12

Object-Oriented Programming: Inheritance

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

—Johann Kasper Lavater

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

—Bertrand Russell

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

—Augustine Birrell

Save base authority from others' books.

—William Shakespeare

OBJECTIVES

In this chapter you'll learn:

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

Assignment Checklist

Name: _____ **Date:** _____

Section: _____

Exercises	Assigned: Circle assignments	Date Due
Prelab Activities		
Matching	YES NO	
Fill in the Blank	11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21	
Short Answer	22, 23, 24, 25, 26, 27, 28	
Correct the Code	29, 30, 31, 32	
Lab Exercises		
Lab Exercise 1 — `Account` Hierarchy	YES NO	
Lab Exercise 2 — Composition	YES NO	
Debugging	YES NO	
Labs Provided by Instructor		
1.		
2.		
3.		
Postlab Activities		
Coding Exercises	1, 2, 3, 4	
Programming Challenge	1	

Prelab Activities

Matching

Name: _____ **Date:** _____

Section: _____

After reading Chapter 12 of *C++ How to Program, Seventh Edition*, answer the given questions. These questions are intended to test and reinforce your understanding of key concepts and may be done either before the lab or during the lab.

For each term in the column on the left, write the corresponding letter for the description that best matches it from the column on the right.

	Term	Description
___	1. Inheritance	a) Class from which others are derived.
___	2. Derived class	b) Deriving from more than one base class.
___	3. "Has a" relationship	c) Class that is created by inheriting from an existing class.
___	4. "Is a" relationship	d) Inheritance.
___	5. Single inheritance	e) Passes arguments to the base-class constructor.
___	6. Base class	f) Base class that is not listed explicitly in the derived class's definition.
___	7. Indirect base class	g) Composition.
___	8. Base-class initializer	h) Deriving from only one base class.
___	9. Multiple inheritance	i) New classes are created from existing classes.

Prelab Activities

Name:

Fill in the Blank

Name: _____ **Date:** _____

Section: _____

Fill in the blank for each of the following statements:

10. The programmer can designate that a new class is to _____ the data members and member functions of a previously defined base class.

11. A base class's _____ may be accessed by members and `friend`s of the base class and by members and `friend`s of the derived class.

12. With _____ inheritance, a class is derived from only one base class.

13. A derived class cannot access the _____ members of its base class; allowing such access would violate the _____ of the base class.

14. A derived class's constructor always calls the constructor of its _____ first.

15. Destructors are called in _____ order of constructor calls.

16. The three forms of inheritance are _____, _____ and _____.

17. With `private` inheritance, `public` and `protected` members of the base class become _____ members of the derived class.

18. A(n) _____ base class is not explicitly listed in the derived-class definition; rather, it is inherited from several levels up the class hierarchy tree.

19. A pointer to a derived-class object can be cast implicitly to a(n) _____ pointer.

Prelab Activities

Name:

Short Answer

Name: _____ **Date:** _____

Section: _____

In the space provided, answer each of the given questions. Your answers should be as concise as possible; aim for two or three sentences.

20. What is inheritance?

21. What is `protected` access?

22. What is the difference between single and multiple inheritance?

Prelab Activities

Name:

Short Answer

23. What is the difference between direct and indirect base classes?

24. What is the sequence of events that takes place when a derived-class object is destroyed? (That is, in what order are the destructors invoked, and why?)

25. What are the primary differences between `public`, `private` and `protected` inheritance?

26. What is meant by redefining a base-class member? How does this process differ from function overloading?

Prelab Activities

Name:

Correct the Code

Name: _____ **Date:** _____

Section: _____

For each of the given program segments, determine if there is an error in the code. If there is an error, specify whether it is a logic or compilation error, circle the error in the program, and write the corrected code in the space provided after each problem. If the code does not contain an error, write "no error." [*Note:* It is possible that a program segment may contain multiple errors.]

27. Class X inherits from class Y.

```
1  #include <iostream>
2  using namespace std;
3
4  // class Y definition
5  class Y
6  {
7  public:
8     Y(); // default constructor
9     ~Y(); // destructor
10 private:
11    int data;
12 }; // end class Y
13
14 // class X definition
15 class X ; public Y
16 {
17 public:
18    // function print
19    void print() const
20    {
21       cout << data;
22    } // end function print
23 }; // end class X
```

Your answer:

Prelab Activities

Name:

Correct the Code

28. The following code should construct a `Derived` object.

```
1  #include <iostream>
2  using namespace std;
3
4  // class Base definition
5  class Base
6  {
7  private:
8     // constructor
9     Base( int b )
10    {
11       cout << b;
12    } // end class Base constructor
13 }; // end class Base
14
15 // class Derived definition
16 class Derived : public Base
17 {
18    // constructor calls base-class constructor
19    Derived( int a )
20       : Base( a )
21    {
22       // empty
23    } // end class Derived constructor
24 }; // end class Derived
25
26 int main()
27 {
28    Derived d( 5 );
29 } // end main
```

Your answer:

Prelab Activities

Name:

Correct the Code

29. The following code creates an object of type `B`. Class `B` inherits from class `A`.

```
1  #include <iostream>
2  using namespace std;
3
4  // class A definition
5  class A
6  {
7  public:
8     // constructor
9     A( int a )
10    {
11       value = a;
12    } // end class A constructor
13
14    // return value
15    int getValue() const
16    {
17       return value;
18    } // end function getValue
19 private:
20    int value;
21 }; // end class A
22
23 // class B definition
24 class B
25 {
26 public:
27    // constructor
28    B( int b )
29       : A( b )
30    {
31       // empty
32    } // end class B constructor
33 }; // end class B
34
35 int main()
36 {
37    B object( 50 );
38    cout << object.getValue();
39 } // end main
```

Your answer:

30. The following code creates an object of type `Y`. Class `Y` inherits from class `X`.

```
#include <iostream>
using namespace std;

// class X definition
class X
{
public:
   // constructor
   X()
   {
      cout << "X constructed!";
   } // end class X constructor
}; // end class X

// class Y definition
class Y
{
public:
   // redefine inherited constructor
   X()
   {
      cout << "Y created, not X!";
   } // end class Y constructor
}; // end class Y

int main()
{
   Y yObject();
} // end main
```

Your answer:

Lab Exercises

Name:

Lab Exercise 1 — Account Hierarchy

Lab Exercises

Lab Exercise 1 — Account Hierarchy

Name: _____ **Date:** _____

Section: _____

This problem is intended to be solved in a closed-lab session with a teaching assistant or instructor present. The problem is divided into five parts:

1. Lab Objectives
2. Description of the Problem
3. Sample Output
4. Program Template (Fig. L 12.1–Fig. L 12.7)
5. Problem-Solving Tips

The program template represents a complete working C++ program, with one or more key lines of code replaced with comments. Read the problem description and examine the sample output; then study the template code. Using the problem-solving tips as a guide, replace the `/* */` comments with C++ code. Compile and execute the program. Compare your output with the sample output provided. The source code for the template is available from the Companion Website for *C++ How to Program, Seventh Edition* at `www.pearsonhighered.com/deitel/`.

Lab Objectives

This lab was designed to reinforce programming concepts from Chapter 12 of *C++ How To Program, Seventh Edition*. In this lab, you will practice:

- Using inheritance to create an account hierarchy that includes a `Account` class, a `SavingsAccount` class and a `CheckingAccount` class.
- Using `private` data members to limit access to data members.
- Redefining base-class member functions in a derived class.

Description of the Problem

Create an inheritance hierarchy that a bank might use to represent customers' bank accounts. All customers at this bank can deposit (i.e., credit) money into their accounts and withdraw (i.e., debit) money from their accounts. More specific types of accounts also exist. Savings accounts, for instance, earn interest on the money they hold. Checking accounts, on the other hand, charge a fee per transaction (i.e., credit or debit).

Create an inheritance hierarchy containing base class `Account` and derived classes `SavingsAccount` and `CheckingAccount` that inherit from class `Account`. Base class `Account` should include one data member of type `double` to represent the account balance. The class should provide a constructor that receives an initial balance and uses it to initialize the data member. The constructor should validate the initial balance to ensure that it is greater than or equal to `0.0`. If not, the balance should be set to `0.0` and the constructor should display an error message, indicating that the initial balance was invalid. The class should provide three member functions. Member function `credit` should add an amount to the current balance. Member function `debit` should withdraw money from the `Account` and ensure that the debit amount does not exceed the `Account`'s balance. If it

Lab Exercises

Name:

Lab Exercise 1 — Account Hierarchy

does, the balance should be left unchanged and the function should print the message `"Debit amount exceeded account balance."` Member function `getBalance` should return the current balance.

Derived class `SavingsAccount` should inherit the functionality of an `Account`, but also include a data member of type `double` indicating the interest rate (percentage) assigned to the `Account`. `SavingsAccount`'s constructor should receive the initial balance, as well as an initial value for the `SavingsAccount`'s interest rate. `SavingsAccount` should provide a public member function `calculateInterest` that returns a `double` indicating the amount of interest earned by an account. Member function `calculateInterest` should determine this amount by multiplying the interest rate by the account balance. [*Note:* `SavingsAccount` should inherit member functions `credit` and `debit` as is without redefining them.]

Derived class `CheckingAccount` should inherit from base class `Account` and include an additional data member of type `double` that represents the fee charged per transaction. `CheckingAccount`'s constructor should receive the initial balance, as well as a parameter indicating a fee amount. Class `CheckingAccount` should redefine member functions `credit` and `debit` so that they subtract the fee from the account balance whenever either transaction is performed successfully. `CheckingAccount`'s versions of these functions should invoke the base-class `Account` version to perform the updates to an account balance. `CheckingAccount`'s debit function should charge a fee only if money is actually withdrawn (i.e., the debit amount does not exceed the account balance). [*Hint:* Define `Account`'s `debit` function so that it returns a `bool` indicating whether money was withdrawn. Then use the return value to determine whether a fee should be charged.]

After defining the classes in this hierarchy, write a program that creates objects of each class and tests their member functions. Add interest to the `SavingsAccount` object by first invoking its `calculateInterest` function, then passing the returned interest amount to the object's `credit` function.

Sample Output

```
account1 balance: $50.00
account2 balance: $25.00
account3 balance: $80.00

Attempting to debit $25.00 from account1.

Attempting to debit $30.00 from account2.
Debit amount exceeded account balance.

Attempting to debit $40.00 from account3.
$1.00 transaction fee charged.

account1 balance: $25.00
account2 balance: $25.00
account3 balance: $39.00

Crediting $40.00 to account1.

Crediting $65.00 to account2.

Crediting $20.00 to account3.
$1.00 transaction fee charged.

account1 balance: $65.00
account2 balance: $90.00
account3 balance: $58.00

Adding $2.70 interest to account2.

New account2 balance: $92.70
```

Lab Exercises

Name:

Lab Exercise 1 — Account Hierarchy

Template

```
1  // Lab 1: Account.h
2  // Definition of Account class.
3  #ifndef ACCOUNT_H
4  #define ACCOUNT_H
5
6  class Account
7  {
8  public:
9     Account( double ); // constructor initializes balance
10    void credit( double ); // add an amount to the account balance
11    bool debit( double ); // subtract an amount from the account balance
12    void setBalance( double ); // sets the account balance
13    double getBalance(); // return the account balance
14 private:
15    double balance; // data member that stores the balance
16 }; // end class Account
17
18 #endif
```

Fig. L 12.1 | Contents of `Account.h`.

```
1  // Lab 1: Account.cpp
2  // Member-function definitions for class Account.
3  #include <iostream>
4  using namespace std;
5
6  #include "Account.h" // include definition of class Account
7
8  // Account constructor initializes data member balance
9  Account::Account( double initialBalance )
10 {
11    // if initialBalance is greater than or equal to 0.0, set this value
12    // as the balance of the Account
13    if ( initialBalance >= 0.0 )
14       balance = initialBalance;
15    else // otherwise, output message and set balance to 0.0
16    {
17       cout << "Error: Initial balance cannot be negative." << endl;
18       balance = 0.0;
19    } // end if...else
20 } // end Account constructor
21
22 // credit (add) an amount to the account balance
23 void Account::credit( double amount )
24 {
25    balance = balance + amount; // add amount to balance
26 } // end function credit
27
28 // debit (subtract) an amount from the account balance
29 // return bool indicating whether money was debited
30 bool Account::debit( double amount )
31 {

32    if ( amount > balance ) // debit amount exceeds balance
```

Fig. L 12.2 | Contents of `Account.cpp`. (Part 1 of 2.)

Lab Exercises

Name:

Lab Exercise 1 — Account Hierarchy

```
33      {
34         cout << "Debit amount exceeded account balance." << endl;
35         return false;
36      } // end if
37      else // debit amount does not exceed balance
38      {
39         balance = balance - amount;
40         return true;
41      } // end else
42   } // end function debit
43
44   // set the account balance
45   void Account::setBalance( double newBalance )
46   {
47      balance = newBalance;
48   } // end function setBalance
49
50   // return the account balance
51   double Account::getBalance()
52   {
53      return balance;
54   } // end function getBalance
```

Fig. L 12.2 | Contents of `Account.cpp`. (Part 2 of 2.)

```
1   // Lab 1: SavingsAccount.h
2   // Definition of SavingsAccount class.
3   #ifndef SAVINGS_H
4   #define SAVINGS_H
5
6   /* Write a directive to include the Account header file */
7
8   /* Write a line to have class SavingsAccount inherit publicly from Account */
9   {
10  public:
11     // constructor initializes balance and interest rate
12     /* Declare a two-parameter constructor for SavingsAccount */
13
14     /* Declare member function calculateInterest */
15  private:
16     /* Declare data member interestRate */
17  }; // end class SavingsAccount
18
19  #endif
```

Fig. L 12.3 | Contents of `SavingsAccount.h`.

```
// Lab 1: SavingsAccount.cpp
// Member-function definitions for class SavingsAccount.

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

// constructor initializes balance and interest rate
/* Write the SavingsAccount constructor to call the Account constructor
   and validate and set the interest rate value */

// return the amount of interest earned
/* Write the calculateInterest member function to return the
   interest based on the current balance and interest rate */
```

Fig. L 12.4 | Contents of `SavingsAccount.cpp`.

```
// Lab 1: CheckingAccount.h
// Definition of CheckingAccount class.
#ifndef CHECKING_H
#define CHECKING_H

/* Write a directive to include the Account header file */

/* Write a line to have class CheckingAccount inherit publicly from Account */
{
public:
   // constructor initializes balance and transaction fee
   /* Declare a two-argument constructor for CheckingAccount */

   /* Redeclare member function credit, which will be redefined */
   /* Redeclare member function debit, which will be redefined */
private:
   /* Declare data member transactionFee */

   // utility function to charge fee
   /* Declare member function chargeFee */
}; // end class CheckingAccount

#endif
```

Fig. L 12.5 | Contents of `CheckingAccount.h`.

```
// Lab 1: CheckingAccount.cpp
// Member-function definitions for class CheckingAccount.
#include <iostream>
using namespace std;

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

// constructor initializes balance and transaction fee
/* Write the CheckingAccount constructor to call the Account constructor
   and validate and set the transaction fee value */


// credit (add) an amount to the account balance and charge fee
/* Write the credit member function to call Account's credit function
   and then charge a fee */

// debit (subtract) an amount from the account balance and charge fee
/* Write the debit member function to call Account's debit function
   and then charge a fee if it returned true*/

// subtract transaction fee
/* Write the chargeFee member function to subtract transactionFee
```

Fig. L 12.6 | Contents of `CheckingAccount.cpp` (Part 1 of 2.)

Lab Exercises

Name:

Lab Exercise 1 — Account Hierarchy

```
22     from the current balance and display a message */
```

Fig. L 12.6 | Contents of `CheckingAccount.cpp`. (Part 2 of 2.)

```
1  // Lab 1: bankAccounts.cpp
2  // Test program for Account hierarchy.
3  #include <iostream>
4  #include <iomanip>
5  using namespace std;
6
7  #include "Account.h" // Account class definition
8  #include "SavingsAccount.h" // SavingsAccount class definition
9  #include "CheckingAccount.h" // CheckingAccount class definition
10
11 int main()
12 {
13    Account account1( 50.0 ); // create Account object
14    SavingsAccount account2( 25.0, .03 ); // create SavingsAccount object
15    CheckingAccount account3( 80.0, 1.0 ); // create CheckingAccount object
16
17    cout << fixed << setprecision( 2 );
18
19    // display initial balance of each object
20    cout << "account1 balance: $" << account1.getBalance() << endl;
21    cout << "account2 balance: $" << account2.getBalance() << endl;
22    cout << "account3 balance: $" << account3.getBalance() << endl;
23
24    cout << "\nAttempting to debit $25.00 from account1." << endl;
25    account1.debit( 25.0 ); // try to debit $25.00 from account1
26    cout << "\nAttempting to debit $30.00 from account2." << endl;
27    account2.debit( 30.0 ); // try to debit $30.00 from account2
28    cout << "\nAttempting to debit $40.00 from account3." << endl;
29    account3.debit( 40.0 ); // try to debit $40.00 from account3
30
31    // display balances
32    cout << "\naccount1 balance: $" << account1.getBalance() << endl;
33    cout << "account2 balance: $" << account2.getBalance() << endl;
34    cout << "account3 balance: $" << account3.getBalance() << endl;
35
36    cout << "\nCrediting $40.00 to account1." << endl;
37    account1.credit( 40.0 ); // credit $40.00 to account1

38    cout << "\nCrediting $65.00 to account2." << endl;
39    account2.credit( 65.0 ); // credit $65.00 to account2
40    cout << "\nCrediting $20.00 to account3." << endl;
41    account3.credit( 20.0 ); // credit $20.00 to account3
42
43    // display balances
44    cout << "\naccount1 balance: $" << account1.getBalance() << endl;
45    cout << "account2 balance: $" << account2.getBalance() << endl;
46    cout << "account3 balance: $" << account3.getBalance() << endl;
47
48    // add interest to SavingsAccount object account2
49    /* Declare a variable interestEarned and assign it the interest
50       account2 should earn */
51    cout << "\nAdding $" << interestEarned << " interest to account2."
```

Fig. L 12.7 | Contents of `bankAccount.cpp`. (Part 1 of 2.)

Lab Exercises

Name:

Lab Exercise 1 — Account Hierarchy

```
52          << endl;
53       /* Write a statement to credit the interest to account2's balance */
54
55       cout << "\nNew account2 balance: $" << account2.getBalance() << endl;
56    } // end main
```

Fig. L 12.7 | Contents of `bankAccount.cpp`. (Part 2 of 2.)

Problem-Solving Tips

1. Each derived class constructor, `SavingsAccount` and `CheckingAccount`, should call the `Account` constructor explicitly.
2. Do not use the `debit` member function inside the `chargeFee` member function, because the `debit` member function would then call the `chargeFee` member function, leading to infinite recursion. Instead use the inherited *get* and *set* functions for the account balance.

Lab Exercises

Name:

Lab Exercise 1 — Account Hierarchy

Lab Exercises

Name:

Lab Exercise 2 — Composition

Name: _____ **Date:** _____

Section: _____

This problem is intended to be solved in a closed-lab session with a teaching assistant or instructor present. The problem is divided into six parts:

1. Lab Objectives
2. Description of the Problem
3. Sample Output
4. Program Template (Fig. L 12.8–Fig. L 12.12)
5. Problem-Solving Tips
6. Follow-Up Question and Activity

The program template represents a complete working C++ program, with one or more key lines of code replaced with comments. Read the problem description and examine the sample output; then study the template code. Using the problem-solving tips as a guide, replace the `/* */` comments with C++ code. Compile and execute the program. Compare your output with the sample output provided. Then answer the follow-up question. The source code for the template is available from the Companion Website for *C++ How to Program, Seventh Edition* at `www.pearsonhighered.com/deitel/`.

Lab Objectives

This lab was designed to reinforce programming concepts from Chapter 12 of *C++ How To Program, Seventh Edition*. In this lab, you will practice

- Using composition to incorporate one class's members into another class.

The follow-up question and activity also will give you practice:

- Comparing inheritance and composition.

Description of the Problem

Many programs written with inheritance could be written with composition instead, and vice versa. Rewrite class `BasePlusCommissionEmployee` of the `CommissionEmployee–BasePlusCommissionEmployee` hierarchy to use composition rather than inheritance.

Lab Exercises

Name:

Lab Exercise 2 — Composition

Sample Output

```
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
```

Template

```
// Lab 2: CommissionEmployee.h
// CommissionEmployee class definition represents a commission employee.
#ifndef COMMISSION_H
#define COMMISSION_H

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

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

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

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

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

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

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

   double earnings() const; // calculate earnings
   void print() const; // print CommissionEmployee object
private:
   string firstName;
   string lastName;
```

Fig. L 12.8 | Contents of CommissionEmployee.h (Part 1 of 2).

Lab Exercises

Name:

Lab Exercise 2 — Composition

```
   string socialSecurityNumber;
   double grossSales; // gross weekly sales
   double commissionRate; // commission percentage
}; // end class CommissionEmployee

#endif
```

Fig. L 12.8 | Contents of `CommissionEmployee.h`. (Part 2 of 2.)

```
// Lab 2: CommissionEmployee.cpp
// Class CommissionEmployee member-function definitions.
#include <iostream>
using namespace std;

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

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

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

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

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

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

```

Fig. L 12.9 | Contents of `CommissionEmployee.cpp`. (Part 1 of 2.)

```
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
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
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
```

Fig. L 12.9 | Contents of `CommissionEmployee.cpp`. (Part 2 of 2.)

Lab Exercises

Name:

Lab Exercise 2 — Composition

```
// Lab 2: BasePlusCommissionEmployee.h
// BasePlusCommissionEmployee class using composition.
#ifndef BASEPLUS_H
#define BASEPLUS_H

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

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

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

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

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

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

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

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

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

   double earnings() const; // calculate earnings
   void print() const; // print BasePlusCommissionEmployee object
private:
   double baseSalary; // base salary
   /* Write a declaration for a CommissionEmployee
      data member */
}; // end class BasePlusCommissionEmployee

#endif
```

Fig. L 12.10 | Contents of `BasePlusCommissionEmployee.h`.

```
// Lab 2: BasePlusCommissionEmployee.cpp
// Member-function definitions of class BasePlusCommissionEmployee
// using composition.
#include <iostream>
using namespace std;

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

```

Fig. L 12.11 | Contents of `BasePlusCommissionEmployee.cpp` (Part 1 of 3.)

Lab Exercises

Name:

Lab Exercise 2 — Composition

```
10  // constructor
11  BasePlusCommissionEmployee::BasePlusCommissionEmployee(
12     const string &first, const string &last, const string &ssn,
13     double sales, double rate, double salary )
14     // initialize composed object
15     : /* Initialize the commissionEmployee data member,
16          pass (first, last, ssn, sales, rate) to its constructor */
17  {
18     setBaseSalary( salary ); // validate and store base salary
19  } // end BasePlusCommissionEmployee constructor
20
21  // set commission employee's first name
22  void BasePlusCommissionEmployee::setFirstName( const string &first )
23  {
24     /* Call commissionEmployee's setFirstName function */
25  } // end function setFirstName
26
27  // return commission employee's first name
28  string BasePlusCommissionEmployee::getFirstName() const
29  {
30     /* Call commissionEmployee's getFirstName function */
31  } // end function getFirstName
32
33  // set commission employee's last name
34  void BasePlusCommissionEmployee::setLastName( const string &last )
35  {
36     /* Call commissionEmployee's setLastName function */
37  } // end function setLastName
38
39  // return commission employee's last name
40  string BasePlusCommissionEmployee::getLastName() const
41  {
42     /* Call commissionEmployee's getLastName function */
43  } // end function getLastName
44
45  // set commission employee's social security number
46  void BasePlusCommissionEmployee::setSocialSecurityNumber(
47     const string &ssn )
48  {
49     /* Call commissionEmployee's setSocialSecurity function */
50  } // end function setSocialSecurityNumber
51
52  // return commission employee's social security number
53  string BasePlusCommissionEmployee::getSocialSecurityNumber() const
54  {
55     /* Call commissionEmployee's getSocialSecurity function */
56  } // end function getSocialSecurityNumber
57
58  // set commission employee's gross sales amount
59  void BasePlusCommissionEmployee::setGrossSales( double sales )
60  {
61     /* Call commissionEmployee's setGrossSales function */
62  } // end function setGrossSales
63
```

Fig. L 12.11 | Contents of `BasePlusCommissionEmployee.cpp`. (Part 2 of 3.)

Lab Exercises

Name:

Lab Exercise 2 — Composition

```
64  // return commission employee's gross sales amount
65  double BasePlusCommissionEmployee::getGrossSales() const
66  {
67     /* Call commissionEmployee's getGrossSales function */
68  } // end function getGrossSales
69
70  // set commission employee's commission rate
71  void BasePlusCommissionEmployee::setCommissionRate( double rate )
72  {
73     /* Call commissionEmployee's setCommissionRate function */
74  } // end function setCommissionRate
75
76  // return commission employee's commission rate
77  double BasePlusCommissionEmployee::getCommissionRate() const
78  {
79     /* Call commissionEmployee's getCommissionRate function */
80  } // end function getCommissionRate
81
82  // set base salary
83  void BasePlusCommissionEmployee::setBaseSalary( double salary )
84  {
85     baseSalary = ( salary < 0.0 ) ? 0.0 : salary;
86  } // end function setBaseSalary
87
88  // return base salary
89  double BasePlusCommissionEmployee::getBaseSalary() const
90  {
91     return baseSalary;
92  } // end function getBaseSalary
93
94  // calculate earnings
95  double BasePlusCommissionEmployee::earnings() const
96  {
97     return getBaseSalary() +
98        /* Call commissionEmployee's earnings function */;
99  } // end function earnings
100
101 // print BasePlusCommissionEmployee object
102 void BasePlusCommissionEmployee::print() const
103 {
104    cout << "base-salaried ";
105
106    // invoke composed CommissionEmployee object's print function
107    /* Call commissionEmployee's print function */
108
109    cout << "\nbase salary: " << getBaseSalary();
110 } // end function print
```

Fig. L 12.11 | Contents of `BasePlusCommissionEmployee.cpp`. (Part 3 of 3.)

Lab Exercises

Name:

Lab Exercise 2 — Composition

```
// Lab 2: composition.cpp
// Testing class BasePlusCommissionEmployee.
#include <iostream>
#include <iomanip>
using namespace std;

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

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

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

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

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

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

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

Fig. L 12.12 | Contents of `composition.cpp`.

Problem-Solving Tips

1. To implement `BasePlusCommissionEmployee` using composition, include a `ComissionEmployee` object as a data member in the `BasePlusCommissionEmployee` class.
2. To access a member of `CommissionEmployee` inside a member function of `BasePlusCommissionEmployee`, it must be preceded by the name of the `CommissionEmployee` object and the dot operator.
3. Most of `BasePlusCommissionEmployee`'s member functions will be implemented by simply calling the same member function from the `CommissionEmployee` object; this is known as "delegation."

Lab Exercises

Name:

Debugging

Name: _____ **Date:** _____

Section: _____

The program (Fig. L 12.13–Fig. L 12.19) in this section does not run properly. Fix all the compilation errors so that the program will compile successfully. Once the program compiles, compare the output with the sample output, and eliminate any logic errors that may exist. The sample output demonstrates what the program's output should be once the program's code has been corrected.

Sample Output

```
This animal's height and weight are as follows
Height: 0       Weight: 0

This animal is a dog, its name is: Fido
This animal's height and weight are as follows
Height: 60      Weight: 120

This animal is a dog, its name is: Toto
This animal's height and weight are as follows
Height: 0       Weight: 0

This animal is a lion
This animal's height and weight are as follows
Height: 45      Weight: 300

Animal 1 now has the same height and weight as dog 1
This animal's height and weight are as follows
Height: 60      Weight: 120

Dog 2 now has the same height and weight as animal 1
This animal is a dog, its name is: Toto
This animal's height and weight are as follows
Height: 60      Weight: 120
```

Broken Code

```
1  // Debugging: Animal.h
2  #ifndef ANIMAL_H
3  #define ANIMAL_H
4
5  #include <string>
6  using namespace std;
7
8  // class Animal definition
9  class Animal
10 {
11 public:
12    Animal( const int = 0, const int = 0 );
```

Fig. L 12.13 | Contents of Animal.h. (Part 1 of 2)

Lab Exercises

Name:

Debugging

```
13
14    void setHeight( int );
15    int getHeight() const;
16
17    void setWeight( int );
18    int getWeight() const;
19
20    string getName() const;
21    void print() const;
22 private:
23    int height;
24    int weight;
25 }; // end class Animal
26
27 #endif // ANIMAL_H
```

Fig. L 12.13 | Contents of `Animal.h`. (Part 2 of 2.)

```
1  // Debugging: Animal.cpp
2  #include <iostream>
3  using namespace std;
4
5  #include "Animal.h"
6
7  // default constructor
8  Animal::Animal( const int h, const int w )
9  {
10    height = h;
11    weight = w;
12 } // end class Animal constructor
13
14 // function print definition
15 void Animal::print() const
16 {
17    cout << "This animal's height and weight are as follows\n"
18         << "Height: " << height << "\tWeight: " << weight
19         << endl << endl;
20 } // end function print
21
22 // return height
23 int Animal::getHeight() const
24 {
25    return height;
26 } // end function getHeight
27
28 // return weight
29 int Animal::getWeight() const
30 {
31    return weight;
32 } // end function getWeight
33
34 // function print definition
35 void Animal::setHeight( const int h )
36 {
```

Fig. L 12.14 | Contents of `Animal.cpp`. (Part 1 of 2.)

Lab Exercises

Name:

Debugging

```
37      height = h;
38   } // end function setHeight
39
40   // function print definition
41   void Animal::setWeight( const int w )
42   {
43      weight = w;
44   } // end function setWeight
45
46   // return name
47   string Animal::getName() const
48   {
49      return name;
50   } // end function getName
```

Fig. L 12.14 | Contents of `Animal.cpp`. (Part 2 of 2.)

```
1   // Debugging: Lion.h
2
3   #ifndef LION_H
4   #define LION_H
5
6   #include "Animal.h"
7
8   // class Lion definition
9   class Lion
10  {
11  public:
12     Lion( const int = 0, const int = 0 );
13
14     void print() const;
15  }; // end class Lion
16
17  #endif // LION_H
```

Fig. L 12.15 | Contents of `Lion.h`.

```
1   // Debugging: Lion.cpp
2   #include <iostream>
3   using namespace std;
4
5   #include "Lion.h"
6
7   // default constructor
8   Lion::Lion( const int h, const int w )
9      : Animal( h, w )
10  {
11     // empty
12  } // end class Lion constructor
13
```

Fig. L 12.16 | Contents of `Lion.cpp`. (Part 1 of 2.)

Lab Exercises

Name:

Debugging

```
14  // function print definition
15  void Lion::print() const
16  {
17     cout << "This animal is a lion\n";
18     print();
19  } // end function print
```

Fig. L 12.16 | Contents of `Lion.cpp`. (Part 2 of 2.)

```
1   // Debugging: Dog.h
2   #ifndef DOG_H
3   #define DOG_H
4
5   #include "Animal.h"
6
7   // class Dog definition
8   class Dog : public Animal
9   {
10  public:
11     Dog( const int, const int, string = "Toto" );
12
13     void Print() const;
14     void setName( string );
15  private:
16     string name;
17  }; // end class Dog
18
19  #endif // DOG_H
```

Fig. L 12.17 | Contents of `Dog.h`.

```
1   // Debugging: Dog.cpp
2   #include <iostream>
3   using namespace std;
4
5   #include "Dog.h"
6
7   // constructor
8   Dog::Dog( const int h, const int w, string n )
9      : Animal( h, w )
10  {
11     setName( n );
12  } // end class Dog constructor
13
14  // function setName definition
15  void Dog::setName( const char * n )
16  {
17     n = name;
18  } // end function setName
19
```

Fig. L 12.18 | Contents of `Dog.cpp`. (Part 1 of 2.)

Lab Exercises

Name:

Debugging

```
20  // function print definition
21  void Dog::Print() const
22  {
23     cout << "This animal is a dog, its name is: " << name << endl;
24
25     print();
26  } // end function print
```

Fig. L 12.18 | Contents of `Dog.cpp`. (Part 2 of 2.)

```
1   // Debugging: debugging.cpp
2   #include <iostream>
3   using namespace std;
4
5   #include "Animal.h"
6   #include "Lion.h"
7
8   int main()
9   {
10     Animal a1( 0, 0 );
11     Dog d1( 60, 120, "Fido" );
12     Dog d2;
13     Lion lion1( 45, 300 );
14
15     a1.print();
16     d1.print();
17     d2.print();
18     lion1.print();
19
20     a1 = d1;
21     cout << "Animal 1 now has the same height and weight "
22          << "as dog 1\n";
23     a1.print();
24
25     d2 = a1;
26     cout << "Dog 2 now has the same height and weight as animal 1\n"
27     d2.print();
28  } // end main
```

Fig. L 12.19 | Contents of `debugging.cpp`.

Postlab Activities

Coding Exercises

Name: _____ **Date:** _____

Section: _____

These coding exercises reinforce the lessons learned in the lab and provide additional programming experience outside the classroom and laboratory environment. They serve as a review after you have completed the *Prelab Activities* and *Lab Exercises* successfully.

For each of the following problems, write a program or a program segment that performs the specified action:

1. Write the header file for class `Base`, then, write the header file for class `Derived`, which inherits `publicly` from class `Base`. Do not provide any class members for either class.

Postlab Activities

Name:

Coding Exercises

2. Change the class definition for `Derived` from *Coding Exercise 1* so that `protected` inheritance is used.

3. Modify class `Base` from *Coding Exercise 1* to include two `private` data members, a `string` and an `integer`. Name the `private` data members any way you wish. Write a `print` member function for `Base` that prints the values stored in those `private` data members, separated by a hyphen (-).

Postlab Activities

Name:

Coding Exercises

4. Modify class `Derived` from *Coding Exercise 1* to include two `private` data members, a `string` and an `integer`. Name the `private` data members any way you wish. Redefine the `print` member function in class `Derived`. This member function should print the values stored in those `private` data members, separated by a colon and a space.

Postlab Activities

Name:

Programming Challenges

Name: _____ **Date:** _____

Section: _____

The *Programming Challenges* are more involved than the *Coding Exercises* and may require a significant amount of time to complete. Write a C++ program for each of the problems in this section. The answers to these problems are available from the Companion Website for *C++ How to Program, Seventh Edition* at `www.pearsonhighered.com/deitel/`. Pseudocode, hints and/or sample outputs are provided to aid you in your programming.

1. Package-delivery services, such as FedEx®, DHL® and UPS®, offer a number of different shipping options, each with specific costs associated. Create an inheritance hierarchy to represent various types of packages. Use `Package` as the base class of the hierarchy, then include classes `TwoDayPackage` and `OvernightPackage` that derive from `Package`. Base class `Package` should include data members representing the name, address, city, state and ZIP code for both the sender and the recipient of the package, in addition to data members that store the weight (in ounces) and cost per ounce to ship the package. `Package`'s constructor should initialize these data members. Ensure that the weight and cost per ounce contain positive values. `Package` should provide a public member function `calculateCost` that returns a double indicating the cost associated with shipping the package. `Package`'s `calculateCost` function should determine the cost by multiplying the weight by the cost per ounce. Derived class `TwoDayPackage` should inherit the functionality of base class `Package`, but also include a data member that represents a flat fee that the shipping company charges for two-day-delivery service. `TwoDayPackage`'s constructor should receive a value to initialize this data member. `TwoDayPackage` should redefine member function `calculateCost` so that it computes the shipping cost by adding the flat fee to the weight-based cost calculated by base class `Package`'s `calculateCost` function. Class `OvernightPackage` should inherit directly from class `Package` and contain an additional data member representing an additional fee per ounce charged for overnight-delivery service. `OvernightPackage` should redefine member function `calculateCost` so that it adds the additional fee per ounce to the standard cost per ounce before calculating the shipping cost. Write a test program that creates objects of each type of `Package` and tests member function `calculateCost`.

Postlab Activities

Name:

Programming Challenges

Hint:

- Sample output:

```
Package 1:

Sender:
Lou Brown
1 Main St
Boston, MA 11111

Recipient:
Mary Smith
7 Elm St
New York, NY 22222

Cost: $4.25

Package 2:

Sender:
Lisa Klein
5 Broadway
Somerville, MA 33333

Recipient:
Bob George
21 Pine Rd
Cambridge, MA 44444

Cost: $8.82

Package 3:

Sender:
Ed Lewis
2 Oak St
Boston, MA 55555

Recipient:
Don Kelly
9 Main St
Denver, CO 66666

Cost: $11.64
```