
  - Private
  - Protected
- Example

  class TwoDimensionalShape : public Shape

  class Crectangle : public CPolygon

Concept: Derived Class

- Inheritance
  - Base class
  - Derived class

- Example
  - CRightTriangle is inherited from Cpolygon

```
class CRightTriangle: public CPolygon
```

Public inheritance is usually used

```
1  // derived classes
2  #include <iostream>
3  using namespace std;
4
5  class CPolygon {
6    protected:
7      int width, height;
8    public:
9      void set_values (int a, int b)
10       { width=a; height=b;}
11   };
12
13 class CRectangle: public CPolygon {
14   public:
15     int area ()
16       { return (width * height); }
17   };
```

```
19 class CRightTriangle: public CPolygon {
20   public:
21     int area ()
22       { return (width * height / 2); }
23   };
24
25 int main () {
26   CRectangle rect;
27   CRightTriangle trgl;
28   rect.set_values (4,5);
29   trgl.set_values (4,5);
30   cout << rect.area() << endl;
31   cout << trgl.area() << endl;
32 }
```

Constructor in Derived Class

- Specify the inheritance of a constructor in the derived class
  - Yes: Call the specified contractor

```
13 //Constructor: not specified: call default
14 class daughter : public mother {
15   public:
16     daughter (int a)
17       { cout << "daughter: int parameter\n\n"; }
18 };
19
20 //Construcotr: call mother (int a)
21 class son : public mother {
22   public:
23     son (int a) : mother (a)
24       { cout << "son: int parameter\n\n"; }
25 };
```

```
1  // constructors and derived classes
2  #include <iostream>
3  using namespace std;
4
5  class mother {
6    public:
7      mother ()
8        { cout << "mother: no parameters\n"; }
9      mother (int a)
10       { cout << "mother: int parameter\n"; }
11 };
```

```
26 int main () {
27   daughter Jane(3);
28   son John(5);
29 }
```

Concept: Multiple Inheritance

- Inheritance from more than 1 base class

- Example
  - CRectagle is inherited from
    - CPolygon
    - Coutput

```
class CRectangle: public CPolygon, public Coutput
{

}
```

```
1  // multiple inheritance
2  #include <iostream>
3  using namespace std;
4
5  class CPolygon {
6    protected:
7      int width, height;
8    public:
9      void set_values (int a, int b)
10       { width=a; height=b;}
11   };
12
```

Base Class 1

Base Class 2

```
13 class COutput {
14   public:
15     void output (int i);
16   };
17
18 void COutput::output (int i) {
19   cout << i << endl;
20   }
```

```
22 class CRectangle: public CPolygon, public COutput {
23   public:
24     int area ()
25       { return (width * height); }
26   };
27
28 class CRightTriangle: public CPolygon, public COutput {
29   public:
30     int area ()
31       { return (width * height / 2); }
32   };
33
```

```
34 int main () {
35   CRectangle rect;
36   CRightTriangle trgl;
37   rect.set_values (4,5);
38   trgl.set_values (4,5);
39   rect.output (rect.area());
40   trgl.output (trgl.area());
41 }
```

12.6 Public, Private, Protected Inheritance

- We usually use public 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 member functions, `friend` functions and nonmember functions.	`protected` in derived class. Can be accessed directly by member functions and `friend` functions.	`private` in derived class. Can be accessed directly by member functions and `friend` functions.
`protected`	`protected` in derived class. Can be accessed directly by member functions and `friend` functions.	`protected` in derived class. Can be accessed directly by member functions and `friend` functions.	`private` in derived class. Can be accessed directly by member functions and `friend` functions.
`private`	Hidden in derived class. Can be accessed by member functions and `friend` functions through `public` or `protected` member functions of the base class.	Hidden in derived class. Can be accessed by member functions and `friend` functions through `public` or `protected` member functions of the base class.	Hidden in derived class. Can be accessed by member functions and `friend` functions through `public` or `protected` member functions of the base class.

Fig. 12.27 | Summary of base-class member accessibility in a derived class.

Example (Textbook 12.4.4)

- (public) inheritance using protected data
  - Base Class --- CommisionEmployee
  - Derived Class --- BasePlusCommissionEmployee
  - Fig. 12.12~12.16

- Other examples in the textbook [**In the next lecture**]
  - Without inheritance (Fig 12.4~12.9)
    - Lots of duplicates
  - Errors in inheritance (Fig 12.10~12.11) [**Skipped**]
    - Private base-class data cannot be accessed by derived class → error
  - inheritance using private data (Fig 12.17~12.21)
    - Better information hiding/software engineering

```
1  // Fig. 12.12: CommissionEmployee.h
2  // CommissionEmployee class definition with protected data.
3  #ifndef COMMISSION_H
4  #define COMMISSION_H
5
6  #include <string> // C++ standard string class
7  using namespace std;
8
9  class CommissionEmployee
10 {
11 public:
12    CommissionEmployee( const string &, const string &, const string &,
13       double = 0.0, double = 0.0 );
14
15    void setFirstName( const string & ); // set first name
16    string getFirstName() const; // return first name
17
18    void setLastName( const string & ); // set last name
19    string getLastName() const; // return last name
20
```

Fig. 12.12 | `CommissionEmployee` class definition that declares `protected` data to allow access by derived classes. (Part 1 of 2.)

```
21      void setSocialSecurityNumber( const string & ); // set SSN
22      string getSocialSecurityNumber() const; // return SSN
23
24      void setGrossSales( double ); // set gross sales amount
25      double getGrossSales() const; // return gross sales amount
26
27      void setCommissionRate( double ); // set commission rate
28      double getCommissionRate() const; // return commission rate
29
30      double earnings() const; // calculate earnings
31      void print() const; // print CommissionEmployee object
32   protected:
33      string firstName;
34      string lastName;
35      string socialSecurityNumber;
36      double grossSales; // gross weekly sales
37      double commissionRate; // commission percentage
38   }; // end class CommissionEmployee
39
40   #endif
```

Fig. 12.12 | `CommissionEmployee` class definition that declares `protected` data to allow access by derived classes. (Part 2 of 2.)

```
1  // Fig. 12.13: CommissionEmployee.cpp
2  // Class CommissionEmployee member-function definitions.
3  #include <iostream>
4  #include "CommissionEmployee.h" // CommissionEmployee class definition
5  using namespace std;
6
7  // constructor
8  CommissionEmployee::CommissionEmployee(
9     const string &first, const string &last, const string &ssn,
10    double sales, double rate )
11 {
12    firstName = first; // should validate
13    lastName = last; // should validate
14    socialSecurityNumber = ssn; // should validate
15    setGrossSales( sales ); // validate and store gross sales
16    setCommissionRate( rate ); // validate and store commission rate
17 } // end CommissionEmployee constructor
18
19 // set first name
20 void CommissionEmployee::setFirstName( const string &first )
21 {
22    firstName = first; // should validate
23 } // end function setFirstName
24
```


Fig. 12.13 | CommissionEmployee class with protected data. (Part 1 of 4.)

```
25  // return first name
26  string CommissionEmployee::getFirstName() const
27  {
28     return firstName;
29  } // end function getFirstName
30
31  // set last name
32  void CommissionEmployee::setLastName( const string &last )
33  {
34     lastName = last; // should validate
35  } // end function setLastName
36
37  // return last name
38  string CommissionEmployee::getLastName() const
39  {
40     return lastName;
41  } // end function getLastName
42
43  // set social security number
44  void CommissionEmployee::setSocialSecurityNumber( const string &ssn )
45  {
46     socialSecurityNumber = ssn; // should validate
47  } // end function setSocialSecurityNumber
48
```

Fig. 12.13 | `CommissionEmployee` class with `protected` data. (Part 2 of 4.)

```
49   // return social security number
50   string CommissionEmployee::getSocialSecurityNumber() const
51   {
52      return socialSecurityNumber;
53   } // end function getSocialSecurityNumber
54
55   // set gross sales amount
56   void CommissionEmployee::setGrossSales( double sales )
57   {
58      grossSales = ( sales < 0.0 ) ? 0.0 : sales;
59   } // end function setGrossSales
60
61   // return gross sales amount
62   double CommissionEmployee::getGrossSales() const
63   {
64      return grossSales;
65   } // end function getGrossSales
66
67   // set commission rate
68   void CommissionEmployee::setCommissionRate( double rate )
69   {
70      commissionRate = ( rate > 0.0 && rate < 1.0 ) ? rate : 0.0;
71   } // end function setCommissionRate
72
```

Fig. 12.13 | CommissionEmployee class with protected data. (Part 3 of 4.)

```
73   // return commission rate
74   double CommissionEmployee::getCommissionRate() const
75   {
76      return commissionRate;
77   } // end function getCommissionRate
78
79   // calculate earnings
80   double CommissionEmployee::earnings() const
81   {
82      return commissionRate * grossSales;
83   } // end function earnings
84
85   // print CommissionEmployee object
86   void CommissionEmployee::print() const
87   {
88      cout << "commission employee: " << firstName << ' ' << lastName
89         << "\nsocial security number: " << socialSecurityNumber
90         << "\ngross sales: " << grossSales
91         << "\ncommission rate: " << commissionRate;
92   } // end function print
```

Fig. 12.13 | `CommissionEmployee` class with `protected` data. (Part 4 of 4.)

```
1  // Fig. 12.14: BasePlusCommissionEmployee.h
2  // BasePlusCommissionEmployee class derived from class
3  // CommissionEmployee.
4  #ifndef BASEPLUS_H
5  #define BASEPLUS_H
6
7  #include <string> // C++ standard string class
8  #include "CommissionEmployee.h" // CommissionEmployee class declaration
9  using namespace std;
10
11 class BasePlusCommissionEmployee : public CommissionEmployee
12 {
13 public:
14    BasePlusCommissionEmployee( const string &, const string &,
15       const string &, double = 0.0, double = 0.0, double = 0.0 );
16
17    void setBaseSalary( double ); // set base salary
18    double getBaseSalary() const; // return base salary
19
20    double earnings() const; // calculate earnings
21    void print() const; // print BasePlusCommissionEmployee object
22 private:
23    double baseSalary; // base salary
24 }; // end class BasePlusCommissionEmployee
25
26 #endif
```

```
1  // Fig. 12.15: BasePlusCommissionEmployee.cpp
2  // Class BasePlusCommissionEmployee member-function definitions.
3  #include <iostream>
4  #include "BasePlusCommissionEmployee.h"
5  using namespace std;
6
7  // constructor
8  BasePlusCommissionEmployee::BasePlusCommissionEmployee(
9     const string &first, const string &last, const string &ssn,
10    double sales, double rate, double salary )
11    // explicitly call base-class constructor
12    : CommissionEmployee( first, last, ssn, sales, rate )
13 {
14    setBaseSalary( salary ); // validate and store base salary
15 } // end BasePlusCommissionEmployee constructor
16
17 // set base salary
18 void BasePlusCommissionEmployee::setBaseSalary( double salary )
19 {
20    baseSalary = ( salary < 0.0 ) ? 0.0 : salary;
21 } // end function setBaseSalary
22
```

Fig. 12.15 | `BasePlusCommissionEmployee` implementation file for `BasePlusCommissionEmployee` class that inherits `protected` data from `CommissionEmployee`. (Part 1 of 2.)

```
23  // return base salary
24  double BasePlusCommissionEmployee::getBaseSalary() const
25  {
26     return baseSalary;
27  } // end function getBaseSalary
28
29  // calculate earnings
30  double BasePlusCommissionEmployee::earnings() const
31  {
32     // can access protected data of base class
33     return baseSalary + ( commissionRate * grossSales );
34  } // end function earnings
35
36  // print BasePlusCommissionEmployee object
37  void BasePlusCommissionEmployee::print() const
38  {
39     // can access protected data of base class
40     cout << "base-salaried commission employee: " << firstName << ' '
41        << lastName << "\nsocial security number: " << socialSecurityNumber
42        << "\ngross sales: " << grossSales
43        << "\ncommission rate: " << commissionRate
44        << "\nbase salary: " << baseSalary;
45  } // end function print
```


Fig. 12.15 | BasePlusCommissionEmployee implementation file for

```
1  // Fig. 12.16: fig12_16.cpp
2  // Testing class BasePlusCommissionEmployee.
3  #include <iostream>
4  #include <iomanip>
5  #include "BasePlusCommissionEmployee.h"
6  using namespace std;
7
8  int main()
9  {
10    // instantiate BasePlusCommissionEmployee object
11    BasePlusCommissionEmployee
12       employee( "Bob", "Lewis", "333-33-3333", 5000, .04, 300 );
13
14    // set floating-point output formatting
15    cout << fixed << setprecision( 2 );
16
17    // get commission employee data
18    cout << "Employee information obtained by get functions: \n"
19       << "\nFirst name is " << employee.getFirstName()
20       << "\nLast name is " << employee.getLastName()
21       << "\nSocial security number is "
22       << employee.getSocialSecurityNumber()
```

Fig. 12.16 | protected base-class data can be accessed from derived class. (Part 1 of 3.)

```
23          << "\nGross sales is " << employee.getGrossSales()
24          << "\nCommission rate is " << employee.getCommissionRate()
25          << "\nBase salary is " << employee.getBaseSalary() << endl;
26
27       employee.setBaseSalary( 1000 ); // set base salary
28
29       cout << "\nUpdated employee information output by print function: \n"
30          << endl;
31       employee.print(); // display the new employee information
32
33       // display the employee's earnings
34       cout << "\n\nEmployee's earnings: $" << employee.earnings() << endl;
35    } // end main
```

Fig. 12.16 | `protected` base-class data can be accessed from derived class. (Part 2 of 3.)

```
Employee information obtained by get functions:

First name is Bob
Last name is Lewis
Social security number is 333-33-3333
Gross sales is 5000.00
Commission rate is 0.04
Base salary is 300.00

Updated employee information output by print function:

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
commission rate: 0.04
base salary: 1000.00

Employee's earnings: $1200.00
```

Fig. 12.16 | `protected` base-class data can be accessed from derived class. (Part 3 of 3.)

Section 12.4 examples

- 12.4.1 Commission employee class (Fig 12.4~12.6)
- 12.4.2 Base+Commission employee class (Fig 12.7~12.9)
- 12.4.3 [Error] inheritance (Fig 12.10~12.11)
  - Derived class **cannot** access base class's private data
- 12.4.4 protected data in inheritance(Fig 12.12~12.16)
  - Base class's protected data can be accessed by
    - Member function of derived class
    - Friend of derived class
- 12.4.5 rewrite the example with Set, Get functions (Fig 12.17~12.21)
  - a better example with software engineering techniques

12.5 Constructor and Destructor in Inheritance

- When an object of derived is created
  - (1) Constructor of base class
  - (2) Constructor of derived class

- When an object of derived is destroyed
  - Reverse order of the constructor
    - (1) destructor of derived class
    - (2) destructor of base class