C++

Inheritance

Inheritance

- Introduction
- Base Classes and Derived Classes
- Five Examples of Base Class and Derived Class Relationships
- Constructors and Destructors in Derived Classes

Introduction

- **Inheritance** is a form of software reuse where a class is created that absorbs an existing class's data and behaviors and enhances them with new capabilities.
- The new class, **the derived class**, inherits the members of the existing class, **the base class**.

Introduction

- 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 **single inheritance**, a class is derived from one base class.
- With multiple inheritance, a derived class inherits from multiple base classes.

Introduction

- C++ offers three types of inheritance:
  - **public**:: every object of a derived class is also an object of that derived class's base class. {Note, base-class objects are NOT objects of their derived classes.}
  - **private**:: is essentially an alternative to composition.
  - **protected**:: is rarely used.

Software Engineering Observation 23.1

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

C++ Abstractions

- **is-a** relationship:: inheritance
  - e.g., derived class , car, is a base class, vehicle.
- **has-a** relationship:: composition
  - e.g. object employee has an object birthdate.

Base Classes and Derived Classes

- Base class typically represents larger set of objects than derived classes.

Example

Base class: `Vehicle`

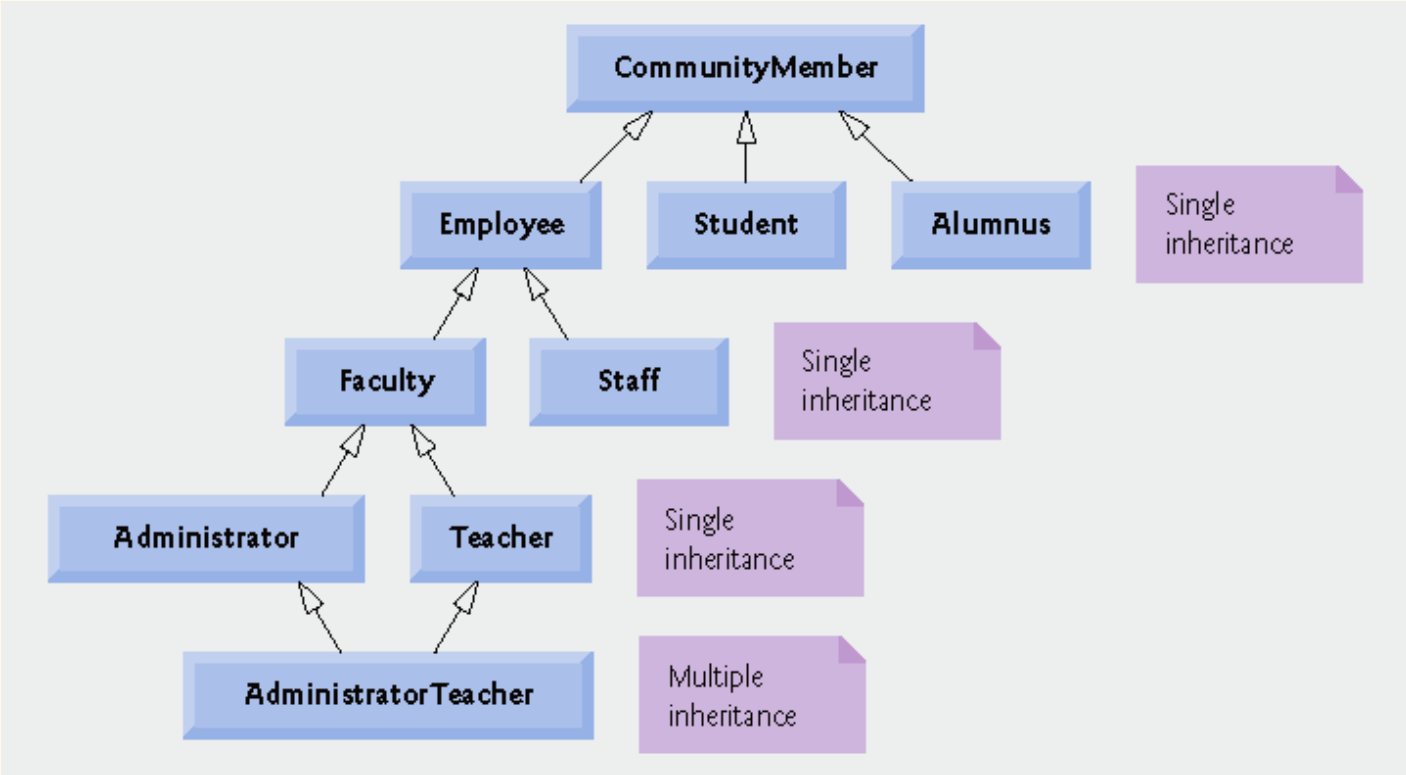
Includes cars, trucks, boats, bicycles, etc.

Derived class: `car`

a smaller, more-specific subset of vehicles

- Inheritance relationships form treelike hierarchical structures.

Fig. 23.2 Inheritance Hierarchy for University CommunityMember



Base Classes and Derived Classes

- `public` inheritance specified by:

  ```
  Class Employee : public CommunityMember
  ```

- Class `Employee` inherits from class `CommunityMember`
- Base class `private` members are not accessible directly, but they are inherited.
  - Manipulated through inherited `public` member functions.
- Base class `public` and `protected` members
  - Are inherited with original member access.
- `friend` functions
  - Are not inherited.

protected Members

- A base class's **protected** members can be accessed within the body of that base class by members and friends of that base class and by members and friends of any class derived from that base class.
- By simply using member names, derived-class member functions can refer to **public** and **protected** members of the base class.
- When a derived-class member function redefines a base-class member function, by preceding the base-class member with the base-class name and the binary scope resolution operator (::), the derived-class can access the base-class member.

Five Examples of Base Class and Derived Class Relationships

1. Create a `CommissionEmployee` class with private data members: First name, last name, SSN, commission rate, gross sale amount.
2. Create a `BasePlusCommissionEmployee` class without inheritance with private data members : First name, last name, SSN, commission rate, gross sale amount and base salary.

Five Examples of Base Class and Derived Class Relationships

3. Create a CommissionEmpolyee-BasePlusCommissionEmployee inheritance hierarchy with private members.
4. Create a CommissionEmpolyee-BasePlusCommissionEmployee inheritance hierarchy with protected members.
5. Create a CommissionEmpolyee-BasePlusCommissionEmployee inheritance hierarchy with private members but access through public member functions.

Example 1: CommissionEmployee Class

- **CommissionEmployee** header file
  - Specify public services:
    - Constructor
    - *get* and *set* functions
    - Member functions **earnings** and **print**
- **CommissionEmployee** source code file
  - Specify member-function definitions.

Example 1: CommissionEmployee Class

```
1  // Fig. 23.4: 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 std::string;
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
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
```

Class **CommissionEmployee** constructor

Example 1: CommissionEmployee Class

```
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

Example 1: CommissionEmployee Class

```
1  // Fig. 23.5: CommissionEmployee.cpp
2  // Class CommissionEmployee member-function definitions.
3  #include <iostream>
4  using std::cout;
5
6  #include "CommissionEmployee.h" // CommissionEmployee class definition
7
8  // constructor
9  CommissionEmployee::CommissionEmployee(
10    const string &first, const string &last, const string &ssn,
11    double sales, double rate )
12 {
13    firstName = first; // should validate
14    lastName = last;   // should validate
15    socialSecurityNumber = ssn; // should validate
16    setGrossSales( sales ); // validate and store gross sales
17    setCommissionRate( rate ); // validate and store commission rate
18 } // end CommissionEmployee constructor
19
20 // set first name
21 void CommissionEmployee::setFirstName( const string &first )
22 {
23    firstName = first; // should validate
24 } // end function setFirstName
25
26 // return first name
27 string CommissionEmployee::getFirstName() const
28 {
29    return firstName;
30 } // end function getFirstName
```

Initialize data members

Example 1: CommissionEmployee Class

```
31
32 // set last name
33 void CommissionEmployee::setLastName( const string &last )
34 {
35    lastName = last; // should validate
36 } // end function setLastName
37
38 // return last name
39 string CommissionEmployee::getLastName() const
40 {
41    return lastName;
42 } // end function getLastName
43
44 // set social security number
45 void CommissionEmployee::setSocialSecurityNumber( const string &ssn )
46 {
47    socialSecurityNumber = ssn; // should validate
48 } // end function setSocialSecurityNumber
49
50 // return social security number
51 string CommissionEmployee::getSocialSecurityNumber() const
52 {
53    return socialSecurityNumber;
54 } // end function getSocialSecurityNumber
55
56 // set gross sales amount
57 void CommissionEmployee::setGrossSales( double sales )
58 {
59    grossSales = ( sales < 0.0 ) ? 0.0 : sales;
60 } // end function setGrossSales
```

Function **`setGrossSales`** validates gross sales amount

Example 1: CommissionEmployee Class

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

Function **`setCommissionRate`** validates commission rate

Example 1: CommissionEmployee Class

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

Function **earnings** calculates earnings

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

Example 1: CommissionEmployee Class

```
1  // Fig. 23.6: fig23_06.cpp
2  // Testing class CommissionEmployee.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6  using std::fixed;
7
8  #include <iomanip>
9  using std::setprecision;
10
11 #include "CommissionEmployee.h" // CommissionEmployee class definition
12
13 int main()
14 {
15    // instantiate a CommissionEmployee object
16    CommissionEmployee employee(
17       "Sue", "Jones", "222-22-2222", 10000, .06 );
18
19    // set floating-point output formatting
20    cout << fixed << setprecision( 2 );
21
22    // get commission employee data
23    cout << "Employee information obtained by get functions: \n"
24       << "\nFirst name is " << employee.getFirstName()
25       << "\nLast name is " << employee.getLastName()
26       << "\nSocial security number is "
27       << employee.getSocialSecurityNumber()
28       << "\nGross sales is " << employee.getGrossSales()
29       << "\nCommission rate is " << employee.getCommissionRate() << endl;
```

Instantiate **`CommissionEmployee`** object

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

Example 1: CommissionEmployee Class

```
30
31    employee.setGrossSales( 8000 ); // set gross sales
32    employee.setCommissionRate( .1 ); // set commission rate
33
34    cout << "\nUpdated employee information output by prin
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

```
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
```

Example 2: BasePlusCommissionEmployee Class

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

Example 2: BasePlusCommissionEmployee Class

```
1  // Fig. 23.7: 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 std::string;
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
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
```

Constructor takes one more argument, which specifies the base salary

Example 2: BasePlusCommissionEmployee Class

```
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
```

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

Add data member **`baseSalary`**

Example 2: BasePlusCommissionEmployee Class

```
1  // Fig. 23.8: BasePlusCommissionEmployee.cpp
2  // Class BasePlusCommissionEmployee member-function definitions.
3  #include <iostream>
4  using std::cout;
5
6  // BasePlusCommissionEmployee class definition
7  #include "BasePlusCommissionEmployee.h"
8
9  // constructor
10 BasePlusCommissionEmployee::BasePlusCommissionEmployee(
11    const string &first, const string &last, const string &ssn,
12    double sales, double rate, double salary )
13 {
14    firstName = first; // should validate
15    lastName = last; // should validate
16    socialSecurityNumber = ssn; // should validate
17    setGrossSales( sales ); // validate and store gross sales
18    setCommissionRate( rate ); // validate and store commission rate
19    setBaseSalary( salary ); // validate and store base salary
20 } // end BasePlusCommissionEmployee constructor
21
22 // set first name
23 void BasePlusCommissionEmployee::setFirstName( const string &first )
24 {
25    firstName = first; // should validate
26 } // end function setFirstName
```

Constructor takes one more argument, which specifies the base salary

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

Example 2: BasePlusCommissionEmployee Class

```
27
28 // return first name
29 string BasePlusCommissionEmployee::getFirstName() const
30 {
31    return firstName;
32 } // end function getFirstName
33
34 // set last name
35 void BasePlusCommissionEmployee::setLastName( const string &last )
36 {
37    lastName = last; // should validate
38 } // end function setLastName
39
40 // return last name
41 string BasePlusCommissionEmployee::getLastName() const
42 {
43    return lastName;
44 } // end function getLastName
45
46 // set social security number
47 void BasePlusCommissionEmployee::setSocialSecurityNumber(
48    const string &ssn )
49 {
50    socialSecurityNumber = ssn; // should validate
51 } // end function setSocialSecurityNumber
52
```

Example 2: BasePlusCommissionEmployee Class

```
53 // return social security number
54 string BasePlusCommissionEmployee::getSocialSecurityNumber() const
55 {
56    return socialSecurityNumber;
57 } // end function getSocialSecurityNumber
58
59 // set gross sales amount
60 void BasePlusCommissionEmployee::setGrossSales( double sales )
61 {
62    grossSales = ( sales < 0.0 ) ? 0.0 : sales;
63 } // end function setGrossSales
64
65 // return gross sales amount
66 double BasePlusCommissionEmployee::getGrossSales() const
67 {
68    return grossSales;
69 } // end function getGrossSales
70
71 // set commission rate
72 void BasePlusCommissionEmployee::setCommissionRate( double rate )
73 {
74    commissionRate = ( rate > 0.0 && rate < 1.0 ) ? rate : 0.0;
75 } // end function setCommissionRate
76
77 // return commission rate
78 double BasePlusCommissionEmployee::getCommissionRate() const
79 {
80    return commissionRate;
81 } // end function getCommissionRate
82
```

Example 2: BasePlusCommissionEmployee Class

```
83 // set base salary
84 void BasePlusCommissionEmployee::setBaseSalary( double salary )
85 {
86    baseSalary = ( salary < 0.0 ) ? 0.0 : salary;
87 } // end function setBaseSalary
88
89 // return base salary
90 double BasePlusCommissionEmployee::getBaseSalary() const
91 {
92    return baseSalary;
93 } // end function getBaseSalary
94
95 // calculate earnings
96 double BasePlusCommissionEmployee::earnings() const
97 {
98    return baseSalary + ( commissionRate * grossSales );
99 } // end function earnings
100
101// print BasePlusCommissionEmployee object
102void BasePlusCommissionEmployee::print() const
103{
104   cout << "base-salaried commission employee: " << firstName << ' '
105      << lastName << "\nsocial security number: " << socialSecurityNumber
106      << "\ngross sales: " << grossSales
107      << "\ncommission rate: " << commissionRate
108      << "\nbase salary: " << baseSalary;
109} // end function print
```

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

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

Example 2: BasePlusCommissionEmployee Class

```
1  // Fig. 23.9: fig23_09.cpp
2  // Testing class BasePlusCommissionEmployee.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6  using std::fixed;
7
8  #include <iomanip>
9  using std::setprecision;
10
11 // BasePlusCommissionEmployee class definition
12 #include "BasePlusCommissionEmployee.h"
13
14 int main()
15 {
16    // instantiate BasePlusCommissionEmployee object
17    BasePlusCommissionEmployee
18       employee( "Bob", "Lewis", "333-33-3333", 5000, .04, 300 );
19
20    // set floating-point output formatting
21    cout << fixed << setprecision( 2 );
22
```

Instantiate **BasePlusCommissionEmployee** object

Example 2: BasePlusCommissionEmployee Class

```
23    // get commission employee data
24    cout << "Employee information obtained by get functions: \n"
25       << "\nFirst name is " << employee.getFirstName()
26       << "\nLast name is " << employee.getLastName()
27       << "\nSocial security number is "
28       << employee.getSocialSecurityNumber()
29       << "\nGross sales is " << employee.getGrossSales()
30       << "\nCommission rate is " << employee.getCommissionRate()
31       << "\nBase salary is " << employee.getBaseSalary() << endl;
32
33    employee.setBaseSalary( 1000 ); // set base salary
34
35    cout << "\nUpdated employee information output by
36       << endl;
37    employee.print(); // display the new employee information
38
39    // display the employee's earnings
40    cout << "\n\nEmployee's earnings: $" << employee.earnings() << endl;
41
42    return 0;
43 } // end main
```

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

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

Example 2: BasePlusCommissionEmployee Class

```
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
```

Software Engineering Observation 23.4

- 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.

Example 3: 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**.

Example 3: a CommissionEmployee-BasePlusCommissionEmployee Inheritance Hierarchy

```
1  // Fig. 23.10: 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  using std::string;
9
10 #include "CommissionEmployee.h" // CommissionEmployee class declaration
11
12 class BasePlusCommissionEmployee : public CommissionEmployee
13 {
14 public:
15    BasePlusCommissionEmployee( const string &, const string &,
16       const string &, double = 0.0, double = 0.0, double = 0.0 );
17
18    void setBaseSalary( double ); // set base salary
19    double getBaseSalary() const; // return base salary
20
21    double earnings() const; // calculate earnings
22    void print() const; // print BasePlusCommissionEmployee object
23 private:
24    double baseSalary; // base salary
25 }; // end class BasePlusCommissionEmployee
26
27 #endif
```

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

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

Example 3: a CommissionEmployee-BasePlusCommissionEmployee Inheritance Hierarchy

```
1  // Fig. 23.11: BasePlusCommissionEmployee.cpp
2  // Class BasePlusCommissionEmployee member-function definitions.
3  #include <iostream>
4  using std::cout;
5
6  // BasePlusCommissionEmployee class definition
7  #include "BasePlusCommissionEmployee.h"
8
9  // constructor
10 BasePlusCommissionEmployee::BasePlusCommissionEmployee(
11    const string &first, const string &last, const string &ssn,
12    double sales, double rate, double salary )
13    // explicitly call base-class constructor
14    : CommissionEmployee( first, last, ssn, sales, rate )
15 {
16    setBaseSalary( salary ); // validate and store base salary
17 } // end BasePlusCommissionEmployee constructor
18
19 // set base salary
20 void BasePlusCommissionEmployee::setBaseSalary( double salary )
21 {
22    baseSalary = ( salary < 0.0 ) ? 0.0 : salary;
23 } // end function setBaseSalary
24
25 // return base salary
26 double BasePlusCommissionEmployee::getBaseSalary() const
27 {
28    return baseSalary;
29 } // end function getBaseSalary
```

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

Example 3: a CommissionEmployee-BasePlusCommissionEmployee Inheritance Hierarchy

```
30
31 // calculate earnings
32 double BasePlusCommissionEmployee::earnings() const
33 {
34    // derived class cannot access the base class's private data
35    return baseSalary + ( commissionRate * grossSales );
36 } // end function earnings
37
38 // print BasePlusCommissionEmployee object
39 void BasePlusCommissionEmployee::print() const
40 {
41    // derived class cannot access the base class's private data
42    cout << "base-salaried commission employee: " << firstName << ' '
43       << lastName << "\nsocial security number: " << socialSecurityNumber
44       << "\ngross sales: " << grossSales
45       << "\ncommission rate: " << commissionRate
46       << "\nbase salary: " << baseSalary;
47 } // 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**

Example 3: a CommissionEmployee-BasePlusCommissionEmployee Inheritance Hierarchy

```
C:\examples\ch23\Fig23_10_11\BasePlusCommission-Employee.cpp(35) :
   error C2248: 'CommissionEmployee::commissionRate' :
   cannot access private member declared in class 'CommissionEmployee'
      C:\examples\ch23\Fig23_10_11\CommissionEmployee.h(37) :
         see declaration of 'CommissionEmployee::commissionRate'
      C:\examples\ch23\Fig23_10_11\CommissionEmployee.h(10) :
         see declaration of 'CommissionEmployee'

C:\examples\ch23\Fig23_10_11\BasePlusCommission-Employee.cpp(35) :
   error C2248: 'CommissionEmployee::grossSales' :
   cannot access private member declared in class 'CommissionEmployee'
      C:\examples\ch23\Fig23_10_11\CommissionEmployee.h(36) :
         see declaration of 'CommissionEmployee::grossSales'
      C:\examples\ch23\Fig23_10_11\CommissionEmployee.h(10) :
         see declaration of 'CommissionEmployee'

C:\examples\ch23\Fig23_10_11\BasePlusCommission-Employee.cpp(42) :
   error C2248: 'CommissionEmployee::firstName' :
   cannot access private member declared in class 'CommissionEmployee'
      C:\examples\ch23\Fig23_10_11\CommissionEmployee.h(33) :
         see declaration of 'CommissionEmployee::firstName'
      C:\examples\ch23\Fig23_10_11\CommissionEmployee.h(10) :
         see declaration of 'CommissionEmployee'
```

Example 3: a CommissionEmployee-BasePlusCommissionEmployee Inheritance Hierarchy

```
C:\examples\ch23\Fig23_10_11\BasePlusCommission-Employee.cpp(43) :
   error C2248: 'CommissionEmployee::lastName' :
   cannot access private member declared in class 'CommissionEmployee'
      C:\examples\ch23\Fig23_10_11\CommissionEmployee.h(34) :
         see declaration of 'CommissionEmployee::lastName'
      C:\examples\ch23\Fig23_10_11\CommissionEmployee.h(10) :
         see declaration of 'CommissionEmployee'

C:\examples\ch23\Fig23_10_11\BasePlusCommission-Employee.cpp(43) :
   error C2248: 'CommissionEmployee::socialSecurity-Number' :
   cannot access private member declared in class 'CommissionEmployee'
      C:\examples\ch23\Fig23_10_11\CommissionEmployee.h(35) :
         see declaration of 'CommissionEmployee::socialSecurityNumber'
      C:\examples\ch23\Fig23_10_11\CommissionEmployee.h(10) :
         see declaration of 'CommissionEmployee'

C:\examples\ch3\Fig23_10_11\BasePlusCommission-Employee.cpp(44) :
   error C2248: 'CommissionEmployee::grossSales' :
   cannot access private member declared in class 'CommissionEmployee'
      C:\examples\ch23\Fig23_10_11\CommissionEmployee.h(36) :
         see declaration of 'CommissionEmployee::grossSales'
      C:\examples\ch23\Fig23_10_11\CommissionEmployee.h(10) :
         see declaration of 'CommissionEmployee'

C:\examples\ch23\Fig23_10_11\BasePlusCommission-Employee.cpp(45) :
   error C2248: 'CommissionEmployee::commissionRate' :
   cannot access private member declared in class 'CommissionEmployee'
      C:\examples\ch23\Fig23_10_11\CommissionEmployee.h(37) :
         see declaration of 'CommissionEmployee::commissionRate'
      C:\examples\ch23\Fig23_10_11\CommissionEmployee.h(10) :
         see declaration of 'CommissionEmployee'
```

Example 3: a CommissionEmployee-BasePlusCommissionEmployee Inheritance Hierarchy

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

Example 4: a CommissionEmployee-BasePlusCommissionEmployee Inheritance Hierarchy using Protected Data

- Uses `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.

Example 4: a CommissionEmployee-BasePlusCommissionEmployee Inheritance Hierarchy using Protected Data

```
1  // Fig. 23.12: 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 std::string;
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
```

Example 4: a CommissionEmployee-BasePlusCommissionEmployee Inheritance Hierarchy using Protected Data

```
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
```

Example 4: a CommissionEmployee-BasePlusCommissionEmployee Inheritance Hierarchy using Protected Data

```
1  // Fig. 23.13: CommissionEmployee.cpp
2  // Class CommissionEmployee member-function definitions.
3  #include <iostream>
4  using std::cout;
5
6  #include "CommissionEmployee.h" // CommissionEmployee class definition
7
8  // constructor
9  CommissionEmployee::CommissionEmployee(
10    const string &first, const string &last, const string &ssn,
11    double sales, double rate )
12 {
13    firstName = first; // should validate
14    lastName = last; // should validate
15    socialSecurityNumber = ssn; // should validate
16    setGrossSales( sales ); // validate and store gross sales
17    setCommissionRate( rate ); // validate and store commission rate
18 } // end CommissionEmployee constructor
19
20 // set first name
21 void CommissionEmployee::setFirstName( const string &first )
22 {
23    firstName = first; // should validate
24 } // end function setFirstName
25
26 // return first name
27 string CommissionEmployee::getFirstName() const
28 {
29    return firstName;
30 } // end function getFirstName
```

Example 4: a CommissionEmployee-BasePlusCommissionEmployee Inheritance Hierarchy using Protected Data

```
31
32 // set last name
33 void CommissionEmployee::setLastName( const string &last )
34 {
35    lastName = last; // should validate
36 } // end function setLastName
37
38 // return last name
39 string CommissionEmployee::getLastName() const
40 {
41    return lastName;
42 } // end function getLastName
43
44 // set social security number
45 void CommissionEmployee::setSocialSecurityNumber( const string &ssn )
46 {
47    socialSecurityNumber = ssn; // should validate
48 } // end function setSocialSecurityNumber
49
50 // return social security number
51 string CommissionEmployee::getSocialSecurityNumber() const
52 {
53    return socialSecurityNumber;
54 } // end function getSocialSecurityNumber
55
56 // set gross sales amount
57 void CommissionEmployee::setGrossSales( double sales )
58 {
59    grossSales = ( sales < 0.0 ) ? 0.0 : sales;
60 } // end function setGrossSales
```

Example 4: a CommissionEmployee-BasePlusCommissionEmployee Inheritance Hierarchy using Protected Data

```
61
62 // return gross sales amount
63 double CommissionEmployee::getGrossSales() const
64 {
65    return grossSales;
66 } // end function getGrossSales
67
68 // set commission rate
69 void CommissionEmployee::setCommissionRate( double rate )
70 {
71    commissionRate = ( rate > 0.0 && rate < 1.0 ) ? rate : 0.0;
72 } // end function setCommissionRate
73
74 // return commission rate
75 double CommissionEmployee::getCommissionRate() const
76 {
77    return commissionRate;
78 } // end function getCommissionRate
79
80 // calculate earnings
81 double CommissionEmployee::earnings() const
82 {
83    return commissionRate * grossSales;
84 } // end function earnings
```

Example 4:
a CommissionEmployee-BasePlusCommissionEmployee Inheritance Hierarchy using Protected Data

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

Example 4: a CommissionEmployee-BasePlusCommissionEmployee Inheritance Hierarchy using Protected Data

```
1  // Fig. 23.14: BasePlusCommissionEmployee.h
2  // BasePlusCommissionEmployee class derived from class
3  // CommissionEmployee.
4  #ifndef BASEPLUS_H
5  #define BASEPLUS_H
6
7  #include <string> // C++ standard string class
8  using std::string;
9
10 #include "CommissionEmployee.h" // CommissionEmployee class declaration
11
12 class BasePlusCommissionEmployee : public CommissionEmployee
13 {
14 public:
15    BasePlusCommissionEmployee( const string &, const string &,
16       const string &, double = 0.0, double = 0.0, double = 0.0
17
18    void setBaseSalary( double ); // set base salary
19    double getBaseSalary() const; // return base salary
20
21    double earnings() const; // calculate earnings
22    void print() const; // print BasePlusCommissionEmployee object
23 private:
24    double baseSalary; // base salary
25 }; // end class BasePlusCommissionEmployee
26
27 #endif
```

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

Example 4: a CommissionEmployee-BasePlusCommissionEmployee Inheritance Hierarchy using Protected Data

```
1  // Fig. 23.15: BasePlusCommissionEmployee.cpp
2  // Class BasePlusCommissionEmployee member-function definitions.
3  #include <iostream>
4  using std::cout;
5
6  // BasePlusCommissionEmployee class definition
7  #include "BasePlusCommissionEmployee.h"
8
9  // constructor
10 BasePlusCommissionEmployee::BasePlusCommissionEmployee(
11    const string &first, const string &last, const string &ssn,
12    double sales, double rate, double salary )
13    // explicitly call base-class constructor
14    : CommissionEmployee( first, last, ssn, sales, rate )
15 {
16    setBaseSalary( salary ); // validate and store base salary
17 } // end BasePlusCommissionEmployee constructor
18
19 // set base salary
20 void BasePlusCommissionEmployee::setBaseSalary( double salary )
21 {
22    baseSalary = ( salary < 0.0 ) ? 0.0 : salary;
23 } // end function setBaseSalary
24
25 // return base salary
26 double BasePlusCommissionEmployee::getBaseSalary() const
27 {
28    return baseSalary;
29 } // end function getBaseSalary
```

Call base-class constructor using base-class initializer syntax

Example 4: a CommissionEmployee-BasePlusCommissionEmployee Inheritance Hierarchy using Protected Data

```
30
31 // calculate earnings
32 double BasePlusCommissionEmployee::earnings() const
33 {
34    // can access protected data of base class
35    return baseSalary + ( commissionRate * grossSales );
36 } // end function earnings
37
38 // print BasePlusCommissionEmployee object
39 void BasePlusCommissionEmployee::print() const
40 {
41    // can access protected data of base class
42    cout << "base-salaried commission employee: " << firstName << ' '
43       << lastName << "\nsocial security number: " << socialSecurityNumber
44       << "\ngross sales: " << grossSales
45       << "\ncommission rate: " << commissionRate
46       << "\nbase salary: " << baseSalary;
47 } // end function print
```

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

Example 4: a CommissionEmployee-BasePlusCommissionEmployee Inheritance Hierarchy using Protected Data

```
1  // Fig. 23.16: fig23_16.cpp
2  // Testing class BasePlusCommissionEmployee.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6  using std::fixed;
7
8  #include <iomanip>
9  using std::setprecision;
10
11 // BasePlusCommissionEmployee class definition
12 #include "BasePlusCommissionEmployee.h"
13
14 int main()
15 {
16    // instantiate BasePlusCommissionEmployee object
17    BasePlusCommissionEmployee
18       employee( "Bob", "Lewis", "333-33-3333", 5000, .04, 300 );
19
20    // set floating-point output formatting
21    cout << fixed << setprecision( 2 );
22
```

Example 4: a CommissionEmployee-BasePlusCommissionEmployee Inheritance Hierarchy using Protected Data

```
23    // get commission employee data
24    cout << "Employee information obtained by get functions: \n"
25       << "\nFirst name is " << employee.getFirstName()
26       << "\nLast name is " << employee.getLastName()
27       << "\nSocial security number is "
28       << employee.getSocialSecurityNumber()
29       << "\nGross sales is " << employee.getGrossSales()
30       << "\nCommission rate is " << employee.getCommissionRate()
31       << "\nBase salary is " << employee.getBaseSalary() << endl;
32
33    employee.setBaseSalary( 1000 ); // set base salary
34
35    cout << "\nUpdated employee information output by print function: \n"
36       << endl;
37    employee.print(); // display the new employee information
38
39    // display the employee's earnings
40    cout << "\n\nEmployee's earnings: $" << employee.earnings() << endl;
41
42    return 0;
43 } // end main
```

Example 4:
a CommissionEmployee-BasePlusCommissionEmployee Inheritance Hierarchy using Protected Data

```
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
```

Using protected data members

- Advantages
  - Derived class can modify values directly.
  - Avoid *set/get* function 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.
    - This is fragile (brittle) software.

Example 5: a 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* function to obtain values of data members.

Example 5: a CommissionEmployee-BasePlusCommissionEmployee Inheritance Hierarchy using Private Data

```
1  // Fig. 23.17: 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 std::string;
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
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
```

Example 5: a CommissionEmployee-BasePlusCommissionEmployee Inheritance Hierarchy using Private Data

```
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
```

Example 5: a CommissionEmployee-BasePlusCommissionEmployee Inheritance Hierarchy using Private Data

```
1  // Fig. 23.18: CommissionEmployee.cpp
2  // Class CommissionEmployee member-function definitions.
3  #include <iostream>
4  using std::cout;
5
6  #include "CommissionEmployee.h" // CommissionEmployee class definition
7
8  // constructor
9  CommissionEmployee::CommissionEmployee(
10    const string &first, const string &last, const string &ssn,
11    double sales, double rate )
12    : firstName( first ), lastName( last ), socialSecurityNumber( ssn )
13 {
14    setGrossSales( sales ); // validate and store gross sales
15    setCommissionRate( rate ); // validate and st
16 } // end CommissionEmployee constructor
17
18 // set first name
19 void CommissionEmployee::setFirstName( const string &first )
20 {
21    firstName = first; // should validate
22 } // end function setFirstName
23
24 // return first name
25 string CommissionEmployee::getFirstName() const
26 {
27    return firstName;
28 } // end function getFirstName
```

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

Example 5: a CommissionEmployee-BasePlusCommissionEmployee Inheritance Hierarchy using Private Data

```
29
30 // set last name
31 void CommissionEmployee::setLastName( const string &last )
32 {
33    lastName = last; // should validate
34 } // end function setLastName
35
36 // return last name
37 string CommissionEmployee::getLastName() const
38 {
39    return lastName;
40 } // end function getLastName
41
42 // set social security number
43 void CommissionEmployee::setSocialSecurityNumber( const string &ssn )
44 {
45    socialSecurityNumber = ssn; // should validate
46 } // end function setSocialSecurityNumber
47
48 // return social security number
49 string CommissionEmployee::getSocialSecurityNumber() const
50 {
51    return socialSecurityNumber;
52 } // end function getSocialSecurityNumber
53
54 // set gross sales amount
55 void CommissionEmployee::setGrossSales( double sales )
56 {
57    grossSales = ( sales < 0.0 ) ? 0.0 : sales;
58 } // end function setGrossSales
```

Example 5: a CommissionEmployee-BasePlusCommissionEmployee Inheritance Hierarchy using Private Data

```
59
60 // return gross sales amount
61 double CommissionEmployee::getGrossSales() const
62 {
63    return grossSales;
64 } // end function getGrossSales
65
66 // set commission rate
67 void CommissionEmployee::setCommissionRate( double rate )
68 {
69    commissionRate = ( rate > 0.0 && rate < 1.0 ) ? rate : 0.0;
70 } // end function setCommissionRate
71
72 // return commission rate
73 double CommissionEmployee::getCommissionRate() const
74 {
75    return commissionRate;
76 } // end function getCommissionRate
77
78 // calculate earnings
79 double CommissionEmployee::earnings() const
80 {
81    return getCommissionRate() * getGrossSales();
82 } // end function earnings
83
```

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

Example 5: a CommissionEmployee-BasePlusCommissionEmployee Inheritance Hierarchy using Private Data

```
84 // print CommissionEmployee object
85 void CommissionEmployee::print() const
86 {
87    cout << "commission employee: "
88       << getFirstName() << ' ' << getLastName()
89       << "\nsocial security number: " << getSocialSecurityNumber()
90       << "\ngross sales: " << getGrossSales()
91       << "\ncommission rate: " << getCommissionRate();
92 } // end function print
```

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

Example 5: a CommissionEmployee-BasePlusCommissionEmployee Inheritance Hierarchy using Private Data

```
1  // Fig. 23.19: 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  using std::string;
9
10 #include "CommissionEmployee.h" // CommissionEmployee class declaration
11
12 class BasePlusCommissionEmployee : public CommissionEmployee
13 {
14 public:
15    BasePlusCommissionEmployee( const string &, const string &,
16       const string &, double = 0.0, double = 0.0, double = 0.0 );
17
18    void setBaseSalary( double ); // set base salary
19    double getBaseSalary() const; // return base salary
20
21    double earnings() const; // calculate earnings
22    void print() const; // print BasePlusCommissionEmployee object
23 private:
24    double baseSalary; // base salary
25 }; // end class BasePlusCommissionEmployee
26
27 #endif
```

Example 5: a CommissionEmployee-BasePlusCommissionEmployee Inheritance Hierarchy using Private Data

```
1  // Fig. 23.20: BasePlusCommissionEmployee.cpp
2  // Class BasePlusCommissionEmployee member-function definitions.
3  #include <iostream>
4  using std::cout;
5
6  // BasePlusCommissionEmployee class definition
7  #include "BasePlusCommissionEmployee.h"
8
9  // constructor
10 BasePlusCommissionEmployee::BasePlusCommissionEmployee(
11    const string &first, const string &last, const string &ssn,
12    double sales, double rate, double salary )
13    // explicitly call base-class constructor
14    : CommissionEmployee( first, last, ssn, sales, rate )
15 {
16    setBaseSalary( salary ); // validate and store base salary
17 } // end BasePlusCommissionEmployee constructor
18
19 // set base salary
20 void BasePlusCommissionEmployee::setBaseSalary( double salary )
21 {
22    baseSalary = ( salary < 0.0 ) ? 0.0 : salary;
23 } // end function setBaseSalary
24
25 // return base salary
26 double BasePlusCommissionEmployee::getBaseSalary() const
27 {
28    return baseSalary;
29 } // end function getBaseSalary
```

Example 5: a CommissionEmployee-BasePlusCommissionEmployee Inheritance Hierarchy using Private Data

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

Invoke base class's **earnings** function

Invoke base class's **print** function

Example 5: a CommissionEmployee-BasePlusCommissionEmployee Inheritance Hierarchy using Private Data

```
1  // Fig. 23.21: fig23_21.cpp
2  // Testing class BasePlusCommissionEmployee.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6  using std::fixed;
7
8  #include <iomanip>
9  using std::setprecision;
10
11 // BasePlusCommissionEmployee class definition
12 #include "BasePlusCommissionEmployee.h"
13
```

Example 5: a CommissionEmployee-BasePlusCommissionEmployee Inheritance Hierarchy using Private Data

```
14 int main()
15 {
16    // instantiate BasePlusCommissionEmployee object
17    BasePlusCommissionEmployee
18       employee( "Bob", "Lewis", "333-33-3333", 5000, .04, 300 );
19
20    // set floating-point output formatting
21    cout << fixed << setprecision( 2 );
22
23    // get commission employee data
24    cout << "Employee information obtained by get functions: \n"
25       << "\nFirst name is " << employee.getFirstName()
26       << "\nLast name is " << employee.getLastName()
27       << "\nSocial security number is "
28       << employee.getSocialSecurityNumber()
29       << "\nGross sales is " << employee.getGrossSales()
30       << "\nCommission rate is " << employee.getCommissionRate()
31       << "\nBase salary is " << employee.getBaseSalary() << endl;
32
33    employee.setBaseSalary( 1000 ); // set base salary
34
35    cout << "\nUpdated employee information output by print function: \n"
36       << endl;
37    employee.print(); // display the new employee informat
38
39    // display the employee's earnings
40    cout << "\n\nEmployee's earnings: $" << employee.earnings() << endl;
41
42    return 0;
43 } // 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`**

Example 5:
a CommissionEmployee-BasePlusCommissionEmployee Inheritance Hierarchy using Private Data

```
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
```

23.5 Constructors and Destructors in Derived Classes

- Instantiating derived-class object
  - Chain of constructor calls
    - Derived-class constructor invokes base class constructor either implicitly (via a base-class member initializer) or explicitly (by calling the base classes default constructor).
- Base of inheritance hierarchy
  - The last constructor called in an inheritance chain is at the base of the hierarchy and this constructor is the first constructor body to finish executing.

Example: `CommissionEmployee/BasePlusCommissionEmployee` hierarchy

- `CommissionEmployee` constructor called last.
- `CommissionEmployee` constructor body executes first and initializes private data members.
- Each base-class constructor initializes its data members that are inherited by derived class.

Software Engineering Observation 23.7

- 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.

Constructors and Destructors in Derived Classes

- Destroying derived-class objects
  - Chain of destructor calls
    - Reverse order of constructor chain
    - Destructor of derived-class called first.
    - Destructor of next base class up hierarchy is called next.
    - This continues up hierarchy until the final base class is reached.
      - After final base-class destructor, the object is removed from memory.
- Base-class constructors, destructors, and overloaded assignment operators are not inherited by derived classes.

Software Engineering Observation 23.8

- 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.

Software Engineering Observation 23.8

- Destructors for derived-class objects are called in the reverse of the order in which their corresponding constructors are called.

Constructors and Destructors in Derived Classes

```
1  // Fig. 23.22: CommissionEmployee.h
2  // CommissionEmployee class definition represents a commission employee.
3  #ifndef COMMISSION_H
4  #define COMMISSION_H
5
6  #include <string> // C++ standard string class
7  using std::string;
8
9  class CommissionEmployee
10 {
11 public:
12    CommissionEmployee( const string &, const string &, const string &,
13       double = 0.0, double = 0.0 );
14    ~CommissionEmployee(); // destructor
15
16    void setFirstName( const string & ); // set first name
17    string getFirstName() const; // return first name
18
19    void setLastName( const string & ); // set last name
20    string getLastName() const; // return last name
21
22    void setSocialSecurityNumber( const string & ); // set SSN
23    string getSocialSecurityNumber() const; // return SSN
24
25    void setGrossSales( double ); // set gross sales amount
26    double getGrossSales() const; // return gross sales amount
27
28    void setCommissionRate( double ); // set commission rate
29    double getCommissionRate() const; // return commission rate
```

CommissionEmployee destructor

Constructors and Destructors in Derived Classes

```
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
```

Constructors and Destructors in Derived Classes

```
1  // Fig. 23.23: CommissionEmployee.cpp
2  // Class CommissionEmployee member-function definitions.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  #include "CommissionEmployee.h" // CommissionEmployee class definition
8
9  // constructor
10 CommissionEmployee::CommissionEmployee(
11    const string &first, const string &last, const string &ssn,
12    double sales, double rate )
13    : firstName( first ), lastName( last ), socialSecurityNumber( ssn )
14 {
15    setGrossSales( sales ); // validate and store gross sales
16    setCommissionRate( rate ); // validate and store commission rate
17
18    cout << "CommissionEmployee constructor: " << endl;
19    print();
20    cout << "\n\n";
21 } // end CommissionEmployee constructor
22
23 // destructor
24 CommissionEmployee::~CommissionEmployee()
25 {
26    cout << "CommissionEmployee destructor: " << endl;
27    print();
28    cout << "\n\n";
29 } // end CommissionEmployee destructor
```

Constructor and destructor output messages to demonstrate function call order

Constructors and Destructors in Derived Classes

```
30
31 // set first name
32 void CommissionEmployee::setFirstName( const string &first )
33 {
34    firstName = first; // should validate
35 } // end function setFirstName
36
37 // return first name
38 string CommissionEmployee::getFirstName() const
39 {
40    return firstName;
41 } // end function getFirstName
42
43 // set last name
44 void CommissionEmployee::setLastName( const string &last )
45 {
46    lastName = last; // should validate
47 } // end function setLastName
48
49 // return last name
50 string CommissionEmployee::getLastName() const
51 {
52    return lastName;
53 } // end function getLastName
54
55 // set social security number
56 void CommissionEmployee::setSocialSecurityNumber( const string &ssn )
57 {
58    socialSecurityNumber = ssn; // should validate
59 } // end function setSocialSecurityNumber
```

Constructors and Destructors in Derived Classes

```
60
61 // return social security number
62 string CommissionEmployee::getSocialSecurityNumber() const
63 {
64    return socialSecurityNumber;
65 } // end function getSocialSecurityNumber
66
67 // set gross sales amount
68 void CommissionEmployee::setGrossSales( double sales )
69 {
70    grossSales = ( sales < 0.0 ) ? 0.0 : sales;
71 } // end function setGrossSales
72
73 // return gross sales amount
74 double CommissionEmployee::getGrossSales() const
75 {
76    return grossSales;
77 } // end function getGrossSales
78
79 // set commission rate
80 void CommissionEmployee::setCommissionRate( double rate )
81 {
82    commissionRate = ( rate > 0.0 && rate < 1.0 ) ? rate : 0.0;
83 } // end function setCommissionRate
84
85 // return commission rate
86 double CommissionEmployee::getCommissionRate() const
87 {
88    return commissionRate;
89 } // end function getCommissionRate
```

Constructors and Destructors in Derived Classes

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

Constructors and Destructors in Derived Classes

```
1  // Fig. 23.24: BasePlusCommissionEmployee.h
2  // BasePlusCommissionEmployee class derived from class
3  // CommissionEmployee.
4  #ifndef BASEPLUS_H
5  #define BASEPLUS_H
6
7  #include <string> // C++ standard string class
8  using std::string;
9
10 #include "CommissionEmployee.h" // CommissionEmployee class declaration
11
12 class BasePlusCommissionEmployee : public CommissionEmployee
13 {
14 public:
15    BasePlusCommissionEmployee( const string &, const string &,
16       const string &, double = 0.0, double = 0.0, double = 0.0 );
17    ~BasePlusCommissionEmployee(); // destructor
18
19    void setBaseSalary( double ); // set base salary
20    double getBaseSalary() const; // return base salary
21
22    double earnings() const; // calculate earnings
23    void print() const; // print BasePlusCommissionEmployee object
24 private:
25    double baseSalary; // base salary
26 }; // end class BasePlusCommissionEmployee
27
28 #endif
```

BasePlusCommissionEmployee destructor

Constructors and Destructors in Derived Classes

```
1  // Fig. 23.25: BasePlusCommissionEmployee.cpp
2  // Class BasePlusCommissionEmployee member-function definitions.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  // BasePlusCommissionEmployee class definition
8  #include "BasePlusCommissionEmployee.h"
9
10 // constructor
11 BasePlusCommissionEmployee::BasePlusCommissionEmployee(
12    const string &first, const string &last, const string &ssn,
13    double sales, double rate, double salary )
14    // explicitly call base-class constructor
15    : CommissionEmployee( first, last, ssn, sales, rate )
16 {
17    setBaseSalary( salary ); // validate and store base salary
18
19    cout << "BasePlusCommissionEmployee constructor: " << endl;
20    print();
21    cout << "\n\n";
22 } // end BasePlusCommissionEmployee constructor
23
24 // destructor
25 BasePlusCommissionEmployee::~BasePlusCommissionEmployee()
26 {
27    cout << "BasePlusCommissionEmployee destructor: " << endl;
28    print();
29    cout << "\n\n";
30 } // end BasePlusCommissionEmployee destructor
```

Constructor and destructor output messages to demonstrate function call order

Constructors and Destructors in Derived Classes

```
31
32 // set base salary
33 void BasePlusCommissionEmployee::setBaseSalary( double salary )
34 {
35    baseSalary = ( salary < 0.0 ) ? 0.0 : salary;
36 } // end function setBaseSalary
37
38 // return base salary
39 double BasePlusCommissionEmployee::getBaseSalary() const
40 {
41    return baseSalary;
42 } // end function getBaseSalary
43
44 // calculate earnings
45 double BasePlusCommissionEmployee::earnings() const
46 {
47    return getBaseSalary() + CommissionEmployee::earnings();
48 } // end function earnings
49
50 // print BasePlusCommissionEmployee object
51 void BasePlusCommissionEmployee::print() const
52 {
53    cout << "base-salaried ";
54
55    // invoke CommissionEmployee's print function
56    CommissionEmployee::print();
57
58    cout << "\nbase salary: " << getBaseSalary();
59 } // end function print
```

Constructors and Destructors in Derived Classes

```
1  // Fig. 23.26: fig23_26.cpp
2  // Display order in which base-class and derived-class constructors
3  // and destructors are called.
4  #include <iostream>
5  using std::cout;
6  using std::endl;
7  using std::fixed;
8
9  #include <iomanip>
10 using std::setprecision;
11
12 // BasePlusCommissionEmployee class definition
13 #include "BasePlusCommissionEmployee.h"
```

Constructors and Destructors in Derived Classes


```
14
15 int main()
16 {
17    // set floating-point output formatting
18    cout << fixed << setprecision( 2 );
19
20    { // begin new scope
21       CommissionEmployee employee1(
22          "Bob", "Lewis", "333-33-3333", 5000, .04 );
23    } // end scope
24
25    cout << endl;
26    BasePlusCommissionEmployee
27       employee2( "Lisa", "Jones", "555-55-5555", 2000, .06, 800 );
28
29    cout << endl;
30    BasePlusCommissionEmployee
31       employee3( "Mark", "Sands", "888-88-8888", 8000, .15, 2000 );
32    cout << endl;
33    return 0;
34 } // end main
```

CommissionEmployee object goes in and out of scope immediately

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

Constructors and Destructors in Derived Classes

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

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
```


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

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

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

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

Constructors and Destructors in Derived Classes

(... continued from bottom of previous slide)

```
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

Summary

- Base Classes and Derived Classes
- Five Examples of Base Class and Derived Class Relationships
  - Focused on the distinctions in using public, private and protected data members and public get/set member functions
- Order of execution of constructors and destructors in inheritance hierarchy chains.