

Instructor Note: *C How to Program*, Chapter 19 is a copy of *C++ How to Program*, 9/e Chapter 11. We have not renumbered the PowerPoint Slides.

# Chapter 11

## Object-Oriented Programming:

### Inheritance

C++ How to Program, 9/e, GE

## OBJECTIVES

In this chapter you'll learn:

- What inheritance is and how it promotes software reuse.
- 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.

- 11.1** Introduction
- 11.2** Base Classes and Derived Classes
- 11.3** Relationship between Base and Derived Classes
  - 11.3.1 Creating and Using a `CommissionEmployee` Class
  - 11.3.2 Creating a `BasePlusCommission-Employee` Class Without Using Inheritance
  - 11.3.3 Creating a `CommissionEmployee–BasePlusCommissionEmployee` Inheritance Hierarchy
  - 11.3.4 `CommissionEmployee–BasePlusCommissionEmployee` Inheritance Hierarchy Using `protected` Data
  - 11.3.5 `CommissionEmployee–BasePlusCommissionEmployee` Inheritance Hierarchy Using `private` Data
- 11.4** Constructors and Destructors in Derived Classes
- 11.5** `public`, `protected` and `private` Inheritance
- 11.6** Software Engineering with Inheritance
- 11.7** Wrap-Up

## 11.1 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.
- 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.

## 11.1 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.

## 11.2 Base Classes and Derived Classes

- Figure 11.1 lists several simple examples of base classes and derived classes.
  - Base classes tend to be *more general* and derived classes tend to be *more specific*.
- 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.
- Inheritance relationships form **class hierarchies**.

## 11.2 Base Classes and Derived Classes (cont.)

- 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*.

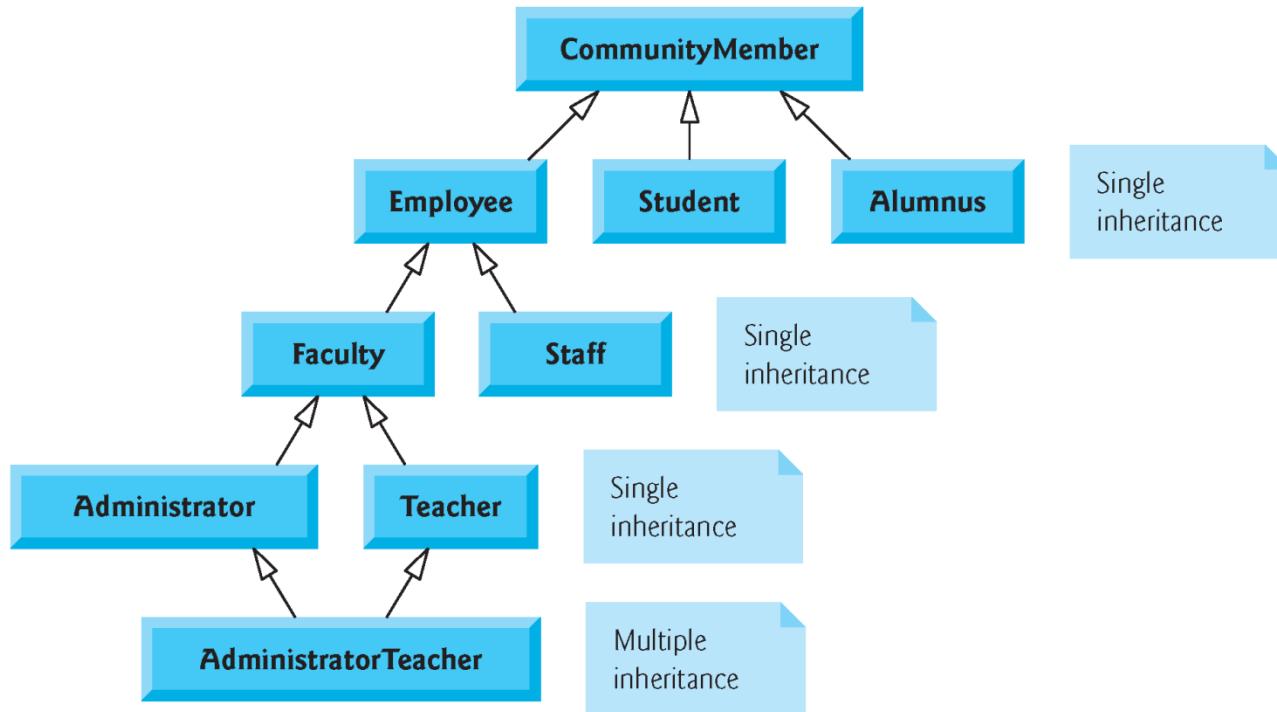
Base class	Derived classes
Student	GraduateStudent, UndergraduateStudent
Shape	Circle, Triangle, Rectangle, Sphere, Cube
Loan	CarLoan, HomeImprovementLoan, MortgageLoan
Employee	Faculty, Staff
Account	CheckingAccount, SavingsAccount

**Fig. 11.1** | Inheritance examples.

## 11.2 Base Classes and Derived Classes (cont.)

### ***CommunityMember Class Hierarchy***

- Let's develop a simple inheritance hierarchy with five levels (represented by the UML class diagram in Fig. 11.2).
- A university community has thousands of **CommunityMembers**.
- Employees are either **Faculty** or **Staff**.
- Faculty are either **Administrators** or **Teachers**.
- Some **Administrators**, however, are also **Teachers**.
- We've used *multiple inheritance* to form class **AdministratorTeacher**.



**Fig. 11.2 |** Inheritance hierarchy for university **CommunityMembers**.

## 11.2 Base Classes and Derived Classes (cont.)

- With **single inheritance**, a class is derived from *one* base class.
- With **multiple inheritance**, a derived class inherits simultaneously from *two or more* (possibly unrelated) base classes.
  - We discuss multiple inheritance in Chapter 23, Other Topics.

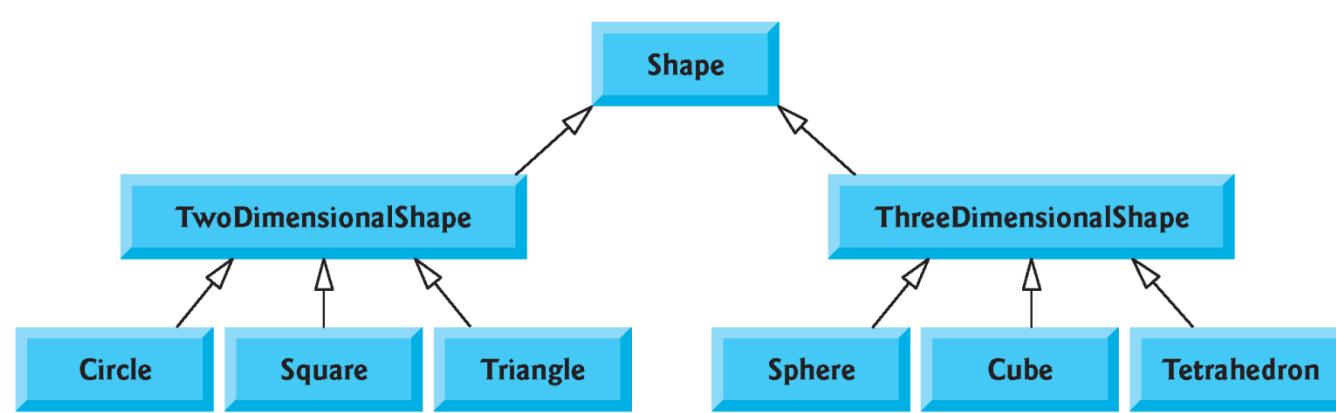
## 11.2 Base Classes and Derived Classes (cont.)

- Each arrow in the hierarchy (Fig. 11.2) 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.

## 11.2 Base Classes and Derived Classes (cont.)

### ***Shape Class Hierarchy***

- Consider the Shape inheritance hierarchy in Fig. 11.3.
- Begins with base class Shape.
- Classes TwoDimensionalShape and ThreeDimensionalShape derive from base class Shape—Shapes are either TwoDimensionalShapes or Three-DimensionalShapes.
- The third level of this hierarchy contains some more specific types of TwoDimensionalShapes and ThreeDimensionalShapes.
- As in Fig. 11.2, 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. 11.3** | Inheritance hierarchy for Shapes.

## 11.3 Relationship between Base 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.

## 11.3.1 Creating and Using a CommissionEmployee Class

- CommissionEmployee's class definition (Figs. 11.4–11.5).
- 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. 11.4: 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
8 class CommissionEmployee
9 {
10 public:
11     CommissionEmployee( const std::string &, const std::string &,
12                         const std::string &, double = 0.0, double = 0.0 );
13
14     void setFirstName( const std::string & ); // set first name
15     std::string getFirstName() const; // return first name
16
17     void setLastName( const std::string & ); // set last name
18     std::string getLastName() const; // return last name
19
20     void setSocialSecurityNumber( const std::string & ); // set SSN
21     std::string getSocialSecurityNumber() const; // return SSN
22
23     void setGrossSales( double ); // set gross sales amount
24     double getGrossSales() const; // return gross sales amount
```

---

**Fig. 11.4** | CommissionEmployee class header. (Part I of 2.)

---

```
25
26     void setCommissionRate( double ); // set commission rate (percentage)
27     double getCommissionRate() const; // return commission rate
28
29     double earnings() const; // calculate earnings
30     void print() const; // print CommissionEmployee object
31 private:
32     std::string firstName;
33     std::string lastName;
34     std::string socialSecurityNumber;
35     double grossSales; // gross weekly sales
36     double commissionRate; // commission percentage
37 } // end class CommissionEmployee
38
39 #endif
```

---

**Fig. 11.4** | CommissionEmployee class header. (Part 2 of 2.)

---

```
1 // Fig. 11.5: CommissionEmployee.cpp
2 // Class CommissionEmployee member-function definitions.
3 #include <iostream>
4 #include <stdexcept>
5 #include "CommissionEmployee.h" // CommissionEmployee class definition
6 using namespace std;
7
8 // constructor
9 CommissionEmployee::CommissionEmployee(
10     const string &first, const string &last, const string &ssn,
11     double sales, double rate )
12 {
13     firstName = first; // should validate
14     lastName = last; // should validate
15     socialSecurityNumber = ssn; // should validate
16     setGrossSales( sales ); // validate and store gross sales
17     setCommissionRate( rate ); // validate and store commission rate
18 } // end CommissionEmployee constructor
19
```

---

**Fig. 11.5** | Implementation file for CommissionEmployee class that represents an employee who is paid a percentage of gross sales. (Part I of 5.)

---

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

---

**Fig. 11.5** | Implementation file for CommissionEmployee class that represents an employee who is paid a percentage of gross sales. (Part 2 of 5.)

---

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

---

**Fig. 11.5** | Implementation file for `CommissionEmployee` class that represents an employee who is paid a percentage of gross sales. (Part 3 of 5.)

---

```
56 // set gross sales amount
57 void CommissionEmployee::setGrossSales( double sales )
58 {
59     if ( sales >= 0.0 )
60         grossSales = sales;
61     else
62         throw invalid_argument( "Gross sales must be >= 0.0" );
63 } // end function setGrossSales
64
65 // return gross sales amount
66 double CommissionEmployee::getGrossSales() const
67 {
68     return grossSales;
69 } // end function getGrossSales
70
71 // set commission rate
72 void CommissionEmployee::setCommissionRate( double rate )
73 {
74     if ( rate > 0.0 && rate < 1.0 )
75         commissionRate = rate;
76     else
77         throw invalid_argument( "Commission rate must be > 0.0 and < 1.0" );
78 } // end function setCommissionRate
```

---

**Fig. 11.5** | Implementation file for `CommissionEmployee` class that represents an employee who is paid a percentage of gross sales. (Part 4 of 5.)

---

```
79
80 // return commission rate
81 double CommissionEmployee::getCommissionRate() const
82 {
83     return commissionRate;
84 } // end function getCommissionRate
85
86 // calculate earnings
87 double CommissionEmployee::earnings() const
88 {
89     return commissionRate * grossSales;
90 } // end function earnings
91
92 // print CommissionEmployee object
93 void CommissionEmployee::print() const
94 {
95     cout << "commission employee: " << firstName << ' ' << lastName
96     << "\nsocial security number: " << socialSecurityNumber
97     << "\ngross sales: " << grossSales
98     << "\ncommission rate: " << commissionRate;
99 } // end function print
```

---

**Fig. 11.5** | Implementation file for `CommissionEmployee` class that represents an employee who is paid a percentage of gross sales. (Part 5 of 5.)

## 11.3.1 Creating and Using a `CommissionEmployee` Class (cont.)

### ***CommissionEmployee Constructor***

- 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.

## 11.3.1 Creating and Using a `CommissionEmployee` Class (cont.)

***CommissionEmployee Member Functions `earnings` and `print`***

- Member function `earnings` calculates a `CommissionEmployee`'s earnings.
- Member function `print` displays the values of a `CommissionEmployee` object's data members.

***Testing Class `CommissionEmployee`***

- Figure 11.6 tests class `CommissionEmployee`.

---

```
1 // Fig. 11.6: fig11_06.cpp
2 // CommissionEmployee class test program.
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. 11.6** | CommissionEmployee class test program. (Part I 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. 11.6** | 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. 11.6** | CommissionEmployee class test program. (Part 3 of 3.)

## 11.3.2 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. 11.7–11.8), which contains a first name, last name, social security number, gross sales amount, commission rate *and* base salary.

---

```
1 // Fig. 11.7: 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
9 class BasePlusCommissionEmployee
10 {
11 public:
12     BasePlusCommissionEmployee( const std::string &, const std::string &,
13         const std::string &, double = 0.0, double = 0.0, double = 0.0 );
14
15     void setFirstName( const std::string & ); // set first name
16     std::string getFirstName() const; // return first name
17
18     void setLastName( const std::string & ); // set last name
19     std::string getLastName() const; // return last name
20
21     void setSocialSecurityNumber( const std::string & ); // set SSN
22     std::string getSocialSecurityNumber() const; // return SSN
23
```

---

**Fig. 11.7** | BasePlusCommissionEmployee class header. (Part I 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
28     double getCommissionRate() const; // return commission rate
29
30     void setBaseSalary( double ); // set base salary
31     double getBaseSalary() const; // return base salary
32
33     double earnings() const; // calculate earnings
34     void print() const; // print BasePlusCommissionEmployee object
35 private:
36     std::string firstName;
37     std::string lastName;
38     std::string socialSecurityNumber;
39     double grossSales; // gross weekly sales
40     double commissionRate; // commission percentage
41     double baseSalary; // base salary
42 }; // end class BasePlusCommissionEmployee
43
44 #endif
```

---

**Fig. 11.7** | BasePlusCommissionEmployee class header. (Part 2 of 2.)

---

```
1 // Fig. 11.8: BasePlusCommissionEmployee.cpp
2 // Class BasePlusCommissionEmployee member-function definitions.
3 #include <iostream>
4 #include <stdexcept>
5 #include "BasePlusCommissionEmployee.h"
6 using namespace std;
7
8 // constructor
9 BasePlusCommissionEmployee::BasePlusCommissionEmployee(
10     const string &first, const string &last, const string &ssn,
11     double sales, double rate, double salary )
12 {
13     firstName = first; // should validate
14     lastName = last; // should validate
15     socialSecurityNumber = ssn; // should validate
16     setGrossSales( sales ); // validate and store gross sales
17     setCommissionRate( rate ); // validate and store commission rate
18     setBaseSalary( salary ); // validate and store base salary
19 } // end BasePlusCommissionEmployee constructor
20
```

---

**Fig. 11.8** | BasePlusCommissionEmployee class represents an employee who receives a base salary in addition to a commission. (Part 1 of 6.)

---

```
21 // set first name
22 void BasePlusCommissionEmployee::setFirstName( const string &first )
23 {
24     firstName = first; // should validate
25 } // end function setFirstName
26
27 // return first name
28 string BasePlusCommissionEmployee::getFirstName() const
29 {
30     return firstName;
31 } // end function getFirstName
32
33 // set last name
34 void BasePlusCommissionEmployee::setLastName( const string &last )
35 {
36     lastName = last; // should validate
37 } // end function setLastName
38
```

---

**Fig. 11.8** | BasePlusCommissionEmployee class represents an employee who receives a base salary in addition to a commission. (Part 2 of 6.)

---

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

---

**Fig. 11.8** | BasePlusCommissionEmployee class represents an employee who receives a base salary in addition to a commission. (Part 3 of 6.)

---

```
58 // set gross sales amount
59 void BasePlusCommissionEmployee::setGrossSales( double sales )
60 {
61     if ( sales >= 0.0 )
62         grossSales = sales;
63     else
64         throw invalid_argument( "Gross sales must be >= 0.0" );
65 } // end function setGrossSales
66
67 // return gross sales amount
68 double BasePlusCommissionEmployee::getGrossSales() const
69 {
70     return grossSales;
71 } // end function getGrossSales
72
73 // set commission rate
74 void BasePlusCommissionEmployee::setCommissionRate( double rate )
75 {
76     if ( rate > 0.0 && rate < 1.0 )
77         commissionRate = rate;
78     else
79         throw invalid_argument( "Commission rate must be > 0.0 and < 1.0" );
80 } // end function setCommissionRate
```

---

**Fig. 11.8** | BasePlusCommissionEmployee class represents an employee who receives a base salary in addition to a commission. (Part 4 of 6.)

---

```
81
82 // return commission rate
83 double BasePlusCommissionEmployee::getCommissionRate() const
84 {
85     return commissionRate;
86 } // end function getCommissionRate
87
88 // set base salary
89 void BasePlusCommissionEmployee::setBaseSalary( double salary )
90 {
91     if ( salary >= 0.0 )
92         baseSalary = salary;
93     else
94         throw invalid_argument( "Salary must be >= 0.0" );
95 } // end function setBaseSalary
96
97 // return base salary
98 double BasePlusCommissionEmployee::getBaseSalary() const
99 {
100     return baseSalary;
101 } // end function getBaseSalary
102
```

---

**Fig. 11.8** | BasePlusCommissionEmployee class represents an employee who receives a base salary in addition to a commission. (Part 5 of 6.)

---

```
103 // calculate earnings
104 double BasePlusCommissionEmployee::earnings() const
105 {
106     return baseSalary + ( commissionRate * grossSales );
107 } // end function earnings
108
109 // print BasePlusCommissionEmployee object
110 void BasePlusCommissionEmployee::print() const
111 {
112     cout << "base-salaried commission employee: " << firstName << ' '
113         << lastName << "\nsocial security number: " << socialSecurityNumber
114         << "\ngross sales: " << grossSales
115         << "\ncommission rate: " << commissionRate
116         << "\nbase salary: " << baseSalary;
117 } // end function print
```

---

**Fig. 11.8** | BasePlusCommissionEmployee class represents an employee who receives a base salary in addition to a commission. (Part 6 of 6.)

## 11.3.2 Creating a BasePlusCommissionEmployee Class Without Using Inheritance (cont.)

### ***Defining Class BasePlusCommissionEmployee***

- The `BasePlusCommissionEmployee` header (Fig. 11.7) specifies class `BasePlusCommissionEmployee`'s `public` services, which include the `BasePlusCommissionEmployee` constructor and member functions `earnings` and `print`.
- Lines 15–31 declare `public` *get* and *set* functions for the class's `private` data members `firstName`, `lastName`, `socialSecurityNumber`, `grossSales`, `commissionRate` and `baseSalary`.

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

- Note the similarity between this class and class `Commission-Employee` (Figs. 11.4–11.5)—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.

### ***Testing Class `BasePlusCommissionEmployee`***

- Figure 11.9 tests class `BasePlusCommissionEmployee`.

---

```
1 // Fig. 11.9: fig11_09.cpp
2 // BasePlusCommissionEmployee class test program.
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()
24        << "\nCommission rate is " << employee.getCommissionRate()
25        << "\nBase salary is " << employee.getBaseSalary() << endl;
```

---

**Fig. 11.9** | BasePlusCommissionEmployee class test program. (Part 1 of 3.)

---

```
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. 11.9** | 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. 11.9** | BasePlusCommissionEmployee class test program. (Part 3 of 3.)

## 11.3.2 Creating a BasePlusCommissionEmployee Class Without Using Inheritance (cont.)

*Exploring the Similarities Between Class  
BasePlusCommissionEmployee and Class  
CommissionEmployee*

- Most of the code for class `BasePlusCommissionEmployee` (Figs. 11.7–11.8) is similar, if not identical, to the code for class `CommissionEmployee` (Figs. 11.4–11.5).
- 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.

## 11.3.2 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 `getBaseSalary`.
- 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`.

## 11.3.2 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 11.1**

---

Copying and pasting code from one class to another can spread many physical copies of the same code and can spread errors throughout a system, creating a code-maintenance nightmare. 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 11.2

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.

### 11.3.3 Creating a CommissionEmployee– BasePlusCommissionEmployee Inheritance Hierarchy

- Now we create and test a new `BasePlusCommissionEmployee` class (Figs. 11.10–11.11) that derives from class `CommissionEmployee` (Figs. 11.4–11.5).
- 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` (Fig. 11.10, line 22).
- The colon (:) in line 10 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.

### 11.3.3 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. 11.10: 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
10 class BasePlusCommissionEmployee : public CommissionEmployee
11 {
12 public:
13     BasePlusCommissionEmployee( const std::string &, const std::string &,
14         const std::string &, double = 0.0, double = 0.0, double = 0.0 );
15
16     void setBaseSalary( double ); // set base salary
17     double getBaseSalary() const; // return base salary
18
```

---

**Fig. 11.10** | BasePlusCommissionEmployee class definition indicating inheritance relationship with class CommissionEmployee. (Part I of 2.)

---

```
19     double earnings() const; // calculate earnings
20     void print() const; // print BasePlusCommissionEmployee object
21 private:
22     double baseSalary; // base salary
23 }; // end class BasePlusCommissionEmployee
24
25 #endif
```

---

**Fig. 11.10** | BasePlusCommissionEmployee class definition indicating inheritance relationship with class CommissionEmployee. (Part 2 of 2.)

---

```
1 // Fig. 11.11: BasePlusCommissionEmployee.cpp
2 // Class BasePlusCommissionEmployee member-function definitions.
3 #include <iostream>
4 #include <stdexcept>
5 #include "BasePlusCommissionEmployee.h"
6 using namespace std;
7
8 // constructor
9 BasePlusCommissionEmployee::BasePlusCommissionEmployee(
10     const string &first, const string &last, const string &ssn,
11     double sales, double rate, double salary )
12     // explicitly call base-class constructor
13     : CommissionEmployee( first, last, ssn, sales, rate )
14 {
15     setBaseSalary( salary ); // validate and store base salary
16 } // end BasePlusCommissionEmployee constructor
17
```

---

**Fig. 11.11** | BasePlusCommissionEmployee implementation file: private base-class data cannot be accessed from derived class. (Part I of 4.)

---

```
18 // set base salary
19 void BasePlusCommissionEmployee::setBaseSalary( double salary )
20 {
21     if ( salary >= 0.0 )
22         baseSalary = salary;
23     else
24         throw invalid_argument( "Salary must be >= 0.0" );
25 } // end function setBaseSalary
26
27 // return base salary
28 double BasePlusCommissionEmployee::getBaseSalary() const
29 {
30     return baseSalary;
31 } // end function getBaseSalary
32
33 // calculate earnings
34 double BasePlusCommissionEmployee::earnings() const
35 {
36     // derived class cannot access the base class's private data
37     return baseSalary + ( commissionRate * grossSales );
38 } // end function earnings
39
```

---

**Fig. 11.11** | BasePlusCommissionEmployee implementation file: private base-class data cannot be accessed from derived class. (Part 2 of 4.)

---

```
40 // print BasePlusCommissionEmployee object
41 void BasePlusCommissionEmployee::print() const
42 {
43     // derived class cannot access the base class's private data
44     cout << "base-salaried commission employee: " << firstName << " "
45         << lastName << "\nsocial security number: " << socialSecurityNumber
46         << "\ngross sales: " << grossSales
47         << "\ncommission rate: " << commissionRate
48         << "\nbase salary: " << baseSalary;
49 } // end function print
```

---

**Fig. 11.11** | BasePlusCommissionEmployee implementation file: private base-class data cannot be accessed from derived class. (Part 3 of 4.)

---

*Compilation Errors from the LLVM Compiler in Xcode 4.5*

```
BasePlusCommissionEmployee.cpp:37:26:  
    'commissionRate' is a private member of 'CommissionEmployee'  
BasePlusCommissionEmployee.cpp:37:43:  
    'grossSales' is a private member of 'CommissionEmployee'  
BasePlusCommissionEmployee.cpp:44:53:  
    'firstName' is a private member of 'CommissionEmployee'  
BasePlusCommissionEmployee.cpp:45:10:  
    'lastName' is a private member of 'CommissionEmployee'  
BasePlusCommissionEmployee.cpp:45:54:  
    'socialSecurityNumber' is a private member of 'CommissionEmployee'  
BasePlusCommissionEmployee.cpp:46:31:  
    'grossSales' is a private member of 'CommissionEmployee'  
BasePlusCommissionEmployee.cpp:47:35:  
    'commissionRate' is a private member of 'CommissionEmployee'
```

**Fig. 11.11** | BasePlusCommissionEmployee implementation file: private base-class data cannot be accessed from derived class. (Part 4 of 4.)

### 11.3.3 Creating a CommissionEmployee– BasePlusCommissionEmployee Inheritance Hierarchy (cont.)

- Figure 11.11 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 11.1

---

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 11.1

---

In a derived-class constructor, invoking base-class constructors and initializing member objects 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.

## 11.3.3 Creating a CommissionEmployee– BasePlusCommissionEmployee Inheritance Hierarchy (cont.)

### ***Compilation Errors from Accessing Base-Class private Members***

- The compiler generates errors for line 37 of Fig. 11.11 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. 11.11 to indicate erroneous code.
- The compiler issues additional errors in lines 44–47 of `BasePlusCommissionEmployee`'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.*

## 11.3.3 Creating a CommissionEmployee– BasePlusCommissionEmployee Inheritance Hierarchy (cont.)

### ***Preventing the Errors in BasePlusCommissionEmployee***

- We purposely included the erroneous code in Fig. 11.11 to emphasize that a derived class's member functions cannot access its base class's **private** data.
- The errors in BasePlusCommissionEmployee could have been prevented by using the *get* member functions inherited from class CommissionEmployee.

### 11.3.3 Creating a CommissionEmployee– BasePlusCommissionEmployee Inheritance Hierarchy (cont.)

- For example, line 37 could have invoked `getCommissionRate` and `getGrossSales` to access `CommissionEmployee`'s private data members `commissionRate` and `grossSales`, respectively.
- Similarly, lines 44–47 could have used appropriate *get* member functions to retrieve the values of the base class's data members.

## 11.3.3 Creating a CommissionEmployee– BasePlusCommissionEmployee Inheritance Hierarchy (cont.)

### ***Including the Base-Class Header in the Derived-Class Header with #include***

- We `#include` the base class's header in the derived class's header (line 8 of Fig. 11.10).
- 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 `#includes` 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.

## 11.3.3 Creating a CommissionEmployee– BasePlusCommissionEmployee Inheritance Hierarchy (cont.)

### ***Linking Process in an Inheritance Hierarchy***

- In Section 3.7, 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.

## 11.3.4 CommissionEmployee– BasePlusCommissionEmployee Inheritance Hierarchy Using **protected** Data

- In this section, we introduce the access specifier **protected**.
- 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 within the body of that base class, by members and **friends** of that base class, and by members and **friends** of any classes derived from that base class.

## 11.3.4 CommissionEmployee– BasePlusCommissionEmployee Inheritance Hierarchy Using **protected** Data (cont.)

### ***Defining Base Class CommissionEmployee with protected Data***

- Class **CommissionEmployee** (Fig. 11.12) now declares data members **firstName**, **lastName**, **socialSecurityNumber**, **grossSales** and **commissionRate** as **protected** (lines 31–36) rather than **private**.
- The member-function implementations are identical to those in Fig. 11.5.

---

```
1 // Fig. 11.12: 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
8 class CommissionEmployee
9 {
10 public:
11     CommissionEmployee( const std::string &, const std::string &,
12                         const std::string &, double = 0.0, double = 0.0 );
13
14     void setFirstName( const std::string & ); // set first name
15     std::string getFirstName() const; // return first name
16
17     void setLastName( const std::string & ); // set last name
18     std::string getLastName() const; // return last name
19
20     void setSocialSecurityNumber( const std::string & ); // set SSN
21     std::string getSocialSecurityNumber() const; // return SSN
22
```

---

**Fig. 11.12** | CommissionEmployee class definition that declares protected data to allow access by derived classes. (Part I of 2.)

---

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

---

**Fig. 11.12** | CommissionEmployee class definition that declares protected data to allow access by derived classes. (Part 2 of 2.)

## 11.3.4 CommissionEmployee– BasePlusCommissionEmployee Inheritance Hierarchy Using **protected** Data (cont.)

- BasePlusCommissionEmployee inherits from class CommissionEmployee in Fig. 11.12.
- 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. 11.11 (lines 34–38 and 41–49, respectively).
- Objects of a derived class also can access **protected** members in any of that derived class's *indirect* base classes.

## 11.3.4 CommissionEmployee– BasePlusCommissionEmployee Inheritance Hierarchy Using **protected** Data (cont.)

### ***Testing the Modified BasePlusCommissionEmployee Class***

- To test the updated class hierarchy, we reused the test program from Fig. 11.9.
- As shown in Fig. 11.13, the output is identical to that of Fig. 11.9.
- The code for class **BasePlusCommissionEmployee**, which is 74 lines, is considerably shorter than the code for the noninherited version of the class, which is 161 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.

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. 11.13** | protected base-class data can be accessed from derived class.

## 11.3.4 CommissionEmployee– BasePlusCommissionEmployee Inheritance Hierarchy Using **protected** Data (cont.)

### ***Notes on Using protected Data***

- 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.



### Software Engineering Observation 11.3

In most cases, it's better to use **private** data members to encourage proper software engineering, and leave code optimization issues to the compiler. Your code will be easier to maintain, modify and debug.

## 11.3.4 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 11.4

---

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



## Software Engineering Observation 11.5

---

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.

## 11.3.5 CommissionEmployee– BasePlusCommissionEmployee Inheritance Hierarchy Using private Data

- We now reexamine our hierarchy once more, this time using the best software engineering practices.
- Class `CommissionEmployee` now declares data members `firstName`, `lastName`, `socialSecurityNumber`, `grossSales` and `commissionRate` as `private` (as shown previously in lines 31–36 of Fig. 11.4).

## 11.3.5 CommissionEmployee– BasePlusCommissionEmployee Inheritance Hierarchy Using **private** Data (cont.)

### *Changes to Class CommissionEmployee's Member Function Definitions*

- In the `CommissionEmployee` constructor implementation (Fig. 11.14, lines 9–16), we use member initializers (line 12) to set the values of members `firstName`, `lastName` and `socialSecurityNumber`.
- We show how derived-class `BasePlusCommissionEmployee` (Fig. 11.15) can invoke non-private base-class member functions (`setFirstName`, `getFirstName`, `setLastName`, `getLastName`, `setSocialSecurityNumber` and `getSocialSecurityNumber`) to manipulate these data members.

---

```
1 // Fig. 11.14: CommissionEmployee.cpp
2 // Class CommissionEmployee member-function definitions.
3 #include <iostream>
4 #include <stdexcept>
5 #include "CommissionEmployee.h" // CommissionEmployee class definition
6 using namespace std;
7
8 // constructor
9 CommissionEmployee::CommissionEmployee(
10     const string &first, const string &last, const string &ssn,
11     double sales, double rate )
12     : firstName( first ), lastName( last ), socialSecurityNumber( ssn )
13 {
14     setGrossSales( sales ); // validate and store gross sales
15     setCommissionRate( rate ); // validate and store commission rate
16 } // end CommissionEmployee constructor
17
```

---

**Fig. 11.14** | CommissionEmployee class implementation file:  
CommissionEmployee class uses member functions to manipulate its private  
data. (Part I of 6.)

---

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

---

**Fig. 11.14** | CommissionEmployee class implementation file:  
CommissionEmployee class uses member functions to manipulate its private  
data. (Part 2 of 6.)

---

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

---

**Fig. 11.14** | CommissionEmployee class implementation file:  
CommissionEmployee class uses member functions to manipulate its private  
data. (Part 3 of 6.)

---

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

---

**Fig. 11.14** | CommissionEmployee class implementation file:  
CommissionEmployee class uses member functions to manipulate its private  
data. (Part 4 of 6.)

---

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

---

**Fig. 11.14** | CommissionEmployee class implementation file:  
CommissionEmployee class uses member functions to manipulate its private  
data. (Part 5 of 6.)

---

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

---

**Fig. 11.14** | CommissionEmployee class implementation file:  
CommissionEmployee class uses member functions to manipulate its private  
data. (Part 6 of 6.)



## Performance Tip 11.2

---

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."

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

### *Changes to Class BasePlusCommissionEmployee's Member Function Definitions*

- Class BasePlusCommissionEmployee has several changes to its member-function implementations (Fig. 11.15) that distinguish it from the previous version of the class (Figs. 11.10–11.11).
- Member functions `earnings` (Fig. 11.15, lines 34–37) and `print` (lines 40–48) each invoke `getBaseSalary` to obtain the base salary value.

---

```
1 // Fig. 11.15: BasePlusCommissionEmployee.cpp
2 // Class BasePlusCommissionEmployee member-function definitions.
3 #include <iostream>
4 #include <stdexcept>
5 #include "BasePlusCommissionEmployee.h"
6 using namespace std;
7
8 // constructor
9 BasePlusCommissionEmployee::BasePlusCommissionEmployee(
10     const string &first, const string &last, const string &ssn,
11     double sales, double rate, double salary )
12 // explicitly call base-class constructor
13 : CommissionEmployee( first, last, ssn, sales, rate )
14 {
15     setBaseSalary( salary ); // validate and store base salary
16 } // end BasePlusCommissionEmployee constructor
17
```

---

**Fig. 11.15** | BasePlusCommissionEmployee class that inherits from class CommissionEmployee but cannot directly access the class's private data. (Part I of 3.)

---

```
18 // set base salary
19 void BasePlusCommissionEmployee::setBaseSalary( double salary )
20 {
21     if ( salary >= 0.0 )
22         baseSalary = salary;
23     else
24         throw invalid_argument( "Salary must be >= 0.0" );
25 } // end function setBaseSalary
26
27 // return base salary
28 double BasePlusCommissionEmployee::getBaseSalary() const
29 {
30     return baseSalary;
31 } // end function getBaseSalary
32
33 // calculate earnings
34 double BasePlusCommissionEmployee::earnings() const
35 {
36     return getBaseSalary() + CommissionEmployee::earnings();
37 } // end function earnings
38
```

---

**Fig. 11.15** | BasePlusCommissionEmployee class that inherits from class CommissionEmployee but cannot directly access the class's private data. (Part 2 of 3.)

---

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

---

**Fig. 11.15** | BasePlusCommissionEmployee class that inherits from class CommissionEmployee but cannot directly access the class's private data. (Part 3 of 3.)

## 11.3.5 CommissionEmployee– BasePlusCommissionEmployee Inheritance Hierarchy Using **private** Data (cont.)

### ***BasePlusCommissionEmployee Member Function earnings***

- Class `BasePlusCommissionEmployee`'s `earnings` function (Fig. 11.15, lines 34–37) redefines class `CommissionEmployee`'s `earnings` member function (Fig. 11.14, lines 85–88) 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.



## Common Programming Error 11.2

---

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.

## 11.3.5 CommissionEmployee– BasePlusCommissionEmployee Inheritance Hierarchy Using **private** Data (cont.)

*BasePlusCommissionEmployee Member Function print*

- BasePlusCommissionEmployee's print function (Fig. 11.15, lines 40–48) redefines class CommissionEmployee's print function (Fig. 11.14, lines 91–98) to output the appropriate base-salaried commission employee information.
- 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.

## 11.4 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 most 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 11.6

---

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.

## 11.4 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 11.7

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 body executes, then the constructors for the derived class's member objects execute, then the derived class's constructor body executes. Destructors for derived-class objects are called in the reverse of the order in which their corresponding constructors are called.

## 11.4 Constructors and Destructors in Derived Classes (cont.)

- Base-class constructors, destructors and overloaded assignment operators (Chapter 10) are *not* inherited by derived classes.
- Derived-class constructors, destructors and overloaded assignment operators, however, can call base-class versions.

## 11.4 Constructors and Destructors in Derived Classes (cont.)

### ***C++11: Inheriting Base Class Constructors***

- Sometimes a derived class's constructors simply mimic the base class's constructors.
- A frequently requested convenience feature for C++11 was the ability to *inherit* a base class's constructors.
- You can now do this by explicitly including a using declaration of the form

```
using BaseClass::BaseClass;
```

- *anywhere* in the derived-class definition.
- In the preceding declaration, **BaseClass** is the base class's name.

## 11.4 Constructors and Destructors in Derived Classes (cont.)

- When you inherit constructors:
  - By default, each inherited constructor has the *same* access level (public, protected or private) as its corresponding base-class constructor.
  - The default, copy and move constructors are *not* inherited.
  - If a constructor is *deleted* in the base class by placing = delete in its prototype, the corresponding constructor in the derived class is *also* deleted.
  - If the derived class does not explicitly define constructors, the compiler generates a default constructor in the derived class—*even* if it inherits other constructors from its base class.
  - If a constructor that you *explicitly* define in a derived class has the *same* parameter list as a base-class constructor, then the base-class constructor is *not* inherited.
  - A base-class constructor's default arguments are *not* inherited. Instead, the compiler generates overloaded constructors in the derived class.

## 11.5 public, protected and private Inheritance

- When deriving a class from a base class, the base class may be inherited through **public**, **protected** or **private** inheritance.
- Use of **protected** and **private** inheritance is rare.
- Figure 11.16 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	<p><b>public</b> in derived class.</p> <p>Can be accessed directly by member functions, <b>friend</b> functions and nonmember functions.</p>	<p><b>protected</b> in derived class.</p> <p>Can be accessed directly by member functions and <b>friend</b> functions.</p>	<p><b>private</b> in derived class.</p> <p>Can be accessed directly by member functions and <b>friend</b> functions.</p>
protected	<p><b>protected</b> in derived class.</p> <p>Can be accessed directly by member functions and <b>friend</b> functions.</p>	<p><b>protected</b> in derived class.</p> <p>Can be accessed directly by member functions and <b>friend</b> functions.</p>	<p><b>private</b> in derived class.</p> <p>Can be accessed directly by member functions and <b>friend</b> functions.</p>
private	<p>Hidden in derived class.</p> <p>Can be accessed by member functions and <b>friend</b> functions through <b>public</b> or <b>protected</b> member functions of the base class.</p>	<p>Hidden in derived class.</p> <p>Can be accessed by member functions and <b>friend</b> functions through <b>public</b> or <b>protected</b> member functions of the base class.</p>	<p>Hidden in derived class.</p> <p>Can be accessed by member functions and <b>friend</b> functions through <b>public</b> or <b>protected</b> member functions of the base class.</p>

## 11.6 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. 11.16.
- 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.

## 11.6 Software Engineering with Inheritance (cont.)

- 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.
- We can customize the new class to meet our needs by redefining base-class members and by including additional 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).

## 11.7 Software Engineering with Inheritance (cont.)

- 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.



### **Software Engineering Observation 11.8**

---

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.



### **Software Engineering Observation 11.9**

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