WASHINGTON STATE UNIVERSITY
World Class. Face to Face.

Cpt S 122 – Data Structures

Inheritance

Nirmalya Roy

School of Electrical Engineering and Computer Science
Washington State University

Topics

- Introduction
- Base Classes & Derived Classes
- Relationship between Base Classes & Derived Classes
  - Without and With Inheritance
  - Inheritance using `Protected` & `Private` data
- `public, protected,` and `private` Inheritance
- Software Engineering Practice with Inheritance

Introduction

- Inheritance is a form of software reuse
  - *you create a class that absorbs an existing class's data and behaviors and enhances them with new capabilities*.
- The new class should inherit the members of an existing class.
- This existing class is called the base class
  - the new class is referred to as the derived class.
- A derived class represents a more specialized group of objects.
- A derived class contains behaviors inherited from its base class and can contain additional behaviors.
- A derived class can also customize behaviors inherited from the base class.

Introduction (cont.)

- A direct base class
  - the base class from which a derived class explicitly inherits.
- An indirect base class
  - inherited from two or more levels up in the class hierarchy.
- In the case of single inheritance
  - a class is derived from one base class.
- C++ also supports multiple inheritance
  - a derived class inherits from multiple (possibly unrelated) base classes.

Introduction (cont.)

- C++ offers `public`, `protected` and `private` inheritance.
- With `public` inheritance, every object of a derived class is also an object of that derived class's base class.
- However, base-class objects are not objects of their derived classes.
- `private` inheritance can be used as an alternative to composition.
- `protected` inheritance, is rarely used.

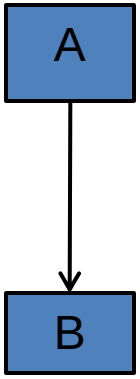
Introduction (cont.)

- With object-oriented programming, we focus on the commonalities among objects in the system rather than on the special cases.
- We distinguish between the is-a relationship and the *has-a* relationship.
- The *is-a* relationship represents inheritance.
  - In an *is-a* relationship, an object of a derived class also can be treated as an object of its base class.
- By contrast, the *has-a* relationship represents composition.

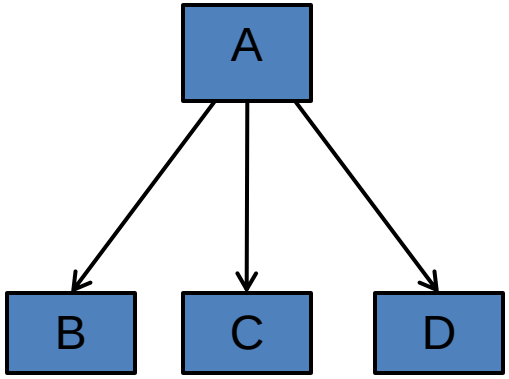
Introduction (cont.)

- Derived-class member functions might require access to
  - base-class data members and member functions.
- A derived class can access the non-`private` members of its base class.
- Base-class members that should not be accessible to the member functions of derived classes
  - should be declared `private` in the base class.
- A derived class can change the values of `private` base-class members,
  - only through non-`private` member functions provided in the base class and inherited into the derived class.

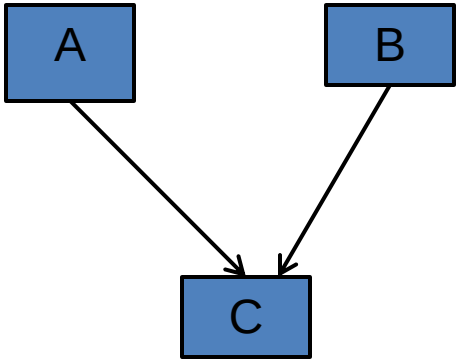
Variety of Inheritance



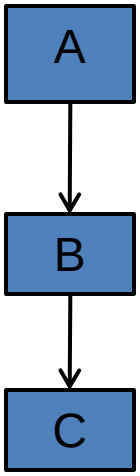
Single Inheritance



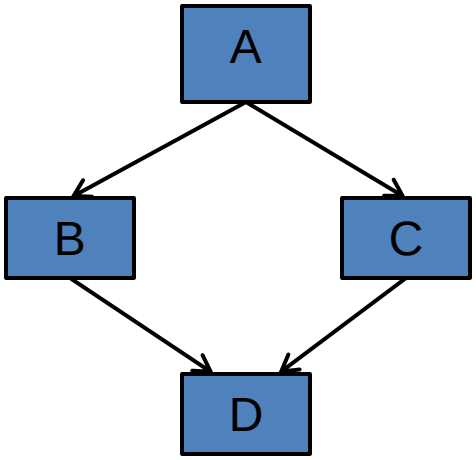
Hierarchical Inheritance



Multiple Inheritance



Multilevel Inheritance



Hybrid Inheritance

Defining Derived Classes

class *derived-class name* : *visibility-mode* **base-class name**

{

……..//

…….// members of derived class

……..//

};

- Visibility mode specifies whether the features of the base class are *privately* derived or *publicly* derived
- The default visibility mode is *private*

Examples:

```
Class ABC : private XYZ        //private derivation
{  …members of ABC
};

Class ABC : public XYZ         //public derivation
{  …members of ABC
};

Class ABC :  XYZ          //private derivation by default
{…members of ABC
};
```

Inheritance Accessibility

- A base class is ***privately*** inherited
  - public members of the base class become private members of the derived class.
  - public members of the base class can only be accessed by the member functions of the derived class.
  - They are inaccessible to the objects of the derived class.
- A base class is ***publicly*** inherited
  - public members of the base class become public members of the derived class.
  - They are accessible to the objects of the derived class.
  - In both the cases, the private members are not inherited
    - The private members of a base class will never become the members of its derived class.

Base Classes and Derived Classes

- Often, an object of one class *is an* object of another class.
  - For example, in geometry, a rectangle *is a* quadrilateral (as are squares, parallelograms and trapezoids).
  - Thus, in C++, class `Rectangle` can be said to inherit from class `Quadrilateral`.
  - In this context, class `Quadrilateral` is a base class, and class `Rectangle` is a derived class.
  - A rectangle *is a* specific type of quadrilateral, but it's incorrect to claim that a quadrilateral is a rectangle
    - the quadrilateral could be a parallelogram or some other shape.

Example: Base Classes & Derived Classes

Base class	Derived classes
Student	GraduateStudent, UndergraduateStudent
Shape	Circle, Triangle, Rectangle, Sphere, Cube
Loan	CarLoan, HomeImprovementLoan, MortgageLoan
Employee	Faculty, Staff
Account	CheckingAccount, SavingsAccount

Fig. 12.1 | Inheritance examples.

Base Classes and Derived Classes (cont.)

- A simple inheritance hierarchy with five levels is shown in the next Fig.
- A university community has thousands of members.
- Employees are either faculty members or staff members.
- Faculty members are either administrators (such as deans and department chairpersons) or teachers.
- Some administrators, however, also teach classes.
- Note that we've used multiple inheritance to form class `AdministratorTeacher`.
- Also, this inheritance hierarchy could contain many other classes.

Multiple Inheritance: Base Classes and Derived Classes

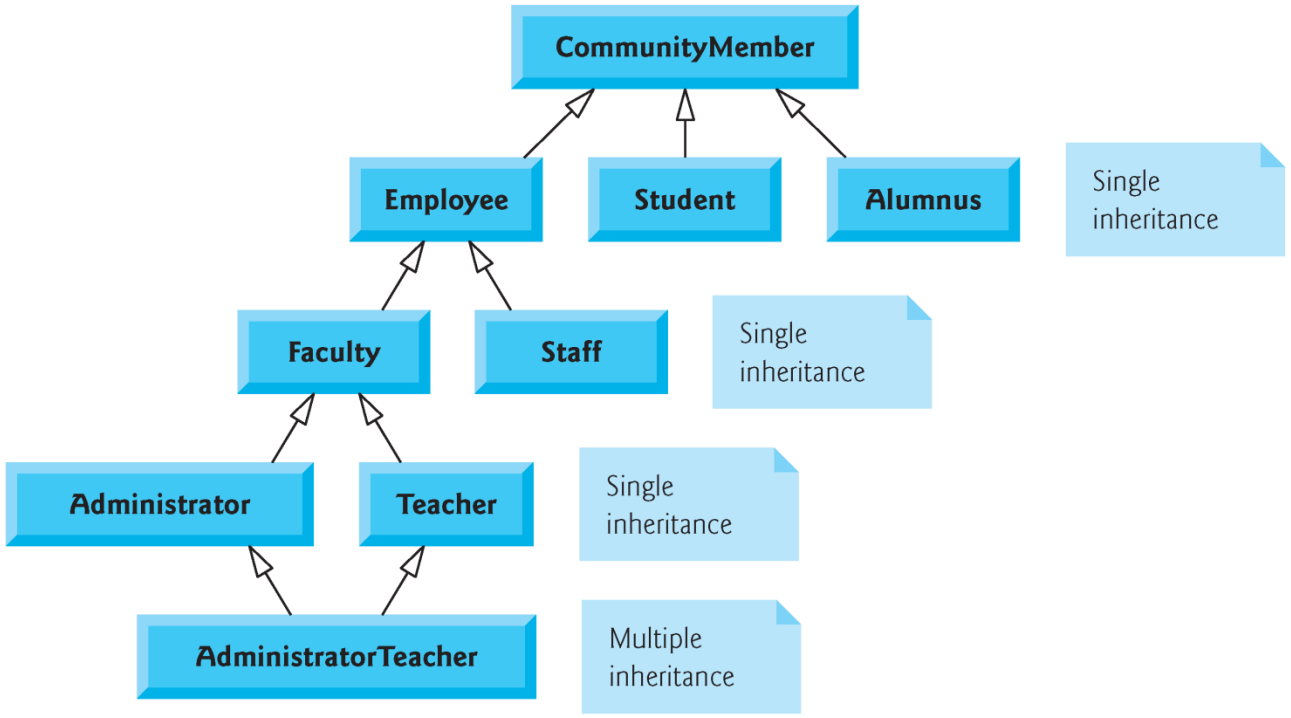


Fig. 12.2 | Inheritance hierarchy for university `CommunityMembers`.

Base Classes and Derived Classes (cont.)

- Each arrow in the hierarchy represents an *is-a relationship*.
  - As we follow the arrows in this class hierarchy, we can state
  - "an `Employee` *is a* `CommunityMember`" and
  - "a `Teacher` *is a* `Faculty` member."
  - `CommunityMember` is the direct base class of `Employee`, `Student` and `Alumnus`.
  - `CommunityMember` is an indirect base class of all the other classes in the diagram.
- Starting from the bottom of the diagram, you can follow the arrows and apply the *is-a* relationship to the topmost base class.
  - An `AdministratorTeacher` *is an* `Administrator`, *is a* `Faculty` member, *is an* `Employee` and *is a* `CommunityMember`.

Example: Base Classes & Derived Classes

- The `Shape` inheritance hierarchy in next Figure.
- Begins with base class `Shape`.
- Classes `TwoDimensionalShape` and `ThreeDimensionalShape` derive from base class `Shape`
  - `Shape`s are either `TwoDimensionalShape`s or `Three-DimensionalShape`s.
- The third level of this hierarchy contains some more specific types of `TwoDimensionalShape`s and `ThreeDimensionalShape`s.
- Follow the arrows from the bottom of the diagram to the topmost base class in this class hierarchy to identify several *is-a* relationships.

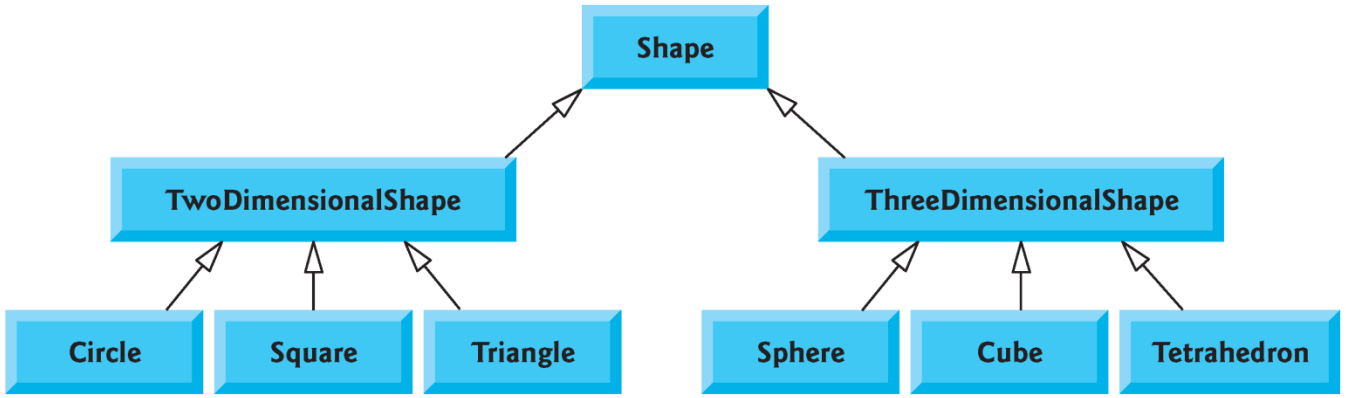


Fig. 12.3 | Inheritance hierarchy for Shapes.

`protected` Members

- Introduced access specifiers `public` and `private`.
- A base class's `public` members are accessible within its body and anywhere that the program has a handle (i.e., a name, reference or pointer) to an object of that class or one of its derived classes.
- A base class's `private` members are accessible only within its body and to the `friend`s of that base class.
- Now we introduce the access specifier `protected`.
- Using `protected` access offers an intermediate level of protection between `public` and `private` access.
- A base class's `protected` members can be accessed within the body of that base class, by members and `friend`s of that base class, and by members and `friend`s of any classes derived from that base class.
- Derived-class member functions can refer to `public` and `protected` members of the base class simply by using the member names.

`protected` Members

- When a derived-class member function redefines a base-class member function, the base-class member can be accessed from the derived class
  - by preceding the base-class member name with the base-class name and the binary scope resolution operator (`::`).

Relationship between Base Classes and Derived Classes

- We use an inheritance hierarchy containing types of employees in a company's payroll application
  - discuss the relationship between a base class and a derived class.
- Commission employees (who will be represented as *objects of a base class*) are paid a percentage of their sales,
- Base-salaried commission employees (who will be represented as *objects of a derived class*) receive a base salary plus a percentage of their sales.

Example: Base Class & Derived Class

- `CommissionEmployee`'s class definition.
- `CommissionEmployee`'s `public` services include a constructor and member functions `earnings` and `print`.
- Also includes `public` *get* and *set* functions that manipulate the class's data members `firstName`, `lastName`, `socialSecurityNumber`, `grossSales` and `commissionRate`.
  - These data members are `private`, so objects of other classes cannot directly access this data.
  - Declaring data members as `private` and providing non-`private` *get* and *set* functions to manipulate and validate the data members helps enforce good software engineering.

```
1  // Fig. 12.4: CommissionEmployee.h
2  // CommissionEmployee class definition represents a commission employee.
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
21    void setSocialSecurityNumber( const string & ); // set SSN
22    string getSocialSecurityNumber() const; // return SSN
23
```

Fig. 12.4 | CommissionEmployee class header. (Part 1 of 2.)

```
24      void setGrossSales( double ); // set gross sales amount
25      double getGrossSales() const; // return gross sales amount
26
27      void setCommissionRate( double ); // set commission rate (percentage)
28      double getCommissionRate() const; // return commission rate
29
30      double earnings() const; // calculate earnings
31      void print() const; // print CommissionEmployee object
32  private:
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. 12.4 | CommissionEmployee class header. (Part 2 of 2.)

```
1  // Fig. 12.5: 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
```

Fig. 12.5 | Implementation file for `CommissionEmployee` class that represents an employee who is paid a percentage of gross sales. (Part 1 of 6.)

```
19  // set first name
20  void CommissionEmployee::setFirstName( const string &first )
21  {
22     firstName = first; // should validate
23  } // end function setFirstName
24
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
```

Fig. 12.5 | Implementation file for `CommissionEmployee` class that represents an employee who is paid a percentage of gross sales. (Part 2 of 6.)

```
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
49  // return social security number
50  string CommissionEmployee::getSocialSecurityNumber() const
51  {
52     return socialSecurityNumber;
53  } // end function getSocialSecurityNumber
54
```

Fig. 12.5 | Implementation file for `CommissionEmployee` class that represents an employee who is paid a percentage of gross sales. (Part 3 of 6.)

```
55  // set gross sales amount
56  void CommissionEmployee::setGrossSales( double sales )
57  {
58     if ( sales >= 0.0 )
59        grossSales = sales;
60     else
61        throw invalid_argument( "Gross sales must be >= 0.0" );
62  } // end function setGrossSales
63
64  // return gross sales amount
65  double CommissionEmployee::getGrossSales() const
66  {
67     return grossSales;
68  } // end function getGrossSales
69
```

Fig. 12.5 | Implementation file for `CommissionEmployee` class that represents an employee who is paid a percentage of gross sales. (Part 4 of 6.)

```
70  // set commission rate
71  void CommissionEmployee::setCommissionRate( double rate )
72  {
73     if ( rate > 0.0 && rate < 1.0 )
74        commissionRate = rate;
75     else
76        throw invalid_argument( "Commission rate must be > 0.0 and < 1.0" );
77  } // end function setCommissionRate
78
79  // return commission rate
80  double CommissionEmployee::getCommissionRate() const
81  {
82     return commissionRate;
83  } // end function getCommissionRate
84
85  // calculate earnings
86  double CommissionEmployee::earnings() const
87  {
88     return commissionRate * grossSales;
89  } // end function earnings
90
```

Fig. 12.5 | Implementation file for `CommissionEmployee` class that represents an employee who is paid a percentage of gross sales. (Part 5 of 6.)

```
91  // print CommissionEmployee object
92  void CommissionEmployee::print() const
93  {
94     cout << "commission employee: " << firstName << ' ' << lastName
95        << "\nsocial security number: " << socialSecurityNumber
96        << "\ngross sales: " << grossSales
97        << "\ncommission rate: " << commissionRate;
98  } // end function print
```

Fig. 12.5 | Implementation file for `CommissionEmployee` class that represents an employee who is paid a percentage of gross sales. (Part 6 of 6.)

CommissionEmployee Class

- The `CommissionEmployee` constructor definition purposely does not use member-initializer syntax in the first several examples of this section, so that we can demonstrate how `private` and `protected` specifiers affect member access in derived classes.
  - Later in this section, we'll return to using member-initializer lists in the constructors.
- Member function `earnings` calculates a `CommissionEmployee`'s earnings.
- Member function `print` displays the values of a `CommissionEmployee` object's data members

```
1  // Fig. 12.6: fig12_06.cpp
2  // Testing class CommissionEmployee.
3  #include <iostream>
4  #include <iomanip>
5  #include "CommissionEmployee.h" // CommissionEmployee class definition
6  using namespace std;
7
8  int main()
9  {
10    // instantiate a CommissionEmployee object
11    CommissionEmployee employee(
12       "Sue", "Jones", "222-22-2222", 10000, .06 );
13
14    // set floating-point output formatting
15    cout << fixed << setprecision( 2 );
16
```

Fig. 12.6 | CommissionEmployee class test program. (Part 1 of 3.)

```
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()
23         << "\nGross sales is " << employee.getGrossSales()
24         << "\nCommission rate is " << employee.getCommissionRate() << endl;
25
26      employee.setGrossSales( 8000 ); // set gross sales
27      employee.setCommissionRate( .1 ); // set commission rate
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. 12.6 | CommissionEmployee class test program. (Part 2 of 3.)

```
Employee information obtained by get functions:

First name is Sue
Last name is Jones
Social security number is 222-22-2222
Gross sales is 10000.00
Commission rate is 0.06

Updated employee information output by print function:

commission employee: Sue Jones
social security number: 222-22-2222
gross sales: 8000.00
commission rate: 0.10

Employee's earnings: $800.00
```

Fig. 12.6 | CommissionEmployee class test program. (Part 3 of 3.)

Creating a Class Without Using Inheritance

- We now discuss the second part of our introduction to inheritance by creating and testing
  - a completely new and independent class `BasePlusCommissionEmployee`,
    - contains a first name, last name, social security number, gross sales amount, commission rate and base salary.

BasePlusCommissionEmployee Class

```
1  // Fig. 12.7: BasePlusCommissionEmployee.h
2  // BasePlusCommissionEmployee class definition represents an employee
3  // that receives a base salary in addition to commission.
4  #ifndef BASEPLUS_H
5  #define BASEPLUS_H
6
7  #include <string> // C++ standard string class
8  using namespace std;
9
```

Fig. 12.7 | BasePlusCommissionEmployee class header. (Part 1 of 3.)

```
10  class BasePlusCommissionEmployee
11  {
12  public:
13     BasePlusCommissionEmployee( const string &, const string &,
14        const string &, double = 0.0, double = 0.0, double = 0.0 );
15
16     void setFirstName( const string & ); // set first name
17     string getFirstName() const; // return first name
18
19     void setLastName( const string & ); // set last name
20     string getLastName() const; // return last name
21
22     void setSocialSecurityNumber( const string & ); // set SSN
23     string getSocialSecurityNumber() const; // return SSN
24
25     void setGrossSales( double ); // set gross sales amount
26     double getGrossSales() const; // return gross sales amount
27
28     void setCommissionRate( double ); // set commission rate
29     double getCommissionRate() const; // return commission rate
30
```

Fig. 12.7 | BasePlusCommissionEmployee class header. (Part 2 of 3.)

```
31      void setBaseSalary( double ); // set base salary
32      double getBaseSalary() const; // return base salary
33
34      double earnings() const; // calculate earnings
35      void print() const; // print BasePlusCommissionEmployee object
36   private:
37      string firstName;
38      string lastName;
39      string socialSecurityNumber;
40      double grossSales; // gross weekly sales
41      double commissionRate; // commission percentage
42      double baseSalary; // base salary
43   }; // end class BasePlusCommissionEmployee
44
45   #endif
```

Fig. 12.7 | BasePlusCommissionEmployee class header. (Part 3 of 3.)

```cpp
// Fig. 12.8: BasePlusCommissionEmployee.cpp
// Class BasePlusCommissionEmployee member-function definitions.
#include <iostream>
#include "BasePlusCommissionEmployee.h"
using namespace std;

// constructor
BasePlusCommissionEmployee::BasePlusCommissionEmployee(
   const string &first, const string &last, const string &ssn,
   double sales, double rate, double salary )
{
   firstName = first; // should validate
   lastName = last; // should validate
   socialSecurityNumber = ssn; // should validate
   setGrossSales( sales ); // validate and store gross sales
   setCommissionRate( rate ); // validate and store commission rate
   setBaseSalary( salary ); // validate and store base salary
} // end BasePlusCommissionEmployee constructor

```

Fig. 12.8 | BasePlusCommissionEmployee class represents an employee who receives a base salary in addition to a commission. (Part 1 of 7.)

```
20  // set first name
21  void BasePlusCommissionEmployee::setFirstName( const string &first )
22  {
23     firstName = first; // should validate
24  } // end function setFirstName
25
26  // return first name
27  string BasePlusCommissionEmployee::getFirstName() const
28  {
29     return firstName;
30  } // end function getFirstName
31
32  // set last name
33  void BasePlusCommissionEmployee::setLastName( const string &last )
34  {
35     lastName = last; // should validate
36  } // end function setLastName
37
```

Fig. 12.8 | BasePlusCommissionEmployee class represents an employee who receives a base salary in addition to a commission. (Part 2 of 7.)

```
38  // return last name
39  string BasePlusCommissionEmployee::getLastName() const
40  {
41     return lastName;
42  } // end function getLastName
43
44  // set social security number
45  void BasePlusCommissionEmployee::setSocialSecurityNumber(
46     const string &ssn )
47  {
48     socialSecurityNumber = ssn; // should validate
49  } // end function setSocialSecurityNumber
50
51  // return social security number
52  string BasePlusCommissionEmployee::getSocialSecurityNumber() const
53  {
54     return socialSecurityNumber;
55  } // end function getSocialSecurityNumber
56
```

Fig. 12.8 | BasePlusCommissionEmployee class represents an employee who receives a base salary in addition to a commission. (Part 3 of 7.)

```
57  // set gross sales amount
58  void BasePlusCommissionEmployee::setGrossSales( double sales )
59  {
60     if ( sales >= 0.0 )
61        grossSales = sales;
62     else
63        throw invalid_argument( "Gross sales must be >= 0.0" );
64  } // end function setGrossSales
65
66  // return gross sales amount
67  double BasePlusCommissionEmployee::getGrossSales() const
68  {
69     return grossSales;
70  } // end function getGrossSales
71
```

Fig. 12.8 | `BasePlusCommissionEmployee` class represents an employee who receives a base salary in addition to a commission. (Part 4 of 7.)

```
72  // set commission rate
73  void BasePlusCommissionEmployee::setCommissionRate( double rate )
74  {
75     if ( rate > 0.0 && rate < 1.0 )
76        commissionRate = rate;
77     else
78        throw invalid_argument( "Commission rate must be > 0.0 and < 1.0" );
79  } // end function setCommissionRate
80
81  // return commission rate
82  double BasePlusCommissionEmployee::getCommissionRate() const
83  {
84     return commissionRate;
85  } // end function getCommissionRate
86
```

Fig. 12.8 | `BasePlusCommissionEmployee` class represents an employee who receives a base salary in addition to a commission. (Part 5 of 7.)

```
87  // set base salary
88  void BasePlusCommissionEmployee::setBaseSalary( double salary )
89  {
90     if ( salary >= 0.0 )
91        baseSalary = salary;
92     else
93        throw invalid_argument( "Salary must be >= 0.0" );
94  } // end function setBaseSalary
95
96  // return base salary
97  double BasePlusCommissionEmployee::getBaseSalary() const
98  {
99     return baseSalary;
100 } // end function getBaseSalary
101
102 // calculate earnings
103 double BasePlusCommissionEmployee::earnings() const
104 {
105    return baseSalary + ( commissionRate * grossSales );
106 } // end function earnings
107
```

Fig. 12.8 | `BasePlusCommissionEmployee` class represents an employee who receives a base salary in addition to a commission. (Part 6 of 7.)

```
108 // print BasePlusCommissionEmployee object
109 void BasePlusCommissionEmployee::print() const
110 {
111    cout << "base-salaried commission employee: " << firstName << ' '
112       << lastName << "\nsocial security number: " << socialSecurityNumber
113       << "\ngross sales: " << grossSales
114       << "\ncommission rate: " << commissionRate
115       << "\nbase salary: " << baseSalary;
116 } // end function print
```

Fig. 12.8 | `BasePlusCommissionEmployee` class represents an employee who receives a base salary in addition to a commission. (Part 7 of 7.)

A Class Without Using Inheritance

- The `BasePlusCommissionEmployee` header specifies class `BasePlusCommissionEmployee`'s `public` services
  - include the `BasePlusCommissionEmployee` constructor and member functions `earnings` and `print`.
  - `public` *get* and *set* functions for the class's `private` data members `firstName`, `lastName`, `social-SecurityNumber`, `grossSales`, `commissionRate` and `baseSalary`.
- Note the similarity between this class and class `Commission-Employee`
  - we won't yet exploit that similarity.
- Class `BasePlusCommissionEmployee`'s `earnings` member function computes the earnings of a base-salaried commission employee.

```
1  // Fig. 12.9: fig12_09.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
```

Fig. 12.9 | BasePlusCommissionEmployee class test program. (Part 1 of 3.)

```
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()
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. 12.9 | BasePlusCommissionEmployee class test program. (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. 12.9 | `BasePlusCommissionEmployee` class test program. (Part 3 of 3.)

A Class Without Using Inheritance

- Most of the code for class `BasePlusCommissionEmployee` is similar, if not identical, to the code for class `CommissionEmployee`.
- In class `BasePlusCommissionEmployee`,
  - `private` data members `firstName` and `lastName` and member functions `setFirstName`, `getFirstName`, `setLastName` and `getLastName` are identical to those of class `CommissionEmployee`.
- Both classes contain `private` data members
  - `socialSecurityNumber`, `commissionRate` and `grossSales`, as well as *get* and *set* functions to manipulate these members.

A Class Without Using Inheritance

- The `BasePlusCommissionEmployee` constructor is almost identical to that of class `CommissionEmployee`,
  - except that `BasePlusCommissionEmployee`'s constructor also sets the `baseSalary`.
- The other additions to class `BasePlusCommissionEmployee` are `private` data member
  - `baseSalary` and member functions `setBaseSalary` and `getBase-Salary`.
- Class `BasePlusCommissionEmployee`'s `print` member function is nearly identical to that of class `CommissionEmployee`,
  - except that `BasePlusCommissionEmployee`'s `print` also outputs the value of data member `baseSalary`.

A Class Without Using Inheritance

- We literally copied code from class `CommissionEmployee` and pasted it into class `BasePlusCommissionEmployee`,
  - modified class `BasePlusCommissionEmployee` to include a base salary and member functions.
- This "copy-and-paste" approach is error prone and time consuming.
- Worse yet, it can spread many physical copies of the same code throughout a system
  - creating a code-maintenance nightmare.
- If changes are required, make the changes only in the base class
  - Derived classes then inherit the changes.

A Class Using Inheritance Hierarchy

- Now we create and test a new `BasePlusCommissionEmployee` class that derives from class `CommissionEmployee`
- In this example, a `BasePlusCommissionEmployee` object *is a* `CommissionEmployee` (because inheritance passes on the capabilities of class `CommissionEmployee`), but class `BasePlusCommissionEmployee` also has data member `baseSalary`.
- The colon (`:`) of the class definition indicates inheritance.
- Keyword `public` indicates the type of inheritance.
- As a derived class (formed with `public` inheritance), `BasePlusCommissionEmployee` inherits all the members of class `CommissionEmployee`, except for the constructor
  - each class provides its own constructors that are specific to the class.

A Class Using Inheritance Hierarchy

- Destructors, too, are not inherited
- Thus, the `public` services of `BasePlusCommissionEmployee` include
  - its constructor
  - the `public` member functions inherited from class `CommissionEmployee`
  - we cannot see these inherited member functions in `BasePlusCommissionEmployee`'s source code
  - they're nevertheless a part of derived class `BasePlusCommissionEmployee`.
- The derived class's `public` services also include member functions `setBaseSalary`, `getBaseSalary`, `earnings` and `print`.

```
1  // Fig. 12.10: 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
```

Fig. 12.10 | BasePlusCommissionEmployee class definition indicating inheritance relationship with class CommissionEmployee. (Part 1 of 2.)

Header file Inclusion

- Notice that we `#include` the base class's header in the derived class's header.
- This is necessary for three reasons.
  - The derived class uses the base class's name, so we must tell the compiler that the base class exists.
  - The compiler uses a class definition to determine the size of an object of that class. A client program that creates an object of a class must `#include` the class definition to enable the compiler to reserve the proper amount of memory for the object.
  - The compiler must determine whether the derived class uses the base class's inherited members properly.

Declaration: Derived Class

```
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
```

Fig. 12.10 | `BasePlusCommissionEmployee` class definition indicating inheritance relationship with class `CommissionEmployee`. (Part 2 of 2.)

```
1  // Fig. 12.11: 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
```

Fig. 12.11 | BasePlusCommissionEmployee implementation file: private base-class data cannot be accessed from derived class. (Part 1 of 5.)

```
17  // set base salary
18  void BasePlusCommissionEmployee::setBaseSalary( double salary )
19  {
20     if ( salary >= 0.0 )
21        baseSalary = salary;
22     else
23        throw invalid_argument( "Salary must be >= 0.0" );
24  } // end function setBaseSalary
25
26  // return base salary
27  double BasePlusCommissionEmployee::getBaseSalary() const
28  {
29     return baseSalary;
30  } // end function getBaseSalary
31
```

Fig. 12.11 | `BasePlusCommissionEmployee` implementation file: `private` base-class data cannot be accessed from derived class. (Part 2 of 5.)

Derived class cannot access Base class private data

```
32  // calculate earnings
33  double BasePlusCommissionEmployee::earnings() const
34  {
35     // derived class cannot access the base class's private data
36     return baseSalary + ( commissionRate * grossSales );
37  } // end function earnings
38
39  // print BasePlusCommissionEmployee object
40  void BasePlusCommissionEmployee::print() const
41  {
42     // derived class cannot access the base class's private data
43     cout << "base-salaried commission employee: " << firstName << ' '
44        << lastName << "\nsocial security number: " << socialSecurityNumber
45        << "\ngross sales: " << grossSales
46        << "\ncommission rate: " << commissionRate
47        << "\nbase salary: " << baseSalary;
48  } // end function print
```

Fig. 12.11 | BasePlusCommissionEmployee implementat
`private` base-class data cannot be accessed from derived c
3 of 5.)

```
24     void setGrossSales( double ); // set gross sales amount
25     double getGrossSales() const; // return gross sales amount
26
27     void setCommissionRate( double ); // set commission rate (percentag
28     double getCommissionRate() const; // return commission rate
29
30     double earnings() const; // calculate earnings
31     void print() const; // print CommissionEmployee object
32  private:
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. 12.4 | CommissionEmployee class header. (Part 2 of 2.)

```
C:\chhttp8_examples\ch12\Fig12_10_11\BasePlusCommissionEmployee.cpp(36) :
   error C2248: 'CommissionEmployee::commissionRate' :
   cannot access private member declared in class 'CommissionEmployee'

C:\chhttp8_examples\ch12\Fig12_10_11\BasePlusCommissionEmployee.cpp(36) :
   error C2248: 'CommissionEmployee::grossSales' :
   cannot access private member declared in class 'CommissionEmployee'

C:\chhttp8_examples\ch12\Fig12_10_11\BasePlusCommissionEmployee.cpp(43) :
   error C2248: 'CommissionEmployee::firstName' :
   cannot access private member declared in class 'CommissionEmployee'

C:\chhttp8_examples\ch12\Fig12_10_11\BasePlusCommissionEmployee.cpp(44) :
   error C2248: 'CommissionEmployee::lastName' :
   cannot access private member declared in class 'CommissionEmployee'
```

Fig. 12.11 | `BasePlusCommissionEmployee` implementation file: `private` base-class data cannot be accessed from derived class. (Part 4 of 5.)

```
C:\chhttp8_examples\ch12\Fig12_10_11\BasePlusCommissionEmployee.cpp(44) :
   error C2248: 'CommissionEmployee::socialSecurityNumber' :
   cannot access private member declared in class 'CommissionEmployee'

C:\chhttp8_examples\ch12\Fig12_10_11\BasePlusCommissionEmployee.cpp(45) :
   error C2248: 'CommissionEmployee::grossSales' :
   cannot access private member declared in class 'CommissionEmployee'

C:\chhttp8_examples\ch12\Fig12_10_11\BasePlusCommissionEmployee.cpp(46) :
   error C2248: 'CommissionEmployee::commissionRate' :
   cannot access private member declared in class 'CommissionEmployee'
```

Fig. 12.11 | `BasePlusCommissionEmployee` implementation file: `private` base-class data cannot be accessed from derived class. (Part 5 of 5.)

Creating a Class Using Inheritance Hierarchy

- Shows `BasePlusCommissionEmployee`'s member-function implementations.
- The constructor introduces base-class initializer syntax, which uses a *member initializer* to pass arguments to the base-class constructor.
- C++ requires that a *derived-class constructor call its base-class constructor* to initialize the base-class data members that are inherited into the derived class.
- If `BasePlusCommissionEmployee`'s constructor did not invoke class `CommissionEmployee`'s constructor explicitly, C++ would attempt to invoke class `CommissionEmployee`'s default constructor
  - *but the class does not have such a constructor, so the compiler would issue an error.*

Accessing private Data in Base Class

- The compiler generates errors because base class `CommissionEmployee`'s data members `commissionRate` and `grossSales` are `private`
  - derived class `BasePlusCommissionEmployee`'s member functions are not allowed to access base class `CommissionEmployee`'s `private` data.
- The compiler issues additional errors of `BasePlusCommissionEmployee`'s `print` member function for the same reason.
- C++ rigidly enforces restrictions on accessing `private` data members,
  - *even a derived class (which is intimately related to its base class) cannot access the base class's `private` data.*
- Example emphasizes that a derived class's member functions cannot access its base class's `private` data.

Accessing private data in base-class using base-class member function

- The errors in `BasePlusCommissionEmployee` could have been prevented by using
  - the *get* member functions inherited from class `CommissionEmployee`.
- For example, we could have invoked `getCommissionRate` and `getGrossSales` to access
  - `CommissionEmployee`'s `private` data members `commissionRate` and `grossSales`, respectively.

Inheritance Hierarchy Using protected Data

- To enable class `BasePlusCommissionEmployee` to directly access `CommissionEmployee` data members `firstName`, `lastName`, `socialSecurityNumber`, `grossSales` and `commissionRate`, we can declare those members as `protected` in the base class.
- A base class's `protected` members can be accessed
  - by members and `friend`s of the *base class* and
  - by members and `friend`s of *any classes derived from that base class.*

Inheritance Hierarchy Using protected Data

- Class `CommissionEmployee` now declares data members `firstName`, `lastName`, `socialSecurityNumber`, `grossSales` and `commissionRate` as `protected` rather than `private`.
- The member-function implementations are identical.

```
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.)

Using `protected` Data

- `BasePlusCommissionEmployee` inherits from class `CommissionEmployee`.
- Objects of class `BasePlusCommissionEmployee` can access inherited data members that are declared `protected` in class `CommissionEmployee`
  - data members `firstName`, `lastName`, `socialSecurityNumber`, `grossSales` and `commissionRate`
- As a result, the compiler does not generate errors when compiling the `BasePlusCommissionEmployee` `earnings` and `print` member function definitions.
- Objects of a derived class also can access `protected` members in any of that derived class's indirect base classes.

Using `protected` Data

- Inheriting `protected` data members slightly increases performance,
  - we can directly access the members without incurring the overhead of calls to *set* or *get* member functions.
- In most cases, it's better to use `private` data members to encourage proper software engineering,
  - leave code optimization issues to the compiler.

Using `protected` Data

- Using `protected` data members creates two serious problems.
  - The derived-class object does not have to use a member function to set the value of the base class's `protected` data member.
  - Derived-class member functions are more likely to be written so that they depend on the base-class implementation.
  - Derived classes should depend only on the base-class services (i.e., non-`private` member functions) and not on the base-class implementation.
- With `protected` data members in the base class, if the base-class implementation changes,
  - we may need to modify all derived classes of that base class.
- Such software is said to be fragile or brittle,
  - a small change in the base class can "break" derived-class implementation.

Base class data members are `protected`

```
// calculate earnings
double BasePlusCommissionEmployee::earnings() const
{
   // derived class cannot access the base class's private data
   return baseSalary + ( commissionRate * grossSales );
} // end function earnings

// print BasePlusCommissionEmployee object
void BasePlusCommissionEmployee::print() const
{
   // derived class cannot access the base class's private data
   cout << "base-salaried commission employee: " << firstName << ' '
      << lastName << "\nsocial security number: " << socialSecurityNumber
      << "\ngross sales: " << grossSales
      << "\ncommission rate: " << commissionRate
      << "\nbase salary: " << baseSalary;
} // end function print
```

```
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.)

Inheritance Hierarchy Using `private` Data

- In the `CommissionEmployee` constructor implementation,
  - we use member initializers to set the values of members `firstName`, `lastName` and `socialSecurityNumber`.
- We show how derived-class `BasePlusCommissionEmployee` can invoke non-`private` base-class member functions
  - `setFirstName`, `getFirstName`, `setLastName`, `getLastName`, `setSocialSecurityNumber` and `getSocialSecurityNumber` to manipulate these data members.

```
1  // Fig. 12.14: 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    : firstName( first ), lastName( last ), socialSecurityNumber( ssn )
12 {
13    setGrossSales( sales ); // validate and store gross sales
14    setCommissionRate( rate ); // validate and store commission rate
15 } // end CommissionEmployee constructor
16
```

Fig. 12.14 | CommissionEmployee class implementation file: CommissionEmployee class uses member functions to manipulate its private data. (Part 1 of 6.)

```
17  // set first name
18  void CommissionEmployee::setFirstName( const string &first )
19  {
20     firstName = first; // should validate
21  } // end function setFirstName
22
23  // return first name
24  string CommissionEmployee::getFirstName() const
25  {
26     return firstName;
27  } // end function getFirstName
28
29  // set last name
30  void CommissionEmployee::setLastName( const string &last )
31  {
32     lastName = last; // should validate
33  } // end function setLastName
34
```

Fig. 12.14 | CommissionEmployee class implementation file: CommissionEmployee class uses member functions to manipulate its private data. (Part 2 of 6.)

```
35  // return last name
36  string CommissionEmployee::getLastName() const
37  {
38     return lastName;
39  } // end function getLastName
40
41  // set social security number
42  void CommissionEmployee::setSocialSecurityNumber( const string &ssn )
43  {
44     socialSecurityNumber = ssn; // should validate
45  } // end function setSocialSecurityNumber
46
47  // return social security number
48  string CommissionEmployee::getSocialSecurityNumber() const
49  {
50     return socialSecurityNumber;
51  } // end function getSocialSecurityNumber
52
```

Fig. 12.14 | CommissionEmployee class implementation file: CommissionEmployee class uses member functions to manipulate its private data. (Part 3 of 6.)

```
53  // set gross sales amount
54  void CommissionEmployee::setGrossSales( double sales )
55  {
56     if ( sales >= 0.0 )
57        grossSales = sales;
58     else
59        throw invalid_argument( "Gross sales must be >= 0.0" );
60  } // end function setGrossSales
61
62  // return gross sales amount
63  double CommissionEmployee::getGrossSales() const
64  {
65     return grossSales;
66  } // end function getGrossSales
67
```

Fig. 12.14 | `CommissionEmployee` class implementation file: `CommissionEmployee` class uses member functions to manipulate its `private` data. (Part 4 of 6.)

```
68  // set commission rate
69  void CommissionEmployee::setCommissionRate( double rate )
70  {
71     if ( rate > 0.0 && rate < 1.0 )
72        commissionRate = rate;
73     else
74        throw invalid_argument( "Commission rate must be > 0.0 and < 1.0" );
75  } // end function setCommissionRate
76
77  // return commission rate
78  double CommissionEmployee::getCommissionRate() const
79  {
80     return commissionRate;
81  } // end function getCommissionRate
82
83  // calculate earnings
84  double CommissionEmployee::earnings() const
85  {
86     return getCommissionRate() * getGrossSales();
87  } // end function earnings
88
```

Fig. 12.14 | `CommissionEmployee` class implementation file: `CommissionEmployee` class uses member functions to manipulate its `private` data. (Part 5 of 6.)

```
89  // print CommissionEmployee object
90  void CommissionEmployee::print() const
91  {
92     cout << "commission employee: "
93        << getFirstName() << ' ' << getLastName()
94        << "\nsocial security number: " << getSocialSecurityNumber()
95        << "\ngross sales: " << getGrossSales()
96        << "\ncommission rate: " << getCommissionRate();
97  } // end function print
```

Fig. 12.14 | `CommissionEmployee` class implementation file: `CommissionEmployee` class uses member functions to manipulate its `private` data. (Part 6 of 6.)

Inheritance Hierarchy Using private Data

- Class `BasePlusCommissionEmployee` has several changes to its member-function implementations
  - distinguish it from the previous version of the class.
- Member functions `earnings` and `print`
  - each invoke `getBaseSalary` to obtain the base salary value.

```
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
```

Fig. 12.15 | BasePlusCommissionEmployee class that inherits from class CommissionEmployee but cannot directly access the class's private data. (Part 1 of 3.)

```
83  // calculate earnings
84  double CommissionEmployee::earnings() const
85  {
86     return getCommissionRate() * getGrossSales();
87  } // end function earnings
88
```

```
17  // set base salary
18  void BasePlusCommissionEmployee::setBaseSalary( double salary )
19  {
20     if ( salary >= 0.0 )
21        baseSalary = salary;
22     else
23        throw invalid_argument( "Salary must be >= 0.0" );
24  } // end function setBaseSalary
25
26  // return base salary
27  double BasePlusCommissionEmployee::getBaseSalary() const
28  {
29     return baseSalary;
30  } // end function getBaseSalary
31
32  // calculate earnings
33  double BasePlusCommissionEmployee::earnings() const
34  {
35     return getBaseSalary() + CommissionEmployee::earnings();
36  } // end function earnings
37
```

Fig. 12.15 | BasePlusCommissionE
from class CommissionEmployee bu
class's private data. (Part 2 of 3.)

```
// calculate earnings
double BasePlusCommissionEmployee::earnings() const
{
   // derived class cannot access the base class's private data
   return baseSalary + ( commissionRate * grossSales );
} // end function earnings
```

```
38  // print BasePlusCommissionEmployee object
39  void BasePlusCommissionEmployee::print() const
40  {
41     cout << "base-salaried ";
42
43     // invoke CommissionEmployee's print function
44     CommissionEmployee::print();
45
46     cout << "\nbase salary: " << getBaseSalary();
47  } // end function print
```

Fig. 12.15 | BasePlusCommissionEmployee class that inherits from class CommissionEmployee but cannot directly access the class's private data. (Part 3 of 3.)

```
// print BasePlusCommissionEmployee object
void BasePlusCommissionEmployee::print() const
{
   // derived class cannot access the base class's private data
   cout << "base-salaried commission employee: " << firstName << ' '
      << lastName << "\nsocial security number: " << socialSecurityNumber
      << "\ngross sales: " << grossSales
      << "\ncommission rate: " << commissionRate
      << "\nbase salary: " << baseSalary;
} // end function print
```

Inheritance Hierarchy Using private Data

- Class `BasePlusCommissionEmployee`'s `earnings` function **redefines** class `CommissionEmployee`'s `earnings` member function to calculate the earnings of a base-salaried commission employee.
  - It also calls `CommissionEmployee`'s `earnings` function.
  - Note the syntax used to invoke a redefined base class member function from a derived class
    - place the base-class name and the binary scope resolution operator (`::`) before the base-class member-function name.
  - Good software engineering practice: If an object's member function performs the actions needed by another object, we should call that member function rather than duplicating its code body.
- By using inheritance and by calling member functions that hide the data and ensure consistency,
  - We can efficiently and effectively constructed a well-engineered class.

Base-class member accessibility in a derived class

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

Conclusions: Software Engineering with Inheritance

- We use inheritance to create a new class from an existing one,
  - the new class inherits the data members and member functions of the existing class.
- We can customize the new class to meet our needs
  - including additional members and redefining base-class members.
- The derived-class programmer does this in C++ without accessing the base class's source code.
- The derived class must be able to link to the base class's object code.

Conclusions

- Software developers can develop proprietary classes for sale or license.
- Users then can derive new classes from these library classes rapidly and without accessing the proprietary source code.
- The software developers need to supply the headers along with the object code.
- The availability of substantial and useful class libraries delivers the maximum benefits of software reuse through inheritance.