Chapter 12 Object-Oriented Programming: Inheritance

Yu-Shuen Wang, CS, NCTU

12.1 Introduction

- 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 contains behaviors inherited from its base class and can contain additional behaviors.
- A derived class can also customize behaviors inherited from the base class.

12.1 Introduction (cont.)

- A direct base class is the base class from which a derived class explicitly inherits.
- An indirect base class is inherited from two or more levels up in the class hierarchy.
- In the case of single inheritance, a class is derived from one base class.
- C++ also supports multiple inheritance, in which a derived class inherits from multiple (possibly unrelated) base classes.
  - We discuss multiple inheritance in Chapter 24, Other Topics.

12.1 Introduction (cont.)

- C++ offers `public`, `protected` and `private` inheritance.
- `Protected` inheritance, is rarely used.
- With `public` inheritance, every object of a derived class is also an object of that derived class's base class.
- However, base-class objects are not objects of their derived classes.
- A derived class can access the non-`private` members of its base class.

Software Engineering Observation 12.1

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

12.2 Base Classes and Derived Classes

- Often, an object of one class *is an* object of another class, as well.
  - For example, a rectangle *is a* quadrilateral.
  - Thus, in C++, class `Rectangle` can be said to inherit from class `Quadrilateral`.
  - In this context, class `Quadrilateral` is a base class, and class `Rectangle` is a derived class.
  - A rectangle *is a* specific type of quadrilateral, but it's incorrect to claim that a quadrilateral is a rectangle.

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.)

- 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, a derived class, or both.

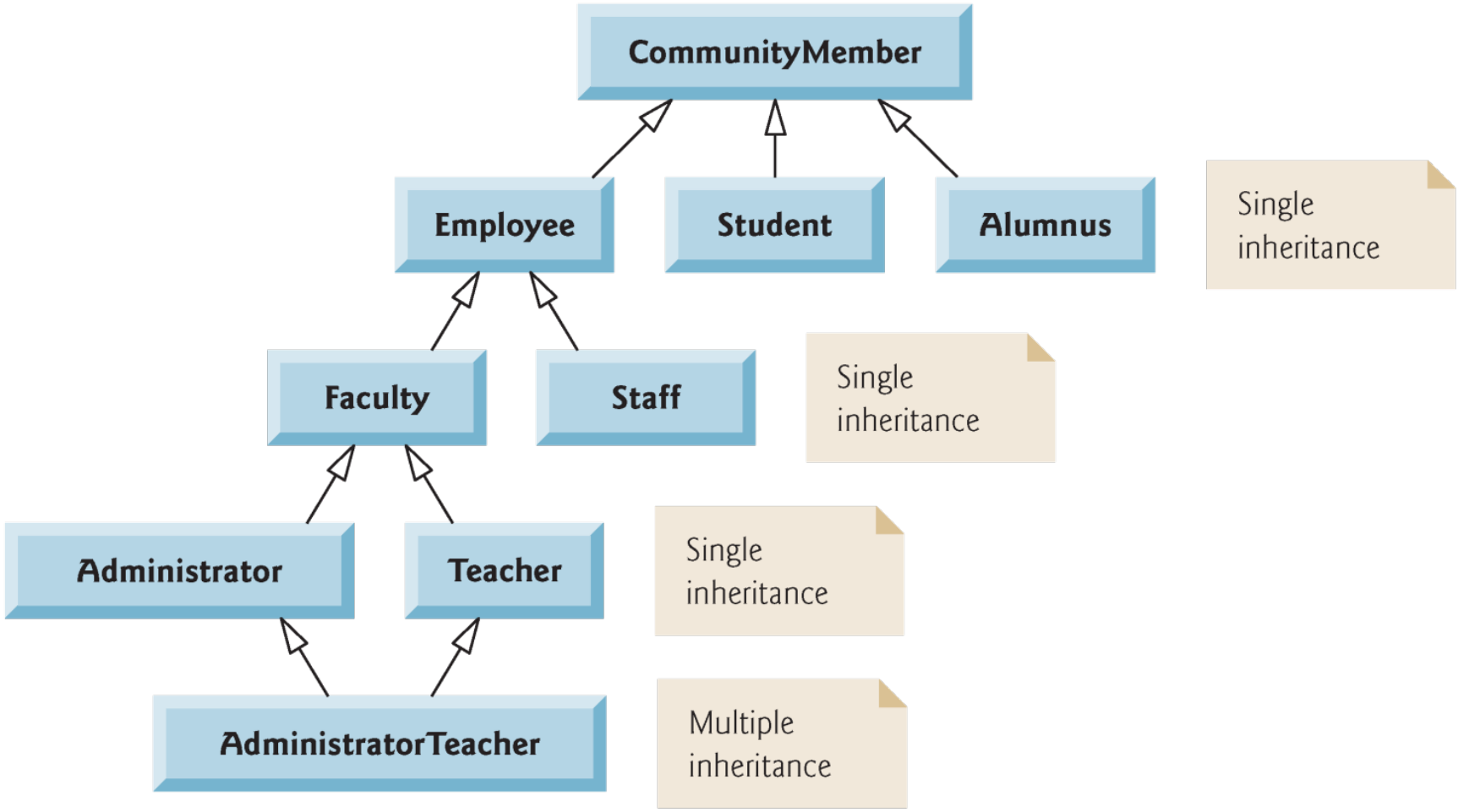

Fig. 12.2 | Inheritance hierarchy for university `CommunityMembers`.

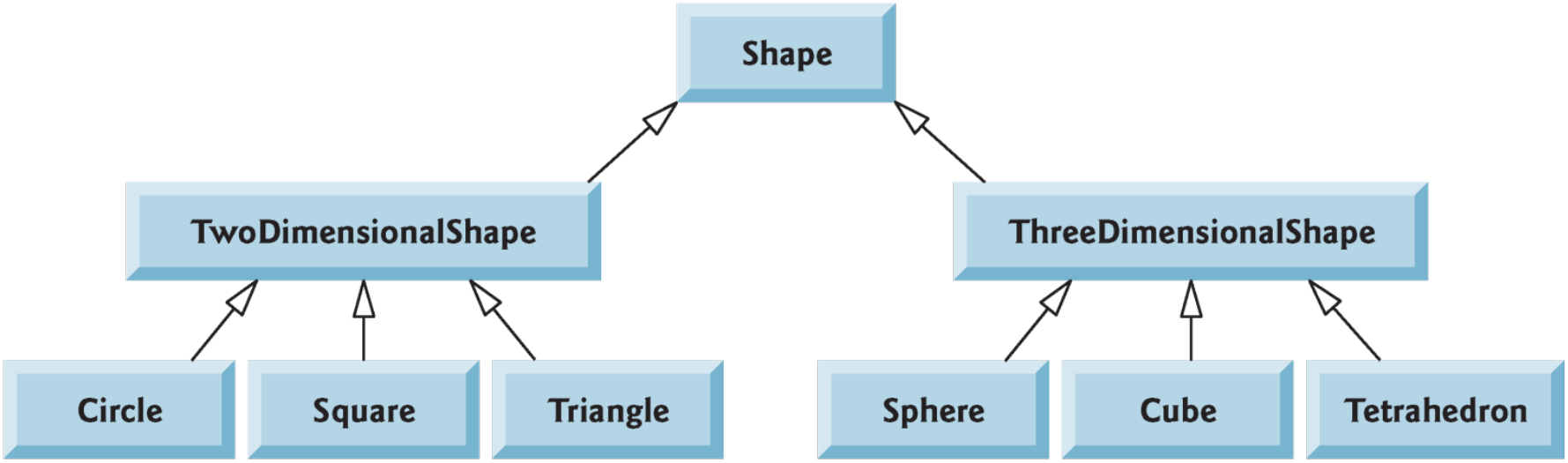

Fig. 12.3 | Inheritance hierarchy for Shapes.

12.3 `protected` Members

- Using `protected` access offers an intermediate level of protection between `public` and `private` access.
- A base class's `protected` members can be accessed within the body of that base class, by members and `friend`s of that base class, and by members and `friend`s of any classes derived from that base class.

```
1  // Fig. 12.4: CommissionEmployee.h
2  // CommissionEmployee class definition represents a commission employee.
3  #ifndef COMMISSION_H
4  #define COMMISSION_H
5
6  #include <string> // C++ standard string class
7  using namespace std;
8
9  class CommissionEmployee
10 {
11 public:
12    CommissionEmployee( const string &, const string &, const string &,
13       double = 0.0, double = 0.0 );
14
15    void setFirstName( const string & ); // set first name
16    string getFirstName() const; // return first name
17
18    void setLastName( const string & ); // set last name
19    string getLastName() const; // return last name
20
21    void setSocialSecurityNumber( const string & ); // set SSN
22    string getSocialSecurityNumber() const; // return SSN
23
```

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

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

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

```
1  // Fig. 12.5: CommissionEmployee.cpp
2  // Class CommissionEmployee member-function definitions.
3  #include <iostream>
4  #include "CommissionEmployee.h" // CommissionEmployee class definition
5  using namespace std;
6
7  // constructor
8  CommissionEmployee::CommissionEmployee(
9     const string &first, const string &last, const string &ssn,
10    double sales, double rate )
11 {
12    firstName = first; // should validate
13    lastName = last; // should validate
14    socialSecurityNumber = ssn; // should validate
15    setGrossSales( sales ); // validate and store gross sales
16    setCommissionRate( rate ); // validate and store commission rate
17 } // end CommissionEmployee constructor
18
19 // set first name
20 void CommissionEmployee::setFirstName( const string &first )
21 {
22    firstName = first; // should validate
23 } // end function setFirstName
```

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

```
24
25   // return first name
26   string CommissionEmployee::getFirstName() const
27   {
28      return firstName;
29   } // end function getFirstName
30
31   // set last name
32   void CommissionEmployee::setLastName( const string &last )
33   {
34      lastName = last; // should validate
35   } // end function setLastName
36
37   // return last name
38   string CommissionEmployee::getLastName() const
39   {
40      return lastName;
41   } // end function getLastName
42
```

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

```
43   // set social security number
44   void CommissionEmployee::setSocialSecurityNumber( const string &ssn )
45   {
46      socialSecurityNumber = ssn; // should validate
47   } // end function setSocialSecurityNumber
48
49   // return social security number
50   string CommissionEmployee::getSocialSecurityNumber() const
51   {
52      return socialSecurityNumber;
53   } // end function getSocialSecurityNumber
54
55   // set gross sales amount
56   void CommissionEmployee::setGrossSales( double sales )
57   {
58      grossSales = ( sales < 0.0 ) ? 0.0 : sales;
59   } // end function setGrossSales
60
61   // return gross sales amount
62   double CommissionEmployee::getGrossSales() const
63   {
64      return grossSales;
65   } // end function getGrossSales
```

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

```
66
67   // set commission rate
68   void CommissionEmployee::setCommissionRate( double rate )
69   {
70      commissionRate = ( rate > 0.0 && rate < 1.0 ) ? rate : 0.0;
71   } // end function setCommissionRate
72
73   // return commission rate
74   double CommissionEmployee::getCommissionRate() const
75   {
76      return commissionRate;
77   } // end function getCommissionRate
78
79   // calculate earnings
80   double CommissionEmployee::earnings() const
81   {
82      return commissionRate * grossSales;
83   } // end function earnings
84
```

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

```
85  // print CommissionEmployee object
86  void CommissionEmployee::print() const
87  {
88     cout << "commission employee: " << firstName << ' ' << lastName
89        << "\nsocial security number: " << socialSecurityNumber
90        << "\ngross sales: " << grossSales
91        << "\ncommission rate: " << commissionRate;
92  } // end function print
```

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

To build a BasePlusCommissionEmployee class, you can…

```
1  // Fig. 12.7: BasePlusCommissionEmployee.h
2  // BasePlusCommissionEmployee class definition represents an employee
3  // that receives a base salary in addition to commission.
4  #ifndef BASEPLUS_H
5  #define BASEPLUS_H
6
7  #include <string> // C++ standard string class
8  using namespace std;
9
10 class BasePlusCommissionEmployee
11 {
12 public:
13    BasePlusCommissionEmployee( const string &, const string &,
14       const string &, double = 0.0, double = 0.0, double = 0.0 );
15
16    void setFirstName( const string & ); // set first name
17    string getFirstName() const; // return first name
18
19    void setLastName( const string & ); // set last name
20    string getLastName() const; // return last name
21
22    void setSocialSecurityNumber( const string & ); // set SSN
23    string getSocialSecurityNumber() const; // return SSN
24
```

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

```
25      void setGrossSales( double ); // set gross sales amount
26      double getGrossSales() const; // return gross sales amount
27
28      void setCommissionRate( double ); // set commission rate
29      double getCommissionRate() const; // return commission rate
30
31      void setBaseSalary( double ); // set base salary
32      double getBaseSalary() const; // return base salary
33
34      double earnings() const; // calculate earnings
35      void print() const; // print BasePlusCommissionEmployee object
36   private:
37      string firstName;
38      string lastName;
39      string socialSecurityNumber;
40      double grossSales; // gross weekly sales
41      double commissionRate; // commission percentage
42      double baseSalary; // base salary
43   }; // end class BasePlusCommissionEmployee
44
45   #endif
```

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

Software Engineering Observation 12.4

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

```
1  // Fig. 12.10: BasePlusCommissionEmployee.h
2  // BasePlusCommissionEmployee class derived from class
3  // CommissionEmployee.
4  #ifndef BASEPLUS_H
5  #define BASEPLUS_H
6
7  #include <string> // C++ standard string class
8  #include "CommissionEmployee.h" // CommissionEmployee class declaration
9  using namespace std;
10
11 class BasePlusCommissionEmployee : public CommissionEmployee
12 {
13 public:
14    BasePlusCommissionEmployee( const string &, const string &,
15       const string &, double = 0.0, double = 0.0, double = 0.0 );
16
17    void setBaseSalary( double ); // set base salary
18    double getBaseSalary() const; // return base salary
19
20    double earnings() const; // calculate earnings
21    void print() const; // print BasePlusCommissionEmployee object
```

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

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

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

```
1  // Fig. 12.11: BasePlusCommissionEmployee.cpp
2  // Class BasePlusCommissionEmployee member-function definitions.
3  #include <iostream>
4  #include "BasePlusCommissionEmployee.h"
5  using namespace std;
6
7  // constructor
8  BasePlusCommissionEmployee::BasePlusCommissionEmployee(
9     const string &first, const string &last, const string &ssn,
10    double sales, double rate, double salary )
11    // explicitly call base-class constructor
12    : CommissionEmployee( first, last, ssn, sales, rate )
13 {
14    setBaseSalary( salary ); // validate and store base salary
15 } // end BasePlusCommissionEmployee constructor
16
17 // set base salary
18 void BasePlusCommissionEmployee::setBaseSalary( double salary )
19 {
20    baseSalary = ( salary < 0.0 ) ? 0.0 : salary;
21 } // end function setBaseSalary
22
```

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

```
23  // return base salary
24  double BasePlusCommissionEmployee::getBaseSalary() const
25  {
26     return baseSalary;
27  } // end function getBaseSalary
28
29  // calculate earnings
30  double BasePlusCommissionEmployee::earnings() const
31  {
32     // derived class cannot access the base class's private data
33     return baseSalary + ( commissionRate * grossSales );
34  } // end function earnings
35
36  // print BasePlusCommissionEmployee object
37  void BasePlusCommissionEmployee::print() const
38  {
39     // derived class cannot access the base class's private data
40     cout << "base-salaried commission employee: " << firstName << ' '
41        << lastName << "\nsocial security number: " << socialSecurityNumber
42        << "\ngross sales: " << grossSales
43        << "\ncommission rate: " << commissionRate
44        << "\nbase salary: " << baseSalary;
45  } // end function print
```

Fig. 12.11 | `BasePlusCommissionEmployee` implementation file: `private` base-class data cannot be accessed from derived class. (Part 2 of 4.)

12.4.4 CommissionEmployee–BasePlusCommissionEmployee Inheritance Hierarchy Using protected Data

- To enable class `BasePlusCommissionEmployee` to directly access `CommissionEmployee` data members `firstName`, `lastName`, `socialSecurityNumber`, `grossSales` and `commissionRate`, we can declare those members as `protected` in the base class.

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

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

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

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

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

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

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

- Derived classes should depend only on the base-class 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.
- A small change in the base class can "break" derived-class implementation.

A software engineering would write the code like ...

```
1  // Fig. 12.17: CommissionEmployee.h
2  // CommissionEmployee class definition with good software engineering.
3  #ifndef COMMISSION_H
4  #define COMMISSION_H
5
6  #include <string> // C++ standard string class
7  using namespace std;
8
9  class CommissionEmployee
10 {
11 public:
12    CommissionEmployee( const string &, const string &, const string &,
13       double = 0.0, double = 0.0 );
14
15    void setFirstName( const string & ); // set first name
16    string getFirstName() const; // return first name
17
18    void setLastName( const string & ); // set last name
19    string getLastName() const; // return last name
20
21    void setSocialSecurityNumber( const string & ); // set SSN
22    string getSocialSecurityNumber() const; // return SSN
```

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

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

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

```
1  // Fig. 12.19: BasePlusCommissionEmployee.h
2  // BasePlusCommissionEmployee class derived from class
3  // CommissionEmployee.
4  #ifndef BASEPLUS_H
5  #define BASEPLUS_H
6
7  #include <string> // C++ standard string class
8  #include "CommissionEmployee.h" // CommissionEmployee class declaration
9  using namespace std;
10
11 class BasePlusCommissionEmployee : public CommissionEmployee
12 {
13 public:
14    BasePlusCommissionEmployee( const string &, const string &,
15       const string &, double = 0.0, double = 0.0, double = 0.0 );
16
17    void setBaseSalary( double ); // set base salary
18    double getBaseSalary() const; // return base salary
19
20    double earnings() const; // calculate earnings
21    void print() const; // print BasePlusCommissionEmployee object
```

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

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

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

```
1  // Fig. 12.20: BasePlusCommissionEmployee.cpp
2  // Class BasePlusCommissionEmployee member-function definitions.
3  #include <iostream>
4  #include "BasePlusCommissionEmployee.h"
5  using namespace std;
6
7  // constructor
8  BasePlusCommissionEmployee::BasePlusCommissionEmployee(
9     const string &first, const string &last, const string &ssn,
10    double sales, double rate, double salary )
11    // explicitly call base-class constructor
12    : CommissionEmployee( first, last, ssn, sales, rate )
13 {
14    setBaseSalary( salary ); // validate and store base salary
15 } // end BasePlusCommissionEmployee constructor
16
17 // set base salary
18 void BasePlusCommissionEmployee::setBaseSalary( double salary )
19 {
20    baseSalary = ( salary < 0.0 ) ? 0.0 : salary;
21 } // end function setBaseSalary
22
```

Fig. 12.20 | `BasePlusCommissionEmployee` class that inherits from class `CommissionEmployee` but cannot directly access the class's `private` data. (Part 1 of 2.)

```
23   // return base salary
24   double BasePlusCommissionEmployee::getBaseSalary() const
25   {
26      return baseSalary;
27   } // end function getBaseSalary
28
29   // calculate earnings
30   double BasePlusCommissionEmployee::earnings() const
31   {
32      return getBaseSalary() + CommissionEmployee::earnings();
33   } // end function earnings
34
35   // print BasePlusCommissionEmployee object
36   void BasePlusCommissionEmployee::print() const
37   {
38      cout << "base-salaried ";
39
40      // invoke CommissionEmployee's print function
41      CommissionEmployee::print();
42
43      cout << "\nbase salary: " << getBaseSalary();
44   } // end function print
```

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

```
1  // Fig. 12.21: fig12_21.cpp
2  // Testing class BasePlusCommissionEmployee.
3  #include <iostream>
4  #include <iomanip>
5  #include "BasePlusCommissionEmployee.h"
6  using namespace std;
7
8  int main()
9  {
10    // instantiate BasePlusCommissionEmployee object
11    BasePlusCommissionEmployee
12       employee( "Bob", "Lewis", "333-33-3333", 5000, .04, 300 );
13
14    // set floating-point output formatting
15    cout << fixed << setprecision( 2 );
16
17    // get commission employee data
18    cout << "Employee information obtained by get functions: \n"
19       << "\nFirst name is " << employee.getFirstName()
20       << "\nLast name is " << employee.getLastName()
21       << "\nSocial security number is "
22       << employee.getSocialSecurityNumber()
```

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

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

Fig. 12.21 | Base-class `private` data is accessible to a derived class via `public` or `protected` member function inherited by the derived class. (Part 2 of 3.)

Performance Tip 12.2

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

12.5 Constructors and Destructors in Derived Classes

Software Engineering Observation 12.7

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

Software Engineering Observation 12.8

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

```
1  // Fig. 12.23: CommissionEmployee.cpp
2  // Class CommissionEmployee member-function definitions.
3  #include <iostream>
4  #include "CommissionEmployee.h" // CommissionEmployee class definition
5  using namespace std;
6
7  // constructor
8  CommissionEmployee::CommissionEmployee(
9     const string &first, const string &last, const string &ssn,
10    double sales, double rate )
11    : firstName( first ), lastName( last ), socialSecurityNumber( ssn )
12 {
13    setGrossSales( sales ); // validate and store gross sales
14    setCommissionRate( rate ); // validate and store commission rate
15
16    cout << "CommissionEmployee constructor: " << endl;
17    print();
18    cout << "\n\n";
19 } // end CommissionEmployee constructor
20
```

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

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

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

```
1  // Fig. 12.25: BasePlusCommissionEmployee.cpp
2  // Class BasePlusCommissionEmployee member-function definitions.
3  #include <iostream>
4  #include "BasePlusCommissionEmployee.h"
5  using namespace std;
6
7  // constructor
8  BasePlusCommissionEmployee::BasePlusCommissionEmployee(
9     const string &first, const string &last, const string &ssn,
10    double sales, double rate, double salary )
11    // explicitly call base-class constructor
12    : CommissionEmployee( first, last, ssn, sales, rate )
13 {
14    setBaseSalary( salary ); // validate and store base salary
15
16    cout << "BasePlusCommissionEmployee constructor: " << endl;
17    print();
18    cout << "\n\n";
19 } // end BasePlusCommissionEmployee constructor
20
```

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

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

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

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

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

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

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

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


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

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

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


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

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

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

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

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

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

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

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

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

- When deriving a class from a base class, the base class may be inherited through `public`, `protected` or `private` inheritance.
- Use of `protected` and `private` inheritance is rare, and each should be used only with great care.

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.

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.

Performance Tip 12.3

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