Chapter 12 Object-Oriented Programming: Inheritance

C++ How to Program, Late Objects Version, 7/e

OBJECTIVES

In this chapter you'll learn:

- To create classes by inheriting from existing classes.
- The notions of base classes and derived classes and the relationships between them.
- The `protected` member access specifier.
- The use of constructors and destructors in inheritance hierarchies.
- The order in which constructors and destructors are called in inheritance hierarchies.
- The differences between `public`, `protected` and `private` inheritance.
- To use inheritance to customize existing software.

12.1 Introduction

- Inheritance is a form of software reuse in which you create a class that absorbs an existing class's data and behaviors and enhances them with new capabilities.
- You can designate that the new class should inherit the members of an existing class.
- This existing class is called the base class, and 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.

12.1 Introduction (cont.)

- A direct base class is the base class from which a derived class explicitly inherits.
- An indirect base class is 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, in which a derived class inherits from multiple (possibly unrelated) base classes.
  - We discuss multiple inheritance in Chapter 24, Other Topics.

12.1 Introduction (cont.)

- C++ offers `public`, `protected` and `private` inheritance.
- In this chapter, we concentrate on `public` inheritance and briefly explain the other two.
- In Chapter 20, Data Structures, we show how `private` inheritance can be used as an alternative to composition.
- The third form, `protected` inheritance, is rarely used.
- 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.

12.1 Introduction (cont.)

- With object-oriented programming, you 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.

12.1 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, but only through non-`private` member functions provided in the base class and inherited into the derived class.

Software Engineering Observation 12.1

Member functions of a derived class cannot directly access `private` *members of the base class.*

Software Engineering Observation 12.2

If a derived class could access its base class's `private` *members, classes that inherit from that derived class could access that data as well. This would propagate access to what should be* `private` *data, and the benefits of information hiding would be lost.*

12.2 Base Classes and Derived Classes

- Often, an object of one class *is an* object of another class, as well.
  - 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.
- Figure 12.1 lists several simple examples of base classes and 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.

12.2 Base Classes and Derived Classes (cont.)

- Because every derived-class object *is an* object of its base class, and one base class can have many derived classes, the set of objects represented by a base class typically is larger than the set of objects represented by any of its derived classes.
- A base class exists in a hierarchical relationship with its derived classes.
- Although classes can exist independently, once they're employed in inheritance relationships, they become affiliated with other classes.
- A class becomes either a base class—supplying members to other classes, a derived class—inheriting its members from other classes, or both.

12.2 Base Classes and Derived Classes (cont.)

- Let's develop a simple inheritance hierarchy with five levels (represented by the UML class diagram in Fig. 12.2).
- 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.

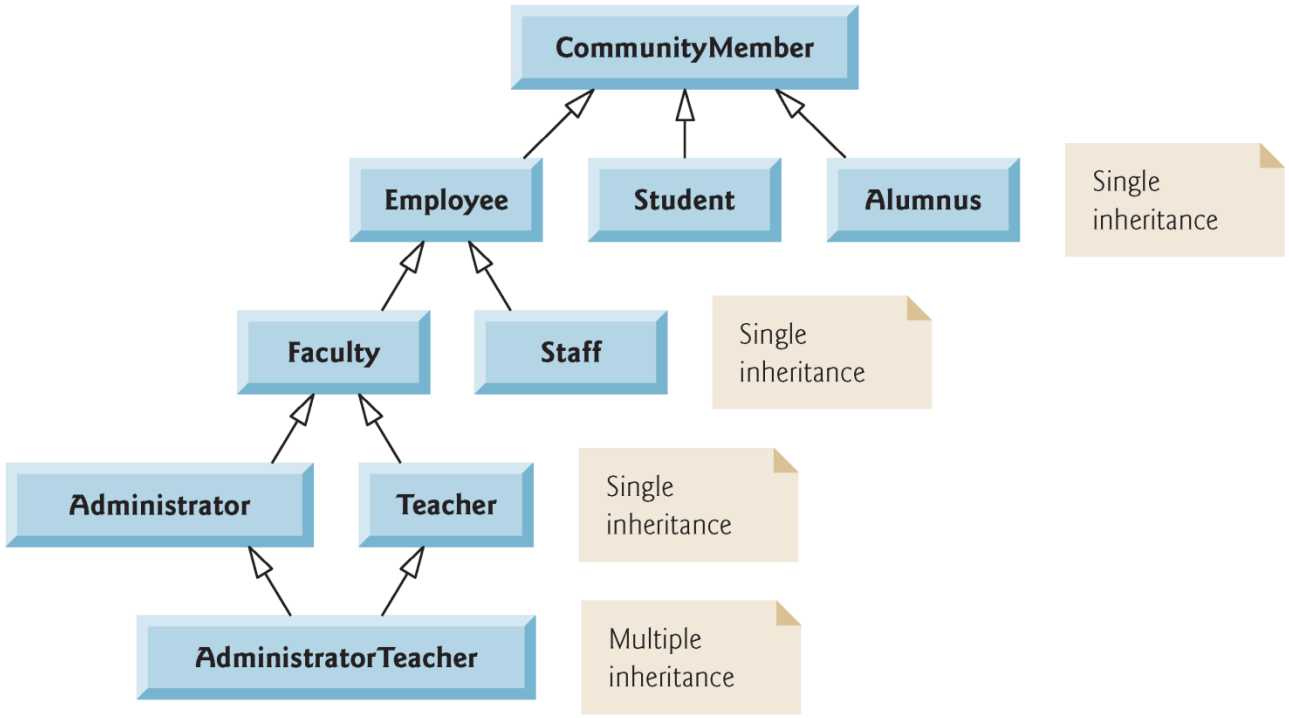

Fig. 12.2 | Inheritance hierarchy for university `CommunityMembers`.

12.2 Base Classes and Derived Classes (cont.)

- Each arrow in the hierarchy (Fig. 12.2) 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`.

12.2 Base Classes and Derived Classes (cont.)

- Consider the `Shape` inheritance hierarchy in Fig. 12.3.
- 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.
- As in Fig. 12.2, we can follow the arrows from the bottom of the diagram to the topmost base class in this class hierarchy to identify several *is-a* relationships.

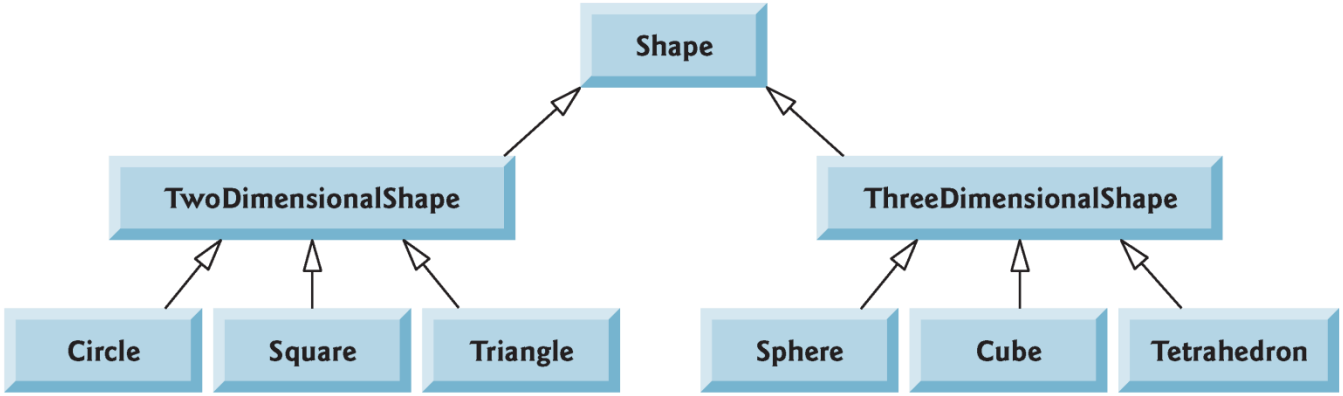

Fig. 12.3 | Inheritance hierarchy for Shapes.

12.3 protected Members

- Chapter 3 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.
- In this section, 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.

12.3 protected Members (cont.)

- 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 (`::`).

12.4 Relationship between Base Classes and Derived Classes

- In this section, we use an inheritance hierarchy containing types of employees in a company's payroll application to 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, while base-salaried commission employees (who will be represented as objects of a derived class) receive a base salary plus a percentage of their sales.

12.4.1 Creating and Using a `CommissionEmployee` Class

- `CommissionEmployee`'s class definition (Figs. 12.4–12.5).
- `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 file. (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 file. (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
19 // set first name
20 void CommissionEmployee::setFirstName( const string &first )
21 {
22    firstName = first; // should validate
23 } // end function setFirstName
```

Fig. 12.5 | Implementation file for `CommissionEmployee` class that represents an employee who is paid a percentage of gross sales. (Part 1 of 5.)

```
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
37   // return last name
38   string CommissionEmployee::getLastName() const
39   {
40      return lastName;
41   } // end function getLastName
42
```

Fig. 12.5 | Implementation file for `CommissionEmployee` class that represents an employee who is paid a percentage of gross sales. (Part 2 of 5.)

```
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
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
```

Fig. 12.5 | Implementation file for `CommissionEmployee` class that represents an employee who is paid a percentage of gross sales. (Part 3 of 5.)

```
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
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
```

Fig. 12.5 | Implementation file for `CommissionEmployee` class that represents an employee who is paid a percentage of gross sales. (Part 4 of 5.)

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

12.4.1 Creating and Using a `CommissionEmployee` Class (cont.)

- 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 earn-ings.
- Member function `print` displays the values of a `CommissionEmployee` object's data members.
- Figure 12.6 tests class `CommissionEmployee`.

```
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
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;
```

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

```
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.)

12.4.2 Creating a `BasePlusCommissionEmployee` 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` (Figs. 12.7–12.8), which contains a first name, last name, social security number, gross sales amount, commission rate and base salary.

```
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
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
```

Fig. 12.7 | `BasePlusCommissionEmployee` class header file. (Part 1 of 2.)

```
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
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 file. (Part 2 of 2.)

```
1  // Fig. 12.8: 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 {
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    setBaseSalary( salary ); // validate and store base salary
18 } // end BasePlusCommissionEmployee constructor
19
```

Fig. 12.8 | `BasePlusCommissionEmployee` class represents an employee who receives a base salary in addition to a commission. (Part 1 of 5.)

```
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
38  // return last name
39  string BasePlusCommissionEmployee::getLastName() const
40  {
41     return lastName;
42  } // end function getLastName
```

Fig. 12.8 | `BasePlusCommissionEmployee` class represents an employee who receives a base salary in addition to a commission. (Part 2 of 5.)

```
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
57 // set gross sales amount
58 void BasePlusCommissionEmployee::setGrossSales( double sales )
59 {
60    grossSales = ( sales < 0.0 ) ? 0.0 : sales;
61 } // end function setGrossSales
62
```

Fig. 12.8 | `BasePlusCommissionEmployee` class represents an employee who receives a base salary in addition to a commission. (Part 3 of 5.)

```
63  // return gross sales amount
64  double BasePlusCommissionEmployee::getGrossSales() const
65  {
66     return grossSales;
67  } // end function getGrossSales
68
69  // set commission rate
70  void BasePlusCommissionEmployee::setCommissionRate( double rate )
71  {
72     commissionRate = ( rate > 0.0 && rate < 1.0 ) ? rate : 0.0;
73  } // end function setCommissionRate
74
75  // return commission rate
76  double BasePlusCommissionEmployee::getCommissionRate() const
77  {
78     return commissionRate;
79  } // end function getCommissionRate
80
81  // set base salary
82  void BasePlusCommissionEmployee::setBaseSalary( double salary )
83  {
84     baseSalary = ( salary < 0.0 ) ? 0.0 : salary;
85  } // end function setBaseSalary
```

Fig. 12.8 | `BasePlusCommissionEmployee` class represents an employee who receives a base salary in addition to a commission. (Part 4 of 5.)

```
86
87    // return base salary
88    double BasePlusCommissionEmployee::getBaseSalary() const
89    {
90       return baseSalary;
91    } // end function getBaseSalary
92
93    // calculate earnings
94    double BasePlusCommissionEmployee::earnings() const
95    {
96       return baseSalary + ( commissionRate * grossSales );
97    } // end function earnings
98
99    // print BasePlusCommissionEmployee object
100   void BasePlusCommissionEmployee::print() const
101   {
102      cout << "base-salaried commission employee: " << firstName << ' '
103         << lastName << "\nsocial security number: " << socialSecurityNumber
104         << "\ngross sales: " << grossSales
105         << "\ncommission rate: " << commissionRate
106         << "\nbase salary: " << baseSalary;
107   } // end function print
```

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

12.4.2 Creating a `BasePlusCommissionEmployee` Class Without Using Inheritance (cont.)

- The `BasePlusCommissionEmployee` header file (Fig. 12.7) specifies class `BasePlusCommissionEmployee`'s `public` services, which include the `BasePlusCommissionEmployee` constructor and member functions `earnings` and `print`.
- Lines 16–32 declare `public` *get* and *set* functions for the class's `private` data members `firstName`, `lastName`, `socialSecurityNumber`, `grossSales`, `commissionRate` and `baseSalary`.
- Note the similarity between this class and class `CommissionEmployee` (Figs. 12.4–12.5)—in this example, we won't yet exploit that similarity.
- Class `BasePlusCommissionEmployee`'s `earnings` member function computes the earnings of a base-salaried commission employee.
- Figure 12.9 tests class `BasePlusCommissionEmployee`.

```
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
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()
```

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

```
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.)

12.4.2 Creating a `BasePlusCommissionEmployee` Class Without Using Inheritance (cont.)

- Most of the code for class `BasePlusCommissionEmployee` (Figs. 12.7–12.8) is similar, if not identical, to the code for class `CommissionEmployee` (Figs. 12.4–12.5).
- 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.

12.4.2 Creating a `BasePlusCommissionEmployee` Class Without Using Inheritance (cont.)

- 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`.

12.4.2 Creating a `BasePlusCommissionEmployee` Class Without Using Inheritance (cont.)

- We literally copied code from class `CommissionEmployee` and pasted it into class `BasePlusCommissionEmployee`, then modified class `BasePlusCommissionEmployee` to include a base salary and member functions that manipulate the base salary.
- 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.

Software Engineering Observation 12.3

Copying and pasting code from one class to another can spread errors across multiple source code files. To avoid duplicating code (and possibly errors), use inheritance, rather than the "copy-and-paste" approach, in situations where you want one class to "absorb" the data members and member functions of another class.

Software Engineering Observation 12.4

With inheritance, the common data members and member functions of all the classes in the hierarchy are declared in a base class. When changes are required for these common features, you need to make the changes only in the base class—derived classes then inherit the changes. Without inheritance, changes would need to be made to all the source code files that contain a copy of the code in question.

12.4.3 Creating a `CommissionEmployee`–`BasePlusCommissionEmployee` Inheritance Hierarchy

- Now we create and test a new `BasePlusCommissionEmployee` class (Figs. 12.10–12.11) that derives from class `CommissionEmployee` (Figs. 12.4–12.5).
- In this example, a `BasePlus-CommissionEmployee` object *is a* `CommissionEmployee` (because inheritance passes on the capabilities of class `CommissionEmployee`), but class `BasePlusCommission-Employee` also has data member `baseSalary` (Fig. 12.10, line 23).
- The colon (`:`) in line 11 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.

12.4.3 Creating a `CommissionEmployee`–`BasePlusCommissionEmployee` Inheritance Hierarchy (cont.)

- Destructors, too, are not inherited
- Thus, the `public` services of `BasePlusCommissionEmployee` include its constructor and the `public` member functions inherited from class `CommissionEmployee`—although 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
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
```

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

```
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
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. 12.11 | `BasePlusCommissionEmployee` implementation file: `private` base-class data cannot be accessed from derived class. (Part 1 of 4.)

```
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     // derived class cannot access the base class's private data
33     return baseSalary + ( commissionRate * grossSales );
34  } // end function earnings
35
36  // print BasePlusCommissionEmployee object
37  void BasePlusCommissionEmployee::print() const
38  {
39     // derived class cannot access the base class's private data
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. 12.11 | `BasePlusCommissionEmployee` implementation file: `private` base-class data cannot be accessed from derived class. (Part 2 of 4.)

```
C:\cpphtp7_examples\ch12\Fig12_10_11\BasePlusCommissionEmployee.cpp(33) :
   error C2248: 'CommissionEmployee::commissionRate' :
   cannot access private member declared in class 'CommissionEmployee'

C:\cpphtp7_examples\ch12\Fig12_10_11\BasePlusCommissionEmployee.cpp(33) :
   error C2248: 'CommissionEmployee::grossSales' :
   cannot access private member declared in class 'CommissionEmployee'

C:\cpphtp7_examples\ch12\Fig12_10_11\BasePlusCommissionEmployee.cpp(40) :
   error C2248: 'CommissionEmployee::firstName' :
   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 3 of 4.)

```
C:\cpphtp7_examples\ch12\Fig12_10_11\BasePlusCommissionEmployee.cpp(41) :
   error C2248: 'CommissionEmployee::lastName' :
   cannot access private member declared in class 'CommissionEmployee'

C:\cpphtp7_examples\ch12\Fig12_10_11\BasePlusCommissionEmployee.cpp(41) :
   error C2248: 'CommissionEmployee::socialSecurityNumber' :
   cannot access private member declared in class 'CommissionEmployee'

C:\cpphtp7_examples\ch12\Fig12_10_11\BasePlusCommissionEmployee.cpp(42) :
   error C2248: 'CommissionEmployee::grossSales' :
   cannot access private member declared in class 'CommissionEmployee'

C:\cpphtp7_examples\ch12\Fig12_10_11\BasePlusCommissionEmployee.cpp(43) :
   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 4 of 4.)

12.4.3 Creating a `CommissionEmployee`–`BasePlusCommissionEmployee` Inheritance Hierarchy (cont.)

- Figure 12.11 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.

Common Programming Error 12.1

When a derived-class constructor calls a base-class constructor, the arguments passed to the base-class constructor must be consistent with the number and types of parameters specified in one of the base-class constructors; otherwise, a compilation error occurs.

Performance Tip 12.1

In a derived-class constructor, initializing member objects and invoking base-class constructors explicitly in the member initializer list prevents duplicate initialization in which a default constructor is called, then data members are modified again in the derived-class constructor's body.

12.4.3 Creating a `CommissionEmployee`–`BasePlusCommissionEmployee` Inheritance Hierarchy (cont.)

- The compiler generates errors for line 33 of Fig. 12.11 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.
- We used red text in Fig. 12.11 to indicate erroneous code.
- The compiler issues additional errors in lines 40–43 of `BasePlus-Commission-Employee`'s `print` member function for the same reason.
- C++ rigidly enforces restrictions on accessing `private` data members, so that even a derived class (which is intimately related to its base class) cannot access the base class's `private` data.
- We purposely included the erroneous code in Fig. 12.11 to emphasize that a derived class's member functions cannot access its base class's `private` data.

12.4.3 Creating a `CommissionEmployee`–`BasePlusCommissionEmployee` Inheritance Hierarchy (cont.)

- The errors in `BasePlusCommissionEmployee` could have been prevented by using the *get* member functions inherited from class `CommissionEmployee`.
- For example, line 33 could have invoked `getCommissionRate` and `getGrossSales` to access `CommissionEmployee`'s `private` data members `commissionRate` and `grossSales`, respectively.
- Similarly, lines 40–43 could have used appropriate *get* member functions to retrieve the values of the base class's data members.

12.4.3 Creating a `CommissionEmployee`–`BasePlusCommissionEmployee` Inheritance Hierarchy (cont.)

- Notice that we `#include` the base class's header file in the derived class's header file (line 8 of Fig. 12.10).
- This is necessary for three reasons.
  - The derived class uses the base class's name in line 10, 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.

12.4.3 Creating a `CommissionEmployee`–`BasePlusCommissionEmployee` Inheritance Hierarchy (cont.)

- In Section 3.8, we discussed the linking process for creating an executable `GradeBook` application.
- The linking process is similar for a program that uses classes in an inheritance hierarchy.
- The process requires the object code for all classes used in the program and the object code for the direct and indirect base classes of any derived classes used by the program.
- The code is also linked with the object code for any C++ Standard Library classes used in the classes or the client code.

12.4.4 CommissionEmployee–BasePlusCommissionEmployee 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 friends of the base class and by members and friends of any classes derived from that base class.

Good Programming Practice 12.1

Declare `public` *members first,* `protected` *members second and* `private` *members last.*

12.4.4 CommissionEmployee–BasePlusCommissionEmployee Inheritance Hierarchy Using protected Data (cont.)

- Class `CommissionEmployee` (Figs. 12.12–12.13) now declares data members `firstName`, `lastName`, `socialSecurityNumber`, `grossSales` and `commissionRate` as `protected` (Fig. 12.12, lines 32–37) rather than `private`.
- The member-function implementations in Fig. 12.13 are identical to those in Fig. 12.5.

```
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.)

12.4.4 CommissionEmployee–BasePlusCommissionEmployee Inheritance Hierarchy Using protected Data (cont.)

- The version of class `BasePlusCommissionEmployee` in Figs. 12.14–12.15 inherits from class `CommissionEmployee` in Figs. 12.12–12.13.
- Objects of class `BasePlusCommissionEmployee` can access inherited data members that are declared `protected` in class `CommissionEmployee` (i.e., 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 in Fig. 12.15 (lines 30–34 and 37–45, respectively).
- Objects of a derived class also can access `protected` members in any of that derived class's indirect base classes.

```
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
```

Fig. 12.14 | `BasePlusCommissionEmployee` class header file. (Part 1 of 2.)

```
22   private:
23      double baseSalary; // base salary
24   }; // end class BasePlusCommissionEmployee
25
26   #endif
```

Fig. 12.14 | `BasePlusCommissionEmployee` class header file. (Part 2 of 2.)

```
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 `BasePlusCommissionEmployee` class that inherits `protected` data from `CommissionEmployee`. (Part 2 of 2.)

12.4.4 CommissionEmployee–BasePlusCommissionEmployee Inheritance Hierarchy Using protected Data (cont.)

- Figure 12.16 uses a `BasePlusCommissionEmployee` object to perform the same tasks that Fig. 12.9 performed on an object of the first version of class `BasePlusCommissionEmployee` (Figs. 12.7–12.8).
- The code and outputs of the two programs are identical.
- The code for class `BasePlusCommissionEmployee`, which is 71 lines, is considerably shorter than the code for the noninherited version of the class, which is 152 lines, because the inherited version absorbs part of its functionality from `CommissionEmployee`, whereas the noninherited version does not absorb any functionality.
- Also, there is now only one copy of the `CommissionEmployee` functionality declared and defined in class `CommissionEmployee`.
  - Makes the source code easier to maintain, modify and debug.

```
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.)

12.4.4 `CommissionEmployee–BasePlusCommissionEmployee` Inheritance Hierarchy Using `protected` Data (cont.)

- Inheriting `protected` data members slightly increases performance, because 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, and leave code optimization issues to the compiler.

12.4.4 `CommissionEmployee–BasePlusCommissionEmployee` Inheritance Hierarchy Using `protected` Data (cont.)

- 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, because a small change in the base class can "break" derived-class implementation.

Software Engineering Observation 12.5

It's appropriate to use the `protected` *access specifier when a base class should provide a service (i.e., a member function) only to its derived classes and* `friend`*s.*

Software Engineering Observation 12.6

Declaring base-class data members `private` *(as opposed to declaring them* `protected`*) enables you to change the base-class implementation without having to change derived-class implementations.*

Error-Prevention Tip 12.1

When possible, avoid including `protected` *data members in a base class. Rather, include non-*`private` *member functions that access* `private` *data members, ensuring that the object maintains a consistent state.*

12.4.5 CommissionEmployee–BasePlusCommissionEmployee Inheritance Hierarchy Using private Data

- We now reexamine our hierarchy once more, this time using the best software engineering practices.
- Class `CommissionEmployee` (Figs. 12.17–12.18) now declares data members `firstName`, `lastName`, `socialSecurityNumber`, `grossSales` and `commissionRate` as `private` (Fig. 12.17, lines 32–37) and provides `public` member functions `setFirstName`, `getFirstName`, `setLastName`, `getLastName`, `setSocialSecurityNumber`, `getSocialSecurityNumber`, `setGrossSales`, `getGrossSales`, `setCommissionRate`, `getCommissionRate`, `earnings` and `print` for manipulating these values.
- Derived class `BasePlusCommissionEmployee` (Figs. 12.19–12.20) inherits `CommissionEmployee`'s member functions and can access the `private` base-class members via the inherited non-`private` member functions.

```
1  // Fig. 12.17: CommissionEmployee.h
2  // CommissionEmployee class definition with good software engineering.
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
```

Fig. 12.17 | `CommissionEmployee` class defined using good software engineering practices. (Part 1 of 2.)

```
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   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. 12.17 | `CommissionEmployee` class defined using good software engineering practices. (Part 2 of 2.)

12.4.5 CommissionEmployee–BasePlusCommissionEmployee Inheritance Hierarchy Using private Data (cont.)

- In the `CommissionEmployee` constructor implementation (Fig. 12.18, lines 8–15), we use member initializers to set the values of members `firstName`, `lastName` and `socialSecurityNumber`.
- We show how derived-class `BasePlusCommissionEmployee` (Figs. 12.19–12.20) can invoke non-`private` base-class member functions (`setFirstName`, `getFirstName`, `setLastName`, `getLastName`, `setSocialSecurityNumber` and `getSocialSecurityNumber`) to manipulate these data members.

Performance Tip 12.2

Using a member function to access a data member's value can be slightly slower than accessing the data directly. However, today's optimizing compilers are carefully designed to perform many optimizations implicitly (such as inlining set *and* get *member-function calls). You should write code that adheres to proper software engineering principles, and leave optimization to the compiler. A good rule is, "Do not second-guess the compiler."*

```
1  // Fig. 12.18: 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
17 // set first name
18 void CommissionEmployee::setFirstName( const string &first )
19 {
20    firstName = first; // should validate
21 } // end function setFirstName
```

Fig. 12.18 | `CommissionEmployee` class implementation file: `CommissionEmployee` class uses member functions to manipulate its `private` data. (Part 1 of 5.)

```
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
35  // return last name
36  string CommissionEmployee::getLastName() const
37  {
38     return lastName;
39  } // end function getLastName
40
```

Fig. 12.18 | `CommissionEmployee` class implementation file: `CommissionEmployee` class uses member functions to manipulate its `private` data. (Part 2 of 5.)

```
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
53  // set gross sales amount
54  void CommissionEmployee::setGrossSales( double sales )
55  {
56     grossSales = ( sales < 0.0 ) ? 0.0 : sales;
57  } // end function setGrossSales
58
```

Fig. 12.18 | `CommissionEmployee` class implementation file: `CommissionEmployee` class uses member functions to manipulate its `private` data. (Part 3 of 5.)

```
59  // return gross sales amount
60  double CommissionEmployee::getGrossSales() const
61  {
62     return grossSales;
63  } // end function getGrossSales
64
65  // set commission rate
66  void CommissionEmployee::setCommissionRate( double rate )
67  {
68     commissionRate = ( rate > 0.0 && rate < 1.0 ) ? rate : 0.0;
69  } // end function setCommissionRate
70
71  // return commission rate
72  double CommissionEmployee::getCommissionRate() const
73  {
74     return commissionRate;
75  } // end function getCommissionRate
76
```

Fig. 12.18 | `CommissionEmployee` class implementation file: `CommissionEmployee` class uses member functions to manipulate its `private` data. (Part 4 of 5.)

```
77  // calculate earnings
78  double CommissionEmployee::earnings() const
79  {
80     return getCommissionRate() * getGrossSales();
81  } // end function earnings
82
83  // print CommissionEmployee object
84  void CommissionEmployee::print() const
85  {
86     cout << "commission employee: "
87        << getFirstName() << ' ' << getLastName()
88        << "\nsocial security number: " << getSocialSecurityNumber()
89        << "\ngross sales: " << getGrossSales()
90        << "\ncommission rate: " << getCommissionRate();
91  } // end function print
```

Fig. 12.18 | `CommissionEmployee` class implementation file: `CommissionEmployee` class uses member functions to manipulate its `private` data. (Part 5 of 5.)

12.4.5 `CommissionEmployee–BasePlusCommissionEmployee` Inheritance Hierarchy Using `private` Data (cont.)

- Class `BasePlusCommissionEmployee` (Figs. 12.19–12.20) has several changes to its member-function implementations (Fig. 12.20) that distinguish it from the previous version of the class (Figs. 12.14–12.15).
- Member functions `earnings` (Fig. 12.20, lines 30–33) and `print` (lines 36–44) each invoke `getBaseSalary` to obtain the base salary value.

```
1  // Fig. 12.19: 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
```

Fig. 12.19 | `BasePlusCommissionEmployee` class header file. (Part 1 of 2.)

```
22   private:
23      double baseSalary; // base salary
24   }; // end class BasePlusCommissionEmployee
25
26   #endif
```

Fig. 12.19 | `BasePlusCommissionEmployee` class header file. (Part 2 of 2.)

```
1  // Fig. 12.20: 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. 12.20 | `BasePlusCommissionEmployee` class that inherits from class `CommissionEmployee` but cannot directly access the class's `private` data. (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     return getBaseSalary() + CommissionEmployee::earnings();
33  } // end function earnings
34
35  // print BasePlusCommissionEmployee object
36  void BasePlusCommissionEmployee::print() const
37  {
38     cout << "base-salaried ";
39
40     // invoke CommissionEmployee's print function
41     CommissionEmployee::print();
42
43     cout << "\nbase salary: " << getBaseSalary();
44  } // end function print
```

Fig. 12.20 | `BasePlusCommissionEmployee` class that inherits from class `CommissionEmployee` but cannot directly access the class's `private` data. (Part 2 of 2.)

Common Programming Error 12.2

When a base-class member function is redefined in a derived class, the derived-class version often calls the base-class version to do additional work. Failure to use the `::` *operator prefixed with the name of the base class when referencing the base class's member function causes infinite recursion, because the derived-class member function would then call itself.*

12.4.5 CommissionEmployee–BasePlusCommissionEmployee Inheritance Hierarchy Using private Data (cont.)

- Class BasePlusCommissionEmployee's earnings function (Fig. 12.20, lines 30–33) redefines class CommissionEmployee's earnings 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.

12.4.5 `CommissionEmployee–BasePlusCommissionEmployee` Inheritance Hierarchy Using `private` Data (cont.)

- `BasePlusCommissionEmployee`'s `print` function (Fig. 12.20, lines 36–44) redefines class `CommissionEmployee`'s `print` to output the appropriate base-salaried commission employee information. It also calles `CommissionEmployee`'s `print`.
- By using inheritance and by calling member functions that hide the data and ensure consistency, we've efficiently and effectively constructed a well-engineered class.

```
1  // Fig. 12.21: fig12_21.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. 12.21 | Base-class `private` data is accessible to a derived class via `public` or `protected` member function inherited by the 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. 12.21 | Base-class `private` data is accessible to a derived class via `public` or `protected` member function inherited by the 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. 12.21 | Base-class `private` data is accessible to a derived class via `public` or `protected` member function inherited by the derived class. (Part 3 of 3.)

12.5 Constructors and Destructors in Derived Classes

- Instantiating a derived-class object begins a chain of constructor calls in which the derived-class constructor, before performing its own tasks, invokes its direct base class's constructor either explicitly (via a base-class member initializer) or implicitly (calling the base class's default constructor).
- If the base class is derived from another class, the base-class constructor is required to invoke the constructor of the next class up in the hierarchy, and so on.
- The last constructor called in this chain is the constructor of the class at the base of the hierarchy, whose body actually finishes executing first.
- The original derived-class constructor's body finishes executing last.
- Each base-class constructor initializes the base-class data members that the derived-class object inherits.

Software Engineering Observation 12.7

When a program creates a derived-class object, the derived-class constructor immediately calls the base-class constructor, the base-class constructor's body executes, then the derived class's member initializers execute and finally the derived-class constructor's body executes. This process cascades up the hierarchy if it contains more than two levels.

12.5 Constructors and Destructors in Derived Classes (cont.)

- When a derived-class object is destroyed, the program calls that object's destructor.
- This begins a chain (or cascade) of destructor calls in which the derived-class destructor and the destructors of the direct and indirect base classes and the classes' members execute in reverse of the order in which the constructors executed.
- When a derived-class object's destructor is called, the destructor performs its task, then invokes the destructor of the next base class up the hierarchy.
- This process repeats until the destructor of the final base class at the top of the hierarchy is called.
- Then the object is removed from memory.

Software Engineering Observation 12.8

Suppose that we create an object of a derived class where both the base class and the derived class contain (via composition) objects of other classes. When an object of that derived class is created, first the constructors for the base class's member objects execute, then the base-class constructor executes, then the constructors for the derived class's member objects execute, then the derived class's constructor executes. Destructors for derived-class objects are called in the reverse of the order in which their corresponding constructors are called.

12.5 Constructors and Destructors in Derived Classes (cont.)

- Base-class constructors, destructors and overloaded assignment operators (see Chapter 11, Operator Overloading) are not inherited by derived classes.
- Derived-class constructors, destructors and overloaded assignment operators can call base-class constructors, destructors and overloaded assignment operators.
- Our next example defines class `CommissionEmployee` (Figs. 12.22–12.23) and class `BasePlusCommissionEmployee` (Figs. 12.24–12.25) with constructors and destructors that each print a message when invoked.
  - These messages demonstrate the order in which the constructors and destructors are called for objects in an inheritance hierarchy.

```
1  // Fig. 12.22: 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    ~CommissionEmployee(); // destructor
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
```

Fig. 12.22 | CommissionEmployee class header file. (Part 1 of 2.)

```
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
31      double earnings() const; // calculate earnings
32      void print() const; // print CommissionEmployee object
33   private:
34      string firstName;
35      string lastName;
36      string socialSecurityNumber;
37      double grossSales; // gross weekly sales
38      double commissionRate; // commission percentage
39   }; // end class CommissionEmployee
40
41   #endif
```

Fig. 12.22 | `CommissionEmployee` class header file. (Part 2 of 2.)

```
1  // Fig. 12.23: 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
16    cout << "CommissionEmployee constructor: " << endl;
17    print();
18    cout << "\n\n";
19 } // end CommissionEmployee constructor
20
```

Fig. 12.23 | `CommissionEmployee`'s constructor and destructor output text. (Part 1 of 5.)

```
21   // destructor
22   CommissionEmployee::~CommissionEmployee()
23   {
24      cout << "CommissionEmployee destructor: " << endl;
25      print();
26      cout << "\n\n";
27   } // end CommissionEmployee destructor
28
29   // set first name
30   void CommissionEmployee::setFirstName( const string &first )
31   {
32      firstName = first; // should validate
33   } // end function setFirstName
34
35   // return first name
36   string CommissionEmployee::getFirstName() const
37   {
38      return firstName;
39   } // end function getFirstName
40
```

Fig. 12.23 | `CommissionEmployee`'s constructor and destructor output text. (Part 2 of 5.)

```
41   // set last name
42   void CommissionEmployee::setLastName( const string &last )
43   {
44      lastName = last; // should validate
45   } // end function setLastName
46
47   // return last name
48   string CommissionEmployee::getLastName() const
49   {
50      return lastName;
51   } // end function getLastName
52
53   // set social security number
54   void CommissionEmployee::setSocialSecurityNumber( const string &ssn )
55   {
56      socialSecurityNumber = ssn; // should validate
57   } // end function setSocialSecurityNumber
58
59   // return social security number
60   string CommissionEmployee::getSocialSecurityNumber() const
61   {
62      return socialSecurityNumber;
63   } // end function getSocialSecurityNumber
```

Fig. 12.23 | `CommissionEmployee`'s constructor and destructor output text. (Part 3 of 5.)

```
64
65   // set gross sales amount
66   void CommissionEmployee::setGrossSales( double sales )
67   {
68      grossSales = ( sales < 0.0 ) ? 0.0 : sales;
69   } // end function setGrossSales
70
71   // return gross sales amount
72   double CommissionEmployee::getGrossSales() const
73   {
74      return grossSales;
75   } // end function getGrossSales
76
77   // set commission rate
78   void CommissionEmployee::setCommissionRate( double rate )
79   {
80      commissionRate = ( rate > 0.0 && rate < 1.0 ) ? rate : 0.0;
81   } // end function setCommissionRate
82
```

Fig. 12.23 | `CommissionEmployee`'s constructor and destructor output text. (Part 4 of 5.)

```
83  // return commission rate
84  double CommissionEmployee::getCommissionRate() const
85  {
86     return commissionRate;
87  } // end function getCommissionRate
88
89  // calculate earnings
90  double CommissionEmployee::earnings() const
91  {
92     return getCommissionRate() * getGrossSales();
93  } // end function earnings
94
95  // print CommissionEmployee object
96  void CommissionEmployee::print() const
97  {
98     cout << "commission employee: "
99        << getFirstName() << ' ' << getLastName()
100       << "\nsocial security number: " << getSocialSecurityNumber()
101       << "\ngross sales: " << getGrossSales()
102       << "\ncommission rate: " << getCommissionRate();
103 } // end function print
```

Fig. 12.23 | `CommissionEmployee`'s constructor and destructor output text. (Part 5 of 5.)

```
1  // Fig. 12.24: 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    ~BasePlusCommissionEmployee(); // destructor
17
18    void setBaseSalary( double ); // set base salary
19    double getBaseSalary() const; // return base salary
20
21    double earnings() const; // calculate earnings
22    void print() const; // print BasePlusCommissionEmployee object
```

Fig. 12.24 | `BasePlusCommissionEmployee` class header file. (Part 1 of 2.)

```
23   private:
24      double baseSalary; // base salary
25   }; // end class BasePlusCommissionEmployee
26
27   #endif
```

Fig. 12.24 | `BasePlusCommissionEmployee` class header file. (Part 2 of 2.)

```
1  // Fig. 12.25: 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
16    cout << "BasePlusCommissionEmployee constructor: " << endl;
17    print();
18    cout << "\n\n";
19 } // end BasePlusCommissionEmployee constructor
20
```

Fig. 12.25 | `BasePlusCommissionEmployee`'s constructor and destructor output text. (Part 1 of 3.)

```
21   // destructor
22   BasePlusCommissionEmployee::~BasePlusCommissionEmployee()
23   {
24      cout << "BasePlusCommissionEmployee destructor: " << endl;
25      print();
26      cout << "\n\n";
27   } // end BasePlusCommissionEmployee destructor
28
29   // set base salary
30   void BasePlusCommissionEmployee::setBaseSalary( double salary )
31   {
32      baseSalary = ( salary < 0.0 ) ? 0.0 : salary;
33   } // end function setBaseSalary
34
35   // return base salary
36   double BasePlusCommissionEmployee::getBaseSalary() const
37   {
38      return baseSalary;
39   } // end function getBaseSalary
40
```

Fig. 12.25 | `BasePlusCommissionEmployee`'s constructor and destructor output text. (Part 2 of 3.)

```
41   // calculate earnings
42   double BasePlusCommissionEmployee::earnings() const
43   {
44      return getBaseSalary() + CommissionEmployee::earnings();
45   } // end function earnings
46
47   // print BasePlusCommissionEmployee object
48   void BasePlusCommissionEmployee::print() const
49   {
50      cout << "base-salaried ";
51
52      // invoke CommissionEmployee's print function
53      CommissionEmployee::print();
54
55      cout << "\nbase salary: " << getBaseSalary();
56   } // end function print
```

Fig. 12.25 | `BasePlusCommissionEmployee`'s constructor and destructor output text. (Part 3 of 3.)

```
1  // Fig. 12.26: fig12_26.cpp
2  // Display order in which base-class and derived-class constructors
3  // and destructors are called.
4  #include <iostream>
5  #include <iomanip>
6  #include "BasePlusCommissionEmployee.h"
7  using namespace std;
8
9  int main()
10 {
11    // set floating-point output formatting
12    cout << fixed << setprecision( 2 );
13
14    { // begin new scope
15       CommissionEmployee employee1(
16          "Bob", "Lewis", "333-33-3333", 5000, .04 );
17    } // end scope
18
19    cout << endl;
20    BasePlusCommissionEmployee
21       employee2( "Lisa", "Jones", "555-55-5555", 2000, .06, 800 );
22
```

Fig. 12.26 | Constructor and destructor call order. (Part 1 of 4.)

```
23      cout << endl;
24      BasePlusCommissionEmployee
25         employee3( "Mark", "Sands", "888-88-8888", 8000, .15, 2000 );
26      cout << endl;
27   } // end main
```

```
CommissionEmployee constructor:
commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
commission rate: 0.04

CommissionEmployee destructor:
commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
commission rate: 0.04


CommissionEmployee constructor:
commission employee: Lisa Jones
social security number: 555-55-5555
gross sales: 2000.00
commission rate: 0.06
```

Fig. 12.26 | Constructor and destructor call order. (Part 2 of 4.)

```
BasePlusCommissionEmployee constructor:
base-salaried commission employee: Lisa Jones
social security number: 555-55-5555
gross sales: 2000.00
commission rate: 0.06
base salary: 800.00


CommissionEmployee constructor:
commission employee: Mark Sands
social security number: 888-88-8888
gross sales: 8000.00
commission rate: 0.15

BasePlusCommissionEmployee constructor:
base-salaried commission employee: Mark Sands
social security number: 888-88-8888
gross sales: 8000.00
commission rate: 0.15
base salary: 2000.00
```

Fig. 12.26 | Constructor and destructor call order. (Part 3 of 4.)

```
BasePlusCommissionEmployee destructor:
base-salaried commission employee: Mark Sands
social security number: 888-88-8888
gross sales: 8000.00
commission rate: 0.15
base salary: 2000.00

CommissionEmployee destructor:
commission employee: Mark Sands
social security number: 888-88-8888
gross sales: 8000.00
commission rate: 0.15

BasePlusCommissionEmployee destructor:
base-salaried commission employee: Lisa Jones
social security number: 555-55-5555
gross sales: 2000.00
commission rate: 0.06
base salary: 800.00

CommissionEmployee destructor:
commission employee: Lisa Jones
social security number: 555-55-5555
gross sales: 2000.00
commission rate: 0.06
```

Fig. 12.26 | Constructor and destructor call order. (Part 4 of 4.)

12.6 public, protected and private Inheritance

- When deriving a class from a base class, the base class may be inherited through `pub-lic`, `protected` or `private` inheritance.
- Use of `protected` and `private` inheritance is rare, and each should be used only with great care; we normally use `public` inheritance in this book.
- Figure 12.27 summarizes for each type of inheritance the accessibility of base-class members in a derived class.
- The first column contains the base-class access specifiers.
- A base class's `private` members are never accessible directly from a derived class, but can be accessed through calls to the `public` and `protected` members of the base 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. 12.27 | Summary of base-class member accessibility in a derived class.

12.7 Software Engineering with Inheritance

- When 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, as described in Fig. 12.27.
- We can customize the new class to meet our needs by including additional members and by 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.

12.7 Software Engineering with Inheritance (cont)

- ISVs can develop proprietary classes for sale or license and make these classes available to users in object-code format.
- Users can derive new classes from these library classes rapidly and without accessing the ISVs' proprietary source code.
- All the ISVs need to supply with the object code are the header files.
- The availability of substantial and useful class libraries delivers the maxi-mum benefits of software reuse through inheritance.

Software Engineering Observation 12.9

At the design stage in an object-oriented system, the designer often determines that certain classes are closely related. The designer should "factor out" common attributes and behaviors and place these in a base class, then use inheritance to form derived classes, endowing them with capabilities beyond those inherited from the base class.

Software Engineering Observation 12.10

The creation of a derived class does not affect its base class's source code. Inheritance preserves the integrity of a base class.

Performance Tip 12.3

If classes produced through inheritance are larger than they need to be (i.e., contain too much functionality), memory and processing resources might be wasted. Inherit from the class whose functionality is "closest" to what's needed.