12

Object-Oriented Programming: Inheritance

Say not you know another entirely, till you have divided an inheritance with him.

— Johann Kasper Lavater

This method is to define as the number of a class the class of all classes similar to the given class.

— Bertrand Russell

Good as it is to inherit a library, it is better to collect one.

— Augustine Birrell

Save base authority from others' books.

— William Shakespeare

OBJECTIVES

In this chapter you will learn:

- To create classes by inheriting from existing classes.
- How inheritance 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 differences between `public`, `protected` and `private` inheritance.
- The use of inheritance to customize existing software.

Outline

12.1 Introduction

- **Inheritance**
  - **Software reusability**
  - **Create new class from existing class**
    - **Absorb existing class’s data and behaviors**
    - **Enhance with new capabilities**
  - **Derived class inherits from base class**
    - **Derived class**
      - **More specialized group of objects**
      - **Behaviors inherited from base class**
        - **Can customize**
      - **Additional behaviors**

12.1 Introduction (Cont.)

- Class hierarchy
  - Direct base class
    - Inherited explicitly (one level up hierarchy)
  - Indirect base class
    - Inherited two or more levels up hierarchy
  - Single inheritance
    - Inherits from one base class
  - Multiple inheritance
    - Inherits from multiple base classes
      - Base classes possibly unrelated
    - More details in chapter 24

12.1 Introduction (Cont.)

- Three types of inheritance
  - `public`
    - Every object of derived class is also an object of base class
      - Base-class objects are not objects of derived classes
      - Example: All cars are vehicles, but not all vehicles are cars
    - Can access non-`private` members of base class
      - To access `private` base-class members
        - Derived class must use inherited non-`private` member functions
  - `private`
    - Alternative to composition
    - Chapter 21
  - `protected`
    - Rarely used

12.1 Introduction (Cont.)

- **Abstraction**
  - **Focus on commonalities among objects in system**
- **"is-a" vs. "has-a"**
  - **"is-a"**
    - **Inheritance**
    - **Derived class object can be treated as base class object**
    - **Example: Car *is a* vehicle**
      - **Vehicle properties/behaviors also apply to a car**
  - **"has-a"**
    - **Composition**
    - **Object contains one or more objects of other classes as members**
    - **Example: Car *has a* steering wheel**

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

- **Base classes and derived classes**
  - **Object of one class "is an" object of another class**
    - **Example: Rectangle is quadrilateral**
      - **Class `Rectangle` inherits from class `Quadrilateral`**
        - `Quadrilateral` **is the base class**
        - `Rectangle` **is the derived class**
  - **Base class typically represents larger set of objects than derived classes**
    - **Example:**
      - **Base class:** `Vehicle`
        - **Includes cars, trucks, boats, bicycles, etc.**
      - **Derived class:** `Car`
        - **Smaller, more-specific subset of vehicles**

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.)

- **Inheritance hierarchy**
  - **Inheritance relationships: tree-like hierarchy structure**
  - **Each class becomes**
    - **Base class**
      - **Supplies data/behaviors to other classes**

    OR

    - **Derived class**
      - **Inherits data/behaviors from other classes**

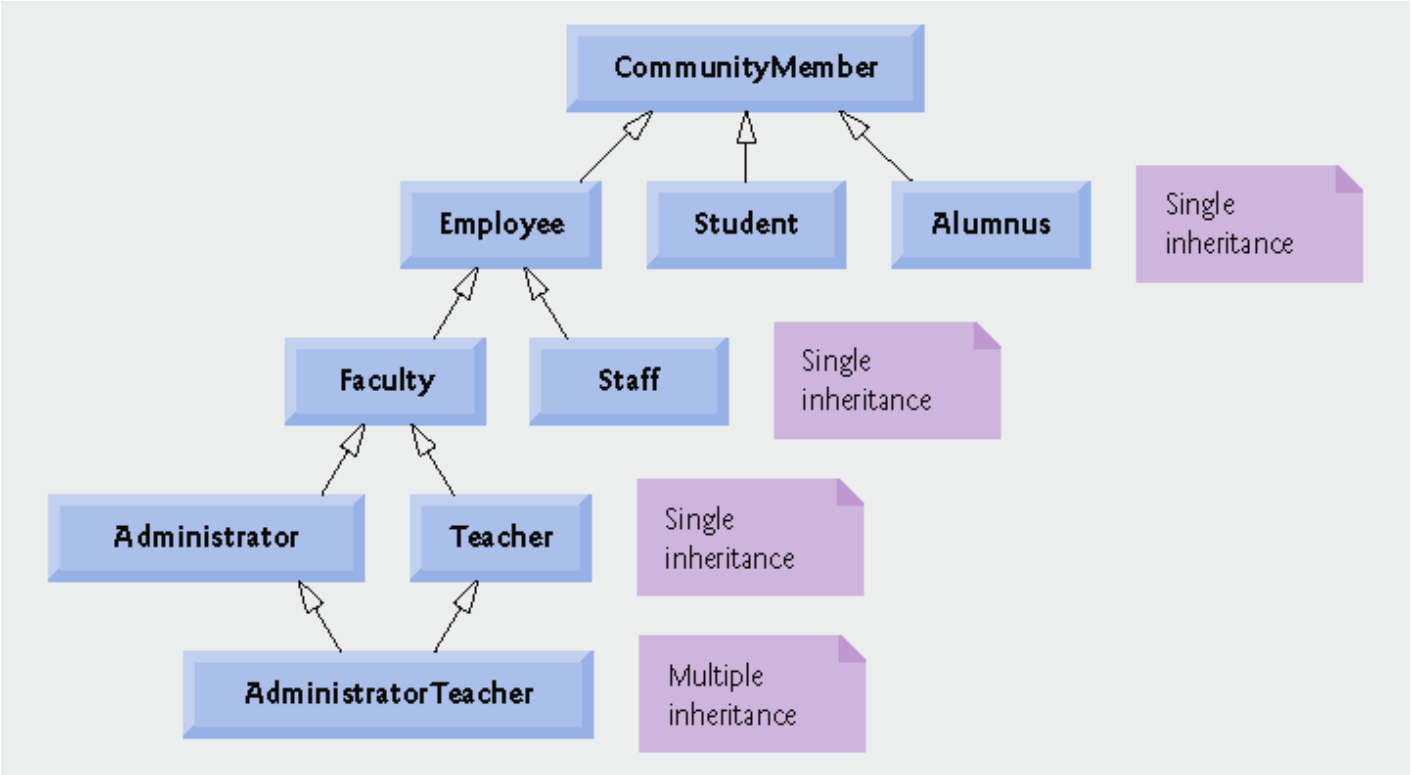

Fig. 12.2 | Inheritance hierarchy for university `CommunityMembers`.

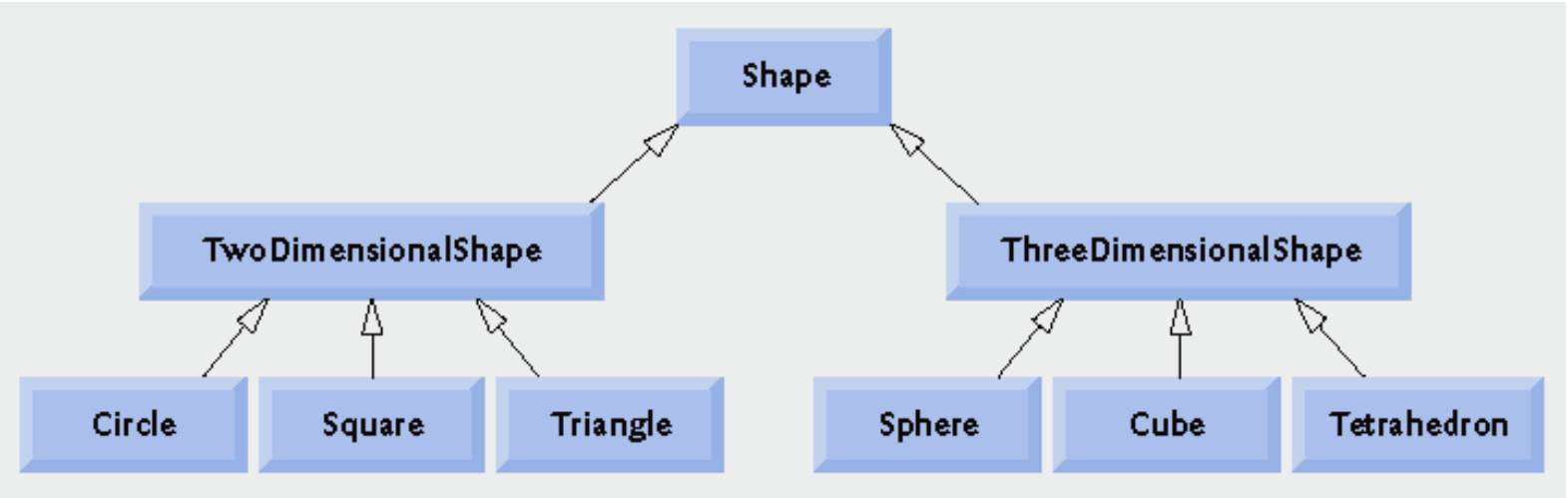

Fig. 12.3 | Inheritance hierarchy for Shapes.

12.2 Base Classes and Derived Classes (Cont.)

- `public` inheritance
  - Specify with:

    ```
    Class TwoDimensionalShape : public Shape
    ```

    - Class `TwoDimensionalShape` inherits from class `Shape`
  - Base class `private` members
    - Not accessible directly
    - Still inherited
      - Manipulated through inherited `public` member functions
  - Base class `public` and `protected` members
    - Inherited with original member access
  - `friend` functions
    - Not inherited

12.3 `protected` Members

- `protected` access
  - Intermediate level of protection between `public` and `private`
  - `protected` members are accessible to
    - Base class members
    - Base class `friend`s
    - Derived class members
    - Derived class `friend`s
- Derived-class members
  - Refer to `public` and `protected` members of base class
    - Simply use member names
  - Redefined base class members can be accessed by using base-class name and binary scope resolution operator (`::`)

12.4 Relationship between Base Classes and Derived Classes

- **Base class and derived class relationship**
  - **Example: `CommissionEmployee/ BasePlusCommissionEmployee` inheritance hierarchy**
    - `CommissionEmployee`
      - **First name, last name, SSN, commission rate, gross sale amount**
    - `BasePlusCommissionEmployee`
      - **First name, last name, SSN, commission rate, gross sale amount**
      - **And also: base salary**

12.4.1 Creating and Using a `CommissionEmployee` Class

- Class `CommissionEmployee`
  - `CommissionEmployee` header file
    - Fig. 12.4
    - Specify public services
      - Constructor
      - *get* and *set* functions
      - Member functions `earnings` and `print`
  - `CommissionEmployee` source code file
    - Fig. 12.5
    - Specify member-function definitions

Outline

Commission Employee.h

(1 of 2)

```
// Fig. 12.4: CommissionEmployee.h
// CommissionEmployee class definition represents a commission employee.
#ifndef COMMISSION_H
#define COMMISSION_H

#include <string> // C++ standard string class
using std::string;

class CommissionEmployee
{
public:
   CommissionEmployee( const string &, const string &, const string &,
      double = 0.0, double = 0.0 );

   void setFirstName( const string & ); // set first name
   string getFirstName() const; // return first name

   void setLastName( const string & ); // set last name
   string getLastName() const; // return last name

   void setSocialSecurityNumber( const string & ); // set SSN
   string getSocialSecurityNumber() const; // return SSN

   void setGrossSales( double ); // set gross sales amount
   double getGrossSales() const; // return gross sales amount

   void setCommissionRate( double ); // set commission rate (percentage)
   double getCommissionRate() const; // return commission rate
```

Class **CommissionEmployee** constructor

```
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
```

Declare **private** data members

Outline

Commission Employee.h

(2 of 2)

Outline

CommissionEmployee.cpp

(1 of 4)

```
// Fig. 12.5: CommissionEmployee.cpp
// Class CommissionEmployee member-function definitions.
#include <iostream>
using std::cout;

#include "CommissionEmployee.h" // CommissionEmployee class definition

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

// set first name
void CommissionEmployee::setFirstName( const string &first )
{
   firstName = first; // should validate
} // end function setFirstName

// return first name
string CommissionEmployee::getFirstName() const
{
   return firstName;
} // end function getFirstName
```

Initialize data members

Outline

CommissionEmployee.cpp

(2 of 4)

```
// set last name
void CommissionEmployee::setLastName( const string &last )
{
   lastName = last; // should validate
} // end function setLastName

// return last name
string CommissionEmployee::getLastName() const
{
   return lastName;
} // end function getLastName

// set social security number
void CommissionEmployee::setSocialSecurityNumber( const string &ssn )
{
   socialSecurityNumber = ssn; // should validate
} // end function setSocialSecurityNumber

// return social security number
string CommissionEmployee::getSocialSecurityNumber() const
{
   return socialSecurityNumber;
} // end function getSocialSecurityNumber

// set gross sales amount
void CommissionEmployee::setGrossSales( double sales )
{
   grossSales = ( sales < 0.0 ) ? 0.0 : sales;
} // end function setGrossSales
```

Function **setGrossSales** validates gross sales amount

Outline

```
// return gross sales amount
double CommissionEmployee::getGrossSales() const
{
   return grossSales;
} // end function getGrossSales

// set commission rate
void CommissionEmployee::setCommissionRate( double rate )
{
   commissionRate = ( rate > 0.0 && rate < 1.0 ) ? rate : 0.0;
} // end function setCommissionRate

// return commission rate
double CommissionEmployee::getCommissionRate() const
{
   return commissionRate;
} // end function getCommissionRate
```

Function **`setCommissionRate`** validates commission rate

on
cpp

(3 of 4)

Outline

CommissionEmployee.cpp

(4 of 4)

```
// calculate earnings
double CommissionEmployee::earnings() const
{
   return commissionRate * grossSales;
} // end function earnings

// print CommissionEmployee object
void CommissionEmployee::print() const
{
   cout << "commission employee: " << firstName << ' ' << l
      << "\nsocial security number: " << socialSecurityNumber
      << "\ngross sales: " << grossSales
      << "\ncommission rate: " << commissionRate;
} // end function print
```

Function **earnings** calculates earnings

Function **print** displays **CommissionEmployee** object

Outline

fig12_06.cpp

(1 of 2)

```
// Fig. 12.6: fig12_06.cpp
// Testing class CommissionEmployee.
#include <iostream>
using std::cout;
using std::endl;
using std::fixed;

#include <iomanip>
using std::setprecision;

#include "CommissionEmployee.h" // CommissionEmployee class definition

int main()
{
   // instantiate a CommissionEmployee object
   CommissionEmployee employee(
      "Sue", "Jones", "222-22-2222", 10000, .06 );

   // set floating-point output formatting
   cout << fixed << setprecision( 2 );

   // get commission employee data
   cout << "Employee information obtained by get functions: \n"
      << "\nFirst name is " << employee.getFirstName()
      << "\nLast name is " << employee.getLastName()
      << "\nSocial security number is "
      << employee.getSocialSecurityNumber()
      << "\nGross sales is " << employee.getGrossSales()
      << "\nCommission rate is " << employee.getCommissionRate() << endl;
```

Instantiate **CommissionEmployee** object

Use **CommissionEmployee**'s *get* functions to retrieve the object's instance variable values

Outline

```
30
31    employee.setGrossSales( 8000 ); // set gross sales
32    employee.setCommissionRate( .1 ); // set commission rate
33
34    cout << "\nUpdated employee information output
35       << endl;
36    employee.print(); // display the new employee information
37
38    // display the employee's earnings
39    cout << "\n\nEmployee's earnings: $" << employee.earnings() << endl;
40
41    return 0;
42 } // end main
```

Use **CommissionEmployee**'s *set* functions to change the object's instance variable values

Call object's **print** function to display employee information

Call object's **earnings** function to calculate earnings

(2 of 2)

```
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
```

12.4.2 Creating a `BasePlusCommissionEmployee` Class Without Using Inheritance

- Class `BasePlusCommissionEmployee`
  - Much of the code is similar to `CommissionEmployee`
    - `private` data members
    - `public` methods
    - constructor
  - Additions
    - `private` data member `baseSalary`
    - Methods `setBaseSalary` and `getBaseSalary`

Outline

BasePlus Commission Employee.h

(1 of 2)

```
// Fig. 12.7: BasePlusCommissionEmployee.h
// BasePlusCommissionEmployee class definition represents an employee
// that receives a base salary in addition to commission.
#ifndef BASEPLUS_H
#define BASEPLUS_H

#include <string> // C++ standard string class
using std::string;

class BasePlusCommissionEmployee
{
public:
   BasePlusCommissionEmployee( const string &, const string &,
      const string &, double = 0.0, double = 0.0, double = 0.0 );

   void setFirstName( const string & ); // set first name
   string getFirstName() const; // return first name

   void setLastName( const string & ); // set last name
   string getLastName() const; // return last name

   void setSocialSecurityNumber( const string & ); // set SSN
   string getSocialSecurityNumber() const; // return SSN

   void setGrossSales( double ); // set gross sales amount
   double getGrossSales() const; // return gross sales amount

   void setCommissionRate( double ); // set commission rate
   double getCommissionRate() const; // return commission rate
```

Constructor takes one more argument, which specifies the base salary

```
   void setBaseSalary( double ); // set base salary
   double getBaseSalary() const; // return base salary

   double earnings() const; // calculate earnings
   void print() const; // print BasePlusCommissionEmployee object
private:
   string firstName;
   string lastName;
   string socialSecurityNumber;
   double grossSales; // gross weekly sales
   double commissionRate; // commission percentage
   double baseSalary; // base salary
}; // end class BasePlusCommissionEmployee

#endif
```

Define *get* and *set* functions for data member **baseSalary**

Add data member **baseSalary**

BasePlus Commission Employee.h

(2 of 2)

Outline

BasePlus Commission Employee.cpp

(1 of 4)

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

// BasePlusCommissionEmployee class definition
#include "BasePlusCommissionEmployee.h"

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

// set first name
void BasePlusCommissionEmployee::setFirstName( const string &first )
{
   firstName = first; // should validate
} // end function setFirstName
```

Constructor takes one more argument, which specifies the base salary

Use function **`setBaseSalary`** to validate data

Outline

BasePlus Commission Employee.cpp

(2 of 4)

```
// return first name
string BasePlusCommissionEmployee::getFirstName() const
{
   return firstName;
} // end function getFirstName

// set last name
void BasePlusCommissionEmployee::setLastName( const string &last )
{
   lastName = last; // should validate
} // end function setLastName

// return last name
string BasePlusCommissionEmployee::getLastName() const
{
   return lastName;
} // end function getLastName

// set social security number
void BasePlusCommissionEmployee::setSocialSecurityNumber(
   const string &ssn )
{
   socialSecurityNumber = ssn; // should validate
} // end function setSocialSecurityNumber

```

Outline

```
// return social security number
string BasePlusCommissionEmployee::getSocialSecurityNumber() const
{
   return socialSecurityNumber;
} // end function getSocialSecurityNumber

// set gross sales amount
void BasePlusCommissionEmployee::setGrossSales( double sales )
{
   grossSales = ( sales < 0.0 ) ? 0.0 : sales;
} // end function setGrossSales

// return gross sales amount
double BasePlusCommissionEmployee::getGrossSales() const
{
   return grossSales;
} // end function getGrossSales

// set commission rate
void BasePlusCommissionEmployee::setCommissionRate( double rate )
{
   commissionRate = ( rate > 0.0 && rate < 1.0 ) ? rate : 0.0;
} // end function setCommissionRate

// return commission rate
double BasePlusCommissionEmployee::getCommissionRate() const
{
   return commissionRate;
} // end function getCommissionRate

```

BasePlus Commission Employee.cpp

(3 of 4)

Outline

```
// set base salary
void BasePlusCommissionEmployee::setBaseSalary( double salary )
{
   baseSalary = ( salary < 0.0 ) ? 0.0 : salary;
} // end function setBaseSalary

// return base salary
double BasePlusCommissionEmployee::getBaseSalary() const
{
   return baseSalary;
} // end function getBaseSalary

// calculate earnings
double BasePlusCommissionEmployee::earnings() const
{
   return baseSalary + ( commissionRate * grossSales );
} // end function earnings

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

Function **setBaseSalary** validates data and sets instance variable **baseSalary**

Commission Employee.cpp

Function **getBaseSalary** returns the value of instance variable **baseSalary**

Update function **earnings** to calculate the earnings of a base-salaried commission employee

Update function **print** to display base salary

Outline

fig12_09.cpp

(1 of 3)

```
// Fig. 12.9: fig12_09.cpp
// Testing class BasePlusCommissionEmployee.
#include <iostream>
using std::cout;
using std::endl;
using std::fixed;

#include <iomanip>
using std::setprecision;

// BasePlusCommissionEmployee class definition
#include "BasePlusCommissionEmployee.h"

int main()
{
   // instantiate BasePlusCommissionEmployee object
   BasePlusCommissionEmployee
      employee( "Bob", "Lewis", "333-33-3333", 5000, .04, 300 );

   // set floating-point output formatting
   cout << fixed << setprecision( 2 );

```

Instantiate **BasePlusCommissionEmployee** object

Outline

```
   // get commission employee data
   cout << "Employee information obtained by get functions: \n"
      << "\nFirst name is " << employee.getFirstName()
      << "\nLast name is " << employee.getLastName()
      << "\nSocial security number is "
      << employee.getSocialSecurityNumber()
      << "\nGross sales is " << employee.getGrossSales()
      << "\nCommission rate is " << employee.getCommissionRate()
      << "\nBase salary is " << employee.getBaseSalary() << endl;

   employee.setBaseSalary( 1000 ); // set base salary

   cout << "\nUpdated employee information output by
      << endl;
   employee.print(); // display the new employee information

   // display the employee's earnings
   cout << "\n\nEmployee's earnings: $" << employee.earnings() << endl;

   return 0;
} // end main
```

Use **BasePlusCommissionEmployee**'s *get* functions to retrieve the object's instance variable values (2 of 3)

Use **BasePlusCommissionEmployee**'s **setBaseSalary** function to set base salary

Call object's **print** function to display employee information

Call object's **earnings** function to calculate employee's earnings

```
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
```


fig12_09.cpp

(3 of 3)

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, software developers 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

- Class `BasePlusCommissionEmployee`
  - Derived from class `CommissionEmployee`
    - Is a `CommissionEmployee`
    - Inherits all `public` members
  - Constructor is not inherited
    - Use base-class initializer syntax to initialize base-class data member
  - Has data member `baseSalary`

Outline

BasePlus
ssion
ee.h

(1 of 1)

```
// Fig. 12.10: BasePlusCommissionEmployee.h
// BasePlusCommissionEmployee class derived from class
// CommissionEmployee.
#ifndef BASEPLUS_H
#define BASEPLUS_H

#include <string> // C++ standard string class
using std::string;

#include "CommissionEmployee.h" // CommissionEmployee class declaration

class BasePlusCommissionEmployee : public CommissionEmployee
{
public:
   BasePlusCommissionEmployee( const string &, const string &
      const string &, double = 0.0, double = 0.0

   void setBaseSalary( double ); // set base salary
   double getBaseSalary() const; // return base salary

   double earnings() const; // calculate earnings
   void print() const; // print BasePlusCommissionEmployee object
private:
   double baseSalary; // base salary
}; // end class BasePlusCommissionEmployee

#endif
```

Include the base-class header file in the derived-class header file

Class `BasePlusCommissionEmployee` derives **`public`**ly from class `CommissionEmployee`

Outline

BasePlus Commission Employee.cpp

(1 of 4)

```
// Fig. 12.11: BasePlusCommissionEmployee.cpp
// Class BasePlusCommissionEmployee member-function definitions.
#include <iostream>
using std::cout;

// BasePlusCommissionEmployee class definition
#include "BasePlusCommissionEmployee.h"

// constructor
BasePlusCommissionEmployee::BasePlusCommissionEmployee(
   const string &first, const string &last, const string &ssn,
   double sales, double rate, double salary )
   // explicitly call base-class constructor
   : CommissionEmployee( first, last, ssn, sales, rate )
{
   setBaseSalary( salary ); // validate and store base salary
} // end BasePlusCommissionEmployee constructor

// set base salary
void BasePlusCommissionEmployee::setBaseSalary( double salary )
{
   baseSalary = ( salary < 0.0 ) ? 0.0 : salary;
} // end function setBaseSalary

// return base salary
double BasePlusCommissionEmployee::getBaseSalary() const
{
   return baseSalary;
} // end function getBaseSalary
```

Initialize base class data member by calling the base -class constructor using base-class initializer syntax

Outline

BasePlus
Commission

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

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

Compiler generates errors because base class's data member **commissionRate** and **grossSales** are **private**

Compiler generates errors because the base class's data members **firstName**, **lastName**, **socialSecurityNumber**, **grossSales** and **commissionRate** are **private**

```
C:\cpphtp5_examples\ch12\Fig12_10_11\BasePlusCommission-Employee.cpp(35) :
   error C2248: 'CommissionEmployee::commissionRate' :
   cannot access private member declared in class 'CommissionEmployee'
      C:\cpphtp5_examples\ch12\Fig12_10_11\CommissionEmployee.h(37) :
         see declaration of 'CommissionEmployee::commissionRate'
      C:\cpphtp5e_examples\ch12\Fig12_10_11\CommissionEmployee.h(10) :
         see declaration of 'CommissionEmployee'

C:\cpphtp5_examples\ch12\Fig12_10_11\BasePlusCommission-Employee.cpp(35) :
   error C2248: 'CommissionEmployee::grossSales' :
   cannot access private member declared in class 'CommissionEmployee'
      C:\cpphtp5_examples\ch12\Fig12_10_11\CommissionEmployee.h(36) :
         see declaration of 'CommissionEmployee::grossSales'
      C:\cpphtp5_examples\ch12\Fig12_10_11\CommissionEmployee.h(10) :
         see declaration of 'CommissionEmployee'

C:\cpphtp5_examples\ch12\Fig12_10_11\BasePlusCommission-Employee.cpp(42) :
   error C2248: 'CommissionEmployee::firstName' :
   cannot access private member declared in class 'CommissionEmployee'
      C:\cpphtp5_examples\ch12\Fig12_10_11\CommissionEmployee.h(33) :
         see declaration of 'CommissionEmployee::firstName'
      C:\cpphtp5_examples\ch12\Fig12_10_11\CommissionEmployee.h(10) :
         see declaration of 'CommissionEmployee'
```

Outline

BasePlus Commission Employee.cpp

(3 of 4)

```
C:\cpphtp5_examples\ch12\Fig12_10_11\BasePlusCommission-Employee.cpp(43) :
  error C2248: 'CommissionEmployee::lastName' :
  cannot access private member declared in class 'CommissionEmployee'
      C:\cpphtp5_examples\ch12\Fig12_10_11\CommissionEmployee.h(34) :
         see declaration of 'CommissionEmployee::lastName'
      C:\cpphtp5_examples\ch12\Fig12_10_11\CommissionEmployee.h(10) :
         see declaration of 'CommissionEmployee'

C:\cpphtp5_examples\ch12\Fig12_10_11\BasePlusCommission-Employee.cpp(43) :
  error C2248: 'CommissionEmployee::socialSecurity-Number' :
  cannot access private member declared in class 'CommissionEmployee'
      C:\cpphtp5_examples\ch12\Fig12_10_11\CommissionEmployee.h(35) :
         see declaration of 'CommissionEmployee::socialSecurityNumber'
      C:\cpphtp5_examples\ch12\Fig12_10_11\CommissionEmployee.h(10) :
         see declaration of 'CommissionEmployee'

C:\cpphtp5_examples\ch12\Fig12_10_11\BasePlusCommission-Employee.cpp(44) :
  error C2248: 'CommissionEmployee::grossSales' :
  cannot access private member declared in class 'CommissionEmployee'
      C:\cpphtp5_examples\ch12\Fig12_10_11\CommissionEmployee.h(36) :
         see declaration of 'CommissionEmployee::grossSales'
      C:\cpphtp5_examples\ch12\Fig12_10_11\CommissionEmployee.h(10) :
         see declaration of 'CommissionEmployee'

C:\cpphtp5_examples\ch12\Fig12_10_11\BasePlusCommission-Employee.cpp(45) :
  error C2248: 'CommissionEmployee::commissionRate' :
  cannot access private member declared in class 'CommissionEmployee'
      C:\cpphtp5_examples\ch12\Fig12_10_11\CommissionEmployee.h(37) :
         see declaration of 'CommissionEmployee::commissionRate'
      C:\cpphtp5_examples\ch12\Fig12_10_11\CommissionEmployee.h(10) :
         see declaration of 'CommissionEmployee'
```

Outline

BasePlus Commission Employee.cpp

(4 of 4)

Common Programming Error 12.1

A compilation error occurs if a derived-class constructor calls one of its base-class constructors with arguments that are inconsistent with the number and types of parameters specified in one of the base-class constructor definitions.

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.)

- **Including the base class header file**
  - **Base class header file must be included in derived class header file for three reasons, the compiler must**
    - **Know that base class exists**
    - **Know size of inherited data members**
    - **Ensure that inherited class members are used properly**

12.4.4 `CommissionEmployee-BasePlusCommissionEmployee` Inheritance Hierarchy Using `protected` Data

- Use `protected` data
  - Enable class `BasePlusCommissionEmployee` to directly access base class data members
  - Base class's `protected` members are inherited by all derived classes of that base class

Good Programming Practice 12.1

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

Outline

Commission Employee.h

(1 of 2)

```
// Fig. 12.12: CommissionEmployee.h
// CommissionEmployee class definition with protected data.
#ifndef COMMISSION_H
#define COMMISSION_H

#include <string> // C++ standard string class
using std::string;

class CommissionEmployee
{
public:
   CommissionEmployee( const string &, const string &, const string &,
      double = 0.0, double = 0.0 );

   void setFirstName( const string & ); // set first name
   string getFirstName() const; // return first name

   void setLastName( const string & ); // set last name
   string getLastName() const; // return last name

   void setSocialSecurityNumber( const string & ); // set SSN
   string getSocialSecurityNumber() const; // return SSN

```

Outline

CommissionEmployee.h

(2 of 2)

```
   void setGrossSales( double ); // set gross sales amount
   double getGrossSales() const; // return gross sales amount

   void setCommissionRate( double ); // set commission rate
   double getCommissionRate() const; // return commission rate

   double earnings() const; // calculate earnings
   void print() const; // print CommissionEmployee object
protected:
   string firstName;
   string lastName;
   string socialSecurityNumber;
   double grossSales; // gross weekly sales
   double commissionRate; // commission percentage
}; // end class CommissionEmployee

#endif
```

Declare **protected** data

Outline

CommissionEmployee.cpp

(1 of 4)

```
// Fig. 12.13: CommissionEmployee.cpp
// Class CommissionEmployee member-function definitions.
#include <iostream>
using std::cout;

#include "CommissionEmployee.h" // CommissionEmployee class definition

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

// set first name
void CommissionEmployee::setFirstName( const string &first )
{
   firstName = first; // should validate
} // end function setFirstName

// return first name
string CommissionEmployee::getFirstName() const
{
   return firstName;
} // end function getFirstName
```

Outline

CommissionEmployee.cpp

(2 of 4)

```
// set last name
void CommissionEmployee::setLastName( const string &last )
{
   lastName = last; // should validate
} // end function setLastName

// return last name
string CommissionEmployee::getLastName() const
{
   return lastName;
} // end function getLastName

// set social security number
void CommissionEmployee::setSocialSecurityNumber( const string &ssn )
{
   socialSecurityNumber = ssn; // should validate
} // end function setSocialSecurityNumber

// return social security number
string CommissionEmployee::getSocialSecurityNumber() const
{
   return socialSecurityNumber;
} // end function getSocialSecurityNumber

// set gross sales amount
void CommissionEmployee::setGrossSales( double sales )
{
   grossSales = ( sales < 0.0 ) ? 0.0 : sales;
} // end function setGrossSales
```

Outline

```
// return gross sales amount
double CommissionEmployee::getGrossSales() const
{
   return grossSales;
} // end function getGrossSales

// set commission rate
void CommissionEmployee::setCommissionRate( double rate )
{
   commissionRate = ( rate > 0.0 && rate < 1.0 ) ? rate : 0.0;
} // end function setCommissionRate

// return commission rate
double CommissionEmployee::getCommissionRate() const
{
   return commissionRate;
} // end function getCommissionRate

// calculate earnings
double CommissionEmployee::earnings() const
{
   return commissionRate * grossSales;
} // end function earnings
```

CommissionEmployee.cpp

(3 of 4)

Outline

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

Commission Employee.cpp

(4 of 4)

Outline

BasePlus Commission Employee.h

(1 of 1)

```
// Fig. 12.14: BasePlusCommissionEmployee.h
// BasePlusCommissionEmployee class derived from class
// CommissionEmployee.
#ifndef BASEPLUS_H
#define BASEPLUS_H

#include <string> // C++ standard string class
using std::string;

#include "CommissionEmployee.h" // CommissionEmployee class declaration

class BasePlusCommissionEmployee : public CommissionEmployee
{
public:
   BasePlusCommissionEmployee( const string &, const string &,
      const string &, double = 0.0, double = 0.0, double = 0.0 );

   void setBaseSalary( double ); // set base salary
   double getBaseSalary() const; // return base salary

   double earnings() const; // calculate earnings
   void print() const; // print BasePlusCommissionEmployee object
private:
   double baseSalary; // base salary
}; // end class BasePlusCommissionEmployee

#endif
```

BasePlusCommissionEmployee still inherits **public**ly from **CommissionEmployee**

Outline

BasePlus Commission Employee.cpp

```cpp
// Fig. 12.15: BasePlusCommissionEmployee.cpp
// Class BasePlusCommissionEmployee member-function definitions.
#include <iostream>
using std::cout;

// BasePlusCommissionEmployee class definition
#include "BasePlusCommissionEmployee.h"

// constructor
BasePlusCommissionEmployee::BasePlusCommissionEmployee(
   const string &first, const string &last, const string &ssn,
   double sales, double rate, double salary )
   // explicitly call base-class constructor
   : CommissionEmployee( first, last, ssn, sales, rate )
{
   setBaseSalary( salary ); // validate and store base salary
} // end BasePlusCommissionEmployee constructor

// set base salary
void BasePlusCommissionEmployee::setBaseSalary( double salary )
{
   baseSalary = ( salary < 0.0 ) ? 0.0 : salary;
} // end function setBaseSalary

// return base salary
double BasePlusCommissionEmployee::getBaseSalary() const
{
   return baseSalary;
} // end function getBaseSalary
```

Call base-class constructor using base-class initializer syntax

Outline

```
// calculate earnings
double BasePlusCommissionEmployee::earnings() const
{
   // can access protected data of base class
   return baseSalary + ( commissionRate * grossSales );
} // end function earnings

// print BasePlusCommissionEmployee object
void BasePlusCommissionEmployee::print() const
{
   // can access protected data of base class
   cout << "base-salaried commission employee: " << firstName << ' '
      << lastName << "\nsocial security number: " << socialSecurityNumber
      << "\ngross sales: " << grossSales
      << "\ncommission rate: " << commissionRate
      << "\nbase salary: " << baseSalary;
} // end function print
```

BasePlus Commission Employee.cpp

Directly access base class's **protected** data

Outline

Fig12_16.cpp

(1 of 3)

```
// Fig. 12.16: fig12_16.cpp
// Testing class BasePlusCommissionEmployee.
#include <iostream>
using std::cout;
using std::endl;
using std::fixed;

#include <iomanip>
using std::setprecision;

// BasePlusCommissionEmployee class definition
#include "BasePlusCommissionEmployee.h"

int main()
{
   // instantiate BasePlusCommissionEmployee object
   BasePlusCommissionEmployee
      employee( "Bob", "Lewis", "333-33-3333", 5000, .04, 300 );

   // set floating-point output formatting
   cout << fixed << setprecision( 2 );

```

Outline

```
   // get commission employee data
   cout << "Employee information obtained by get functions: \n"
      << "\nFirst name is " << employee.getFirstName()
      << "\nLast name is " << employee.getLastName()
      << "\nSocial security number is "
      << employee.getSocialSecurityNumber()
      << "\nGross sales is " << employee.getGrossSales()
      << "\nCommission rate is " << employee.getCommissionRate()
      << "\nBase salary is " << employee.getBaseSalary() << endl;

   employee.setBaseSalary( 1000 ); // set base salary

   cout << "\nUpdated employee information output by print function: \n"
      << endl;
   employee.print(); // display the new employee information

   // display the employee's earnings
   cout << "\n\nEmployee's earnings: $" << employee.earnings() << endl;

   return 0;
} // end main
```

Fig12_16.cpp

(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
```


Fig12_16.cpp

(3 of 3)

12.4.4 `CommissionEmployee-BasePlusCommissionEmployee` Inheritance Hierarchy Using `protected` Data (Cont.)

- Using `protected` data members
  - Advantages
    - Derived class can modify values directly
      - Avoid *set*/*get* method call overhead
        - Slight increase in performance
  - Disadvantages
    - No validity checking
      - Derived class can assign illegal value
    - Implementation dependent
      - Derived class functions more likely dependent on base class implementation
      - Base class implementation changes may result in derived class modifications
        - Fragile (brittle) software

Software Engineering Observation 12.5

It is 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 `friends`), not to other clients.

Software Engineering Observation 12.6

Declaring base-class data members `private` (as opposed to declaring them `protected`) enables programmers 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

- **Reexamine hierarchy**
  - **Use the best software engineering practice**
    - **Declare data members as `private`**
    - **Provide `public` *get* and *set* functions**
    - **Use *get* method to obtain values of data members**

Outline

Commission Employee.h

(1 of 2)

```
// Fig. 12.17: CommissionEmployee.h
// CommissionEmployee class definition with good software engineering.
#ifndef COMMISSION_H
#define COMMISSION_H

#include <string> // C++ standard string class
using std::string;

class CommissionEmployee
{
public:
   CommissionEmployee( const string &, const string &, const string &,
      double = 0.0, double = 0.0 );

   void setFirstName( const string & ); // set first name
   string getFirstName() const; // return first name

   void setLastName( const string & ); // set last name
   string getLastName() const; // return last name

   void setSocialSecurityNumber( const string & ); // set SSN
   string getSocialSecurityNumber() const; // return SSN

   void setGrossSales( double ); // set gross sales amount
   double getGrossSales() const; // return gross sales amount

   void setCommissionRate( double ); // set commission rate
   double getCommissionRate() const; // return commission rate
```

Outline

```
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
```

Declare **private** data

Commission Employee.h

(2 of 2)

Outline

Commission Employee.cpp

(1 of 4)

```
// Fig. 12.18: CommissionEmployee.cpp
// Class CommissionEmployee member-function definitions.
#include <iostream>
using std::cout;

#include "CommissionEmployee.h" // CommissionEmployee class definition

// constructor
CommissionEmployee::CommissionEmployee(
   const string &first, const string &last, const string &ssn,
   double sales, double rate )
   : firstName( first ), lastName( last ), socialSecurityNumber( ssn )
{
   setGrossSales( sales ); // validate and store gros
   setCommissionRate( rate ); // validate and store c
} // end CommissionEmployee constructor

// set first name
void CommissionEmployee::setFirstName( const string &first )
{
   firstName = first; // should validate
} // end function setFirstName

// return first name
string CommissionEmployee::getFirstName() const
{
   return firstName;
} // end function getFirstName
```

Use member initializers to set the values of members **`firstName`**, **`lastname`** and **`socialSecurityNumber`**

Outline

CommissionEmployee.cpp

(2 of 4)

```

// set last name
void CommissionEmployee::setLastName( const string &last )
{
   lastName = last; // should validate
} // end function setLastName

// return last name
string CommissionEmployee::getLastName() const
{
   return lastName;
} // end function getLastName

// set social security number
void CommissionEmployee::setSocialSecurityNumber( const string &ssn )
{
   socialSecurityNumber = ssn; // should validate
} // end function setSocialSecurityNumber

// return social security number
string CommissionEmployee::getSocialSecurityNumber() const
{
   return socialSecurityNumber;
} // end function getSocialSecurityNumber

// set gross sales amount
void CommissionEmployee::setGrossSales( double sales )
{
   grossSales = ( sales < 0.0 ) ? 0.0 : sales;
} // end function setGrossSales
```

Outline

CommissionEmployee.cpp

(3 of 4)

```
// return gross sales amount
double CommissionEmployee::getGrossSales() const
{
   return grossSales;
} // end function getGrossSales

// set commission rate
void CommissionEmployee::setCommissionRate( double rate )
{
   commissionRate = ( rate > 0.0 && rate < 1.0 ) ? rate : 0.0;
} // end function setCommissionRate

// return commission rate
double CommissionEmployee::getCommissionRate() const
{
   return commissionRate;
} // end function getCommissionRate

// calculate earnings
double CommissionEmployee::earnings() const
{
   return getCommissionRate() * getGrossSales();
} // end function earnings
```

Use *get* functions to obtain the values of data members

Outline

```
// print CommissionEmployee object
void CommissionEmployee::print() const
{
   cout << "commission employee: "
      << getFirstName() << ' ' << getLastName()
      << "\nsocial security number: " << getSocialSecurityNumber()
      << "\ngross sales: " << getGrossSales()
      << "\ncommission rate: " << getCommissionRate();
} // end function print
```

Use *get* functions to obtain the values of data members

Commission Employee.cpp

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). As a result, programmers should write code that adheres to proper software engineering principles, and leave optimization issues to the compiler. A good rule is, "Do not second-guess the compiler."

Outline

BasePlus Commission Employee.h

(1 of 1)

```
// Fig. 12.19: BasePlusCommissionEmployee.h
// BasePlusCommissionEmployee class derived from class
// CommissionEmployee.
#ifndef BASEPLUS_H
#define BASEPLUS_H

#include <string> // C++ standard string class
using std::string;

#include "CommissionEmployee.h" // CommissionEmployee class declaration

class BasePlusCommissionEmployee : public CommissionEmployee
{
public:
   BasePlusCommissionEmployee( const string &, const string &,
      const string &, double = 0.0, double = 0.0, double = 0.0 );

   void setBaseSalary( double ); // set base salary
   double getBaseSalary() const; // return base salary

   double earnings() const; // calculate earnings
   void print() const; // print BasePlusCommissionEmployee object
private:
   double baseSalary; // base salary
}; // end class BasePlusCommissionEmployee

#endif
```

Outline

BasePlus Commission Employee.cpp

(1 of 2)

```
// Fig. 12.20: BasePlusCommissionEmployee.cpp
// Class BasePlusCommissionEmployee member-function definitions.
#include <iostream>
using std::cout;

// BasePlusCommissionEmployee class definition
#include "BasePlusCommissionEmployee.h"

// constructor
BasePlusCommissionEmployee::BasePlusCommissionEmployee(
   const string &first, const string &last, const string &ssn,
   double sales, double rate, double salary )
   // explicitly call base-class constructor
   : CommissionEmployee( first, last, ssn, sales, rate )
{
   setBaseSalary( salary ); // validate and store base salary
} // end BasePlusCommissionEmployee constructor

// set base salary
void BasePlusCommissionEmployee::setBaseSalary( double salary )
{
   baseSalary = ( salary < 0.0 ) ? 0.0 : salary;
} // end function setBaseSalary

// return base salary
double BasePlusCommissionEmployee::getBaseSalary() const
{
   return baseSalary;
} // end function getBaseSalary
```

Outline

BasePlus

```
// calculate earnings
double BasePlusCommissionEmployee::earnings() const
{
   return getBaseSalary() + CommissionEmployee::earnings();
} // end function earnings

// print BasePlusCommissionEmployee object
void BasePlusCommissionEmployee::print() const
{
   cout << "base-salaried ";

   // invoke CommissionEmployee's print function
   CommissionEmployee::print();

   cout << "\nbase salary: " << getBaseSalary()
} // end function print
```

Invoke base class's **earnings** function

Invoke base class's **print** function

(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.

Common Programming Error 12.3

Including a base-class member function with a different signature in the derived class hides the base-class version of the function. Attempts to call the base-class version through the `public` interface of a derived-class object result in compilation errors.

Outline

fig12_21.cpp

(1 of 3)

```
// Fig. 12.21: fig12_21.cpp
// Testing class BasePlusCommissionEmployee.
#include <iostream>
using std::cout;
using std::endl;
using std::fixed;

#include <iomanip>
using std::setprecision;

// BasePlusCommissionEmployee class definition
#include "BasePlusCommissionEmployee.h"

```

Outline

fig12_21.cpp

```
int main()
{
   // instantiate BasePlusCommissionEmployee object
   BasePlusCommissionEmployee
      employee( "Bob", "Lewis", "333-33-3333", 5000, .04, 300 );

   // set floating-point output formatting
   cout << fixed << setprecision( 2 );

   // get commission employee data
   cout << "Employee information obtained by get functions: \n"
      << "\nFirst name is " << employee.getFirstName()
      << "\nLast name is " << employee.getLastName()
      << "\nSocial security number is "
      << employee.getSocialSecurityNumber()
      << "\nGross sales is " << employee.getGrossSales()
      << "\nCommission rate is " << employee.getCommissionRate()
      << "\nBase salary is " << employee.getBaseSalary() << endl;

   employee.setBaseSalary( 1000 ); // set base salary

   cout << "\nUpdated employee information
      << endl;
   employee.print(); // display the new em

   // display the employee's earnings
   cout << "\n\nEmployee's earnings: $" << employee.earnings() << endl;

   return 0;
} // end main
```

Create **BasePlusCommissionEmployee** object

Use inherited *get* methods to access base class **private** members

Use **BasePlusCommissionEmployee** *get* method to access **private** member

Use **BasePlusCommissionEmployee** *set* method to modify **private** data member **baseSalary**

```
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
```


fig12_21.cpp

(3 of 3)

12.5 Constructors and Destructors in Derived Classes

- **Instantiating derived-class object**
  - **Chain of constructor calls**
    - **Derived-class constructor invokes base class constructor**
      - **Implicitly or explicitly**
    - **Base of inheritance hierarchy**
      - **Last constructor called in chain**
      - **First constructor body to finish executing**
      - **Example: `CommissionEmployee`/ `BasePlusCommissionEmployee` hierarchy**
        - `CommissionEmployee` **constructor called last**
        - `CommissionEmployee` **constructor body finishes execution first**
    - **Initializing data members**
      - **Each base-class constructor initializes its data members that are inherited by derived class**

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 the hierarchy contains more than two levels.

12.5 Constructors and Destructors in Derived Classes (Cont.)

- **Destroying derived-class object**
  - **Chain of destructor calls**
    - **Reverse order of constructor chain**
    - **Destructor of derived-class called first**
    - **Destructor of next base class up hierarchy next**
      - **Continue up hierarchy until final base reached**
        - **After final base-class destructor, object removed from memory**
- **Base-class constructors, destructors, assignment operators**
  - **Not inherited by derived classes**

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 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.

Outline

Commission Employee.h

(1 of 2)

```
// Fig. 12.22: CommissionEmployee.h
// CommissionEmployee class definition represents a commission employee.
#ifndef COMMISSION_H
#define COMMISSION_H

#include <string> // C++ standard string class
using std::string;

class CommissionEmployee
{
public:
   CommissionEmployee( const string &, const string &, const string &,
      double = 0.0, double = 0.0 );
   ~CommissionEmployee(); // destructor

   void setFirstName( const string & ); // set first name
   string getFirstName() const; // return first name

   void setLastName( const string & ); // set last name
   string getLastName() const; // return last name

   void setSocialSecurityNumber( const string & ); // set SSN
   string getSocialSecurityNumber() const; // return SSN

   void setGrossSales( double ); // set gross sales amount
   double getGrossSales() const; // return gross sales amount

   void setCommissionRate( double ); // set commission rate
   double getCommissionRate() const; // return commission rate
```

CommissionEmployee destructor

```
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
```

Outline

Commission Employee.h

(2 of 2)

Outline

Commission Employee.cpp

(1 of 4)

```
// Fig. 12.23: CommissionEmployee.cpp
// Class CommissionEmployee member-function definitions.
#include <iostream>
using std::cout;
using std::endl;

#include "CommissionEmployee.h" // CommissionEmployee class definition

// constructor
CommissionEmployee::CommissionEmployee(
   const string &first, const string &last, const string &ssn,
   double sales, double rate )
   : firstName( first ), lastName( last ), socialSecurityNumber( ssn )
{
   setGrossSales( sales ); // validate and store gross sales
   setCommissionRate( rate ); // validate and store commission rate

   cout << "CommissionEmployee constructor: " << endl;
   print();
   cout << "\n\n";
} // end CommissionEmployee constructor

// destructor
CommissionEmployee::~CommissionEmployee()
{
   cout << "CommissionEmployee destructor: " << endl;
   print();
   cout << "\n\n";
} // end CommissionEmployee destructor
```

Constructor and destructor output messages to demonstrate function call order

Outline

```
// set first name
void CommissionEmployee::setFirstName( const string &first )
{
   firstName = first; // should validate
} // end function setFirstName

// return first name
string CommissionEmployee::getFirstName() const
{
   return firstName;
} // end function getFirstName

// set last name
void CommissionEmployee::setLastName( const string &last )
{
   lastName = last; // should validate
} // end function setLastName

// return last name
string CommissionEmployee::getLastName() const
{
   return lastName;
} // end function getLastName

// set social security number
void CommissionEmployee::setSocialSecurityNumber( const string &ssn )
{
   socialSecurityNumber = ssn; // should validate
} // end function setSocialSecurityNumber
```

CommissionEmployee.cpp

(2 of 4)

Outline

```
// return social security number
string CommissionEmployee::getSocialSecurityNumber() const
{
   return socialSecurityNumber;
} // end function getSocialSecurityNumber

// set gross sales amount
void CommissionEmployee::setGrossSales( double sales )
{
   grossSales = ( sales < 0.0 ) ? 0.0 : sales;
} // end function setGrossSales

// return gross sales amount
double CommissionEmployee::getGrossSales() const
{
   return grossSales;
} // end function getGrossSales

// set commission rate
void CommissionEmployee::setCommissionRate( double rate )
{
   commissionRate = ( rate > 0.0 && rate < 1.0 ) ? rate : 0.0;
} // end function setCommissionRate

// return commission rate
double CommissionEmployee::getCommissionRate() const
{
   return commissionRate;
} // end function getCommissionRate
```

CommissionEmployee.cpp

(3 of 4)

```
// calculate earnings
double CommissionEmployee::earnings() const
{
   return getCommissionRate() * getGrossSales();
} // end function earnings

// print CommissionEmployee object
void CommissionEmployee::print() const
{
   cout << "commission employee: "
      << getFirstName() << ' ' << getLastName()
      << "\nsocial security number: " << getSocialSecurityNumber()
      << "\ngross sales: " << getGrossSales()
      << "\ncommission rate: " << getCommissionRate();
} // end function print
```

Outline

BasePlus Commission Employee.h

(1 of 1)

```
// Fig. 12.24: BasePlusCommissionEmployee.h
// BasePlusCommissionEmployee class derived from class
// CommissionEmployee.
#ifndef BASEPLUS_H
#define BASEPLUS_H

#include <string> // C++ standard string class
using std::string;

#include "CommissionEmployee.h" // CommissionEmployee class declaration

class BasePlusCommissionEmployee : public CommissionEmployee
{
public:
   BasePlusCommissionEmployee( const string &, const string &,
      const string &, double = 0.0, double = 0.0, double = 0.0 );
   ~BasePlusCommissionEmployee(); // destructor

   void setBaseSalary( double ); // set base salary
   double getBaseSalary() const; // return base salary

   double earnings() const; // calculate earnings
   void print() const; // print BasePlusCommissionEmployee object
private:
   double baseSalary; // base salary
}; // end class BasePlusCommissionEmployee

#endif
```

BasePlusCommissionEmployee destructor

Outline

BasePlus Commission Employee.cpp

(1 of 2)

```
// Fig. 12.25: BasePlusCommissionEmployee.cpp
// Class BasePlusCommissionEmployee member-function definitions.
#include <iostream>
using std::cout;
using std::endl;

// BasePlusCommissionEmployee class definition
#include "BasePlusCommissionEmployee.h"

// constructor
BasePlusCommissionEmployee::BasePlusCommissionEmployee(
   const string &first, const string &last, const string &ssn,
   double sales, double rate, double salary )
   // explicitly call base-class constructor
   : CommissionEmployee( first, last, ssn, sales, rate )
{
   setBaseSalary( salary ); // validate and store base salary

   cout << "BasePlusCommissionEmployee constructor: " << endl;
   print();
   cout << "\n\n";
} // end BasePlusCommissionEmployee constructor

// destructor
BasePlusCommissionEmployee::~BasePlusCommissionEmployee()
{
   cout << "BasePlusCommissionEmployee destructor: " << endl;
   print();
   cout << "\n\n";
} // end BasePlusCommissionEmployee destructor
```

Constructor and destructor output messages to demonstrate function call order

Outline

```
// set base salary
void BasePlusCommissionEmployee::setBaseSalary( double salary )
{
   baseSalary = ( salary < 0.0 ) ? 0.0 : salary;
} // end function setBaseSalary

// return base salary
double BasePlusCommissionEmployee::getBaseSalary() const
{
   return baseSalary;
} // end function getBaseSalary

// calculate earnings
double BasePlusCommissionEmployee::earnings() const
{
   return getBaseSalary() + CommissionEmployee::earnings();
} // end function earnings

// print BasePlusCommissionEmployee object
void BasePlusCommissionEmployee::print() const
{
   cout << "base-salaried ";

   // invoke CommissionEmployee's print function
   CommissionEmployee::print();

   cout << "\nbase salary: " << getBaseSalary();
} // end function print
```

BasePlus Commission Employee.cpp

(2 of 2)

Outline

fig12_26.cpp

(1 of 4)

```
// Fig. 12.26: fig12_26.cpp
// Display order in which base-class and derived-class constructors
// and destructors are called.
#include <iostream>
using std::cout;
using std::endl;
using std::fixed;

#include <iomanip>
using std::setprecision;

// BasePlusCommissionEmployee class definition
#include "BasePlusCommissionEmployee.h"
```

Outline

```
int main()
{
   // set floating-point output formatting
   cout << fixed << setprecision( 2 );

   { // begin new scope
      CommissionEmployee employee1(
         "Bob", "Lewis", "333-33-3333", 5000, .04 );
   } // end scope

   cout << endl;
   BasePlusCommissionEmployee
      employee2( "Lisa", "Jones", "555-55-5555", 2000, .06, 800 );

   cout << endl;
   BasePlusCommissionEmployee
      employee3( "Mark", "Sands", "888-88-8888", 8000, .15, 2000 );
   cout << endl;
   return 0;
} // end main
```

CommissionEmployee object goes in and out of scope immediately

(2 of 4)

Instantiate two **BasePlusCommissionEmployee** objects to demonstrate order of derived-class and base -class constructor/destructor function calls

Outline

```
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 called for object in block; destructor called immediately as execution leaves scope

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

Base-class **CommissionEmployee** constructor executes first when instantiating derived-class **BasePlusCommissionEmployee** object

```
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
```

Derived-class **BasePlusCommissionEmployee** constructor body executes after base-class **CommissionEmployee**'s constructor finishes execution

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

Base-class **CommissionEmployee** constructor executes first when instantiating derived-class **BasePlusCommissionEmployee** object

(continued at top of next slide...)

(... continued from bottom of previous slide)

Outline

(4 of 4)

```
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

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
```

Derived-class **`BasePlusCommissionEmployee`** constructor body executes after base-class **`CommissionEmployee`**'s constructor finishes execution

Destructors for **`BasePlusCommissionEmployee`** object called in reverse order of constructors

Destructors for **`BasePlusCommissionEmployee`** object called in reverse order of constructors

12.6 `public`, `protected` and `private` Inheritance

- `public` inheritance
  - Base class `public` members → derived class `public` members
  - Base class `protected` members → derived class `protected` members
  - Base class `private` members are not accessible
- `protected` inheritance (not *is-a* relationship)
  - Base class `public` and `protected` members → derived class `protected` members
- `private` inheritance (not *is-a* relationship)
  - Base class `public` and `protected` members → derived class `private` members

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

- Customizing existing software
  - Inheriting from existing classes
    - Can include additional members
    - Can redefine base-class members
    - No direct access to base class’s source code
      - Only links to object code
  - Independent software vendors (ISVs)
    - Develop proprietary code for sale/license
      - Available in object-code format
    - Users derive new classes
      - Without accessing ISV proprietary source code

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.

Software Engineering Observation 12.11

Just as designers of non-object-oriented systems should avoid proliferation of functions, designers of object-oriented systems should avoid proliferation of classes. Proliferation of classes creates management problems and can hinder software reusability, because it becomes difficult for a client to locate the most appropriate class of a huge class library. The alternative is to create fewer classes that provide more substantial functionality, but such classes might provide too much functionality.

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 is needed.