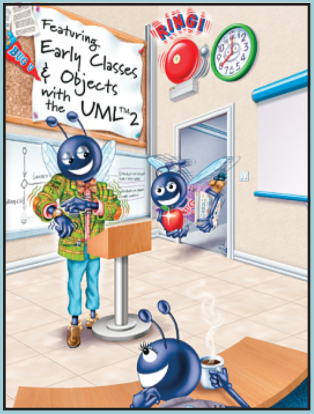
12



Object-Oriented Programming: Inheritance

OBJECTIVES

In this chapter you will learn:

- To create classes by inheriting from existing classes.
- How inheritance promotes software reuse.
- The notions of base classes and derived classes and the relationships between them.
- The `protected` member access specifier.
- The use of constructors and destructors in inheritance hierarchies.
- The differences between `public`, `protected` and `private` inheritance.

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	
Programming Output	29, 30	
Correct the Code	31, 32, 33, 34	
Lab Exercises		
Lab Exercise 1 — `Account` Hierarchy	YES NO	
Lab Exercise 2 — Composition	YES NO	
Follow-Up Question and Activity	1	
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: Fifth 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. Abstraction	b)	Deriving from more than one base class.
___	3. Derived class	c)	Class that is created by inheriting from an existing class.
___	4. “Has a” relationship	d)	The inheritance relationship.
___	5. “Is a” relationship	e)	Passes arguments to the base-class constructor.
___	6. Single inheritance	f)	Base class that is not listed explicitly in the derived class's definition.
___	7. Base class	g)	“Seeing the forest through the trees.”
___	8. Indirect base class	h)	The composition relationship.
___	9. Base-class initializer	i)	Deriving from only one base class.
___	10. Multiple inheritance	j)	A form of software reusability in which 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:

11. Inheritance promotes software ____________.

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

13. A base class's ____________ may be accessed by members and `friends` of the base class and by members and `friends` of the derived class.

14. With ____________ inheritance, a class is derived from only one base class.

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

16. A derived class's constructor always calls the constructor of its ____________ first.

17. Destructors are called in ____________ order of constructor calls.

18. The three forms of inheritance are ____________, ____________ and ____________.

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

20. 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.

21. 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.

22. How does inheritance promote software reusability?

23. What is `protected` access?

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

Prelab Activities

Name:

Short Answer

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

26. 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?)

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

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

Prelab Activities

Name:

Programming Output

Name: ____________________ **Date:** ____________________

Section: ____________________

For each of the given program segments, read the code and write the output in the space provided below each program. [*Note:* Do not execute these programs on a computer.]

29. What is output by the following program?

```cpp
#include <iostream>

using std::cout;
using std::endl;

class Point
{
public:
   Point( int a, int b )
      : x( a ), y( b )
   {
   }

   void print()
   {
      cout << "[" << x << ", " << y << "]";
   }
private:
   int x;
   int y;
};

class Circle : public Point
{
public:
   Circle( int a, int b, int c )
      : Point( a, b ), r( c )
   {
   }

   void print()
   {
      cout << "Center = ";
      Point::print();
      cout << "; Radius = " << r;
   }
private:
   int r;
};

int main()
{
   Point p( 8, 14 );
   p.print();
```

Prelab Activities

Name:

Programming Output

```cpp
   cout << endl;

   Circle circle( 9, 10, 22 );
   circle.print();
   cout << endl;

   return 0;
} // end main
```

Your answer:

30. What is output by the following program?

```cpp
#include <iostream>

using std::cout;
using std::endl;

class Point
{
public:
   Point( int a, int b )
      : x( a ), y( b )
   {
   }

   void print()
   {
      cout << "[" << x << ", " << y << "]";
   }
private:
   int x;
   int y;
};

class Circle : public Point
{
public:
   Circle( int a, int b, int c )
      : Point( a, b ), r( c )
   {
   }

   void print()
   {
      cout << "Center = ";
      Point::print();
      cout << "; Radius = " << r;
   }
```

Prelab Activities

Name:

Programming Output

```cpp
private:
   int r;
};

class Cylinder : public Circle
{
public:
   Cylinder( int a, int b, int c, int d )
      : Circle( a, b, c ), h( d )
   {
   }

   void print()
   {
      Circle::print();
      cout << "; Height = " << h;
   }
private:
   int h;
};

int main()
{
   Point point( 1, 2 );
   Circle circle( -5, -12, 53 );
   Cylinder cylinder( 44, 98, 6, 26 );

   circle.print();
   cout << endl;

   point.print();
   cout << endl;

   cylinder.print();
   cout << endl;

   return 0;
} // end main
```

Your answer:

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, syntax 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.]

31. Class X inherits from class Y.

```
#include <iostream>

using std::cout;

// class Y definition
class Y
{
public:
   Y(); // default constructor
   ~Y(); // destructor
private:
   int data;
}; // end class Y

// class X definition
class X ; public Y
{
public:
   // function print
   void print() const
   {
      cout << data;
   } // end function print
}; // end class X
```

Your answer:

Prelab Activities

Name:

Correct the Code

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

```cpp
#include <iostream>

using std::cout;

// class Base definition
class Base
{
private:
   // constructor
   Base( int b )
   {
      cout << b;
   } // end class Base constructor
}; // end class Base

// class Derived definition
class Derived : public Base
{
   // constructor calls base-class constructor
   Derived( int a )
      : Base( a )
   {
      // empty
   } // end class Derived constructor
}; // end class Derived

int main()
{
   Derived d( 5 );

   return 0;
} // end main
```

Your answer:

Prelab Activities

Name:

Correct the Code

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

```
#include <iostream>
using std::cout;

// class A definition
class A
{
public:
   // constructor
   A( int a )
   {
      value = a;
   } // end class A constructor

   // return value
   int getValue() const
   {
      return value;
   } // end function getValue
private:
   int value;
}; // end class A

// class B definition
class B
{
public:
   // constructor
   B( int b )
      : A( b )
   {
      // empty
   } // end class B constructor
}; // end class B

int main()
{
   B object( 50 );
   cout << object.getValue();

   return 0;
} // end main
```

Your answer:

Prelab Activities

Name:

Correct the Code

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

```
#include <iostream>
using std::cout;

// 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();

   return 0;
} // end main
```

Your answer:

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 at `www.deitel.com` and `www.prenhall.com./deitel`.

Lab Objectives

This lab was designed to reinforce programming concepts from Chapter 12 of *C++ How To Program: Fifth 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 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

Lab Exercises

Name:

Lab Exercise 1 — Account Hierarchy

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

```
// Lab 1: Account.h
// Definition of Account class.
#ifndef ACCOUNT_H
#define ACCOUNT_H

class Account
{
public:
   Account( double ); // constructor initializes balance
   void credit( double ); // add an amount to the account balance
   bool debit( double ); // subtract an amount from the account balance
   void setBalance( double ); // sets the account balance
   double getBalance(); // return the account balance
private:
   double balance; // data member that stores the balance
}; // end class Account

#endif
```

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

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

#include "Account.h" // include definition of class Account

// Account constructor initializes data member balance
Account::Account( double initialBalance )
{
   // if initialBalance is greater than or equal to 0.0, set this value
   // as the balance of the Account
   if ( initialBalance >= 0.0 )
      balance = initialBalance;
   else // otherwise, output message and set balance to 0.0
   {
      cout << "Error: Initial balance cannot be negative." << endl;
      balance = 0.0;
   } // end if...else
} // end Account constructor

// credit (add) an amount to the account balance
void Account::credit( double amount )
{
   balance = balance + amount; // add amount to balance
} // end function credit

// debit (subtract) an amount from the account balance
// return bool indicating whether money was debited
bool Account::debit( double amount )
{
```

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

Lab Exercises

Name:

Lab Exercise 1 — Account Hierarchy

```
   if ( amount > balance ) // debit amount exceeds balance
   {
      cout << "Debit amount exceeded account balance." << endl;
      return false;
   } // end if
   else // debit amount does not exceed balance
   {
      balance = balance - amount;
      return true;
   } // end else
} // end function debit

// set the account balance
void Account::setBalance( double newBalance )
{
   balance = newBalance;
} // end function setBalance

// return the account balance
double Account::getBalance()
{
   return balance;
} // end function getBalance
```

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

```
// Lab 1: SavingsAccount.h
// Definition of SavingsAccount class.
#ifndef SAVINGS_H
#define SAVINGS_H

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

/* Write a line to have class SavingsAccount inherit publicly from Account */
{
public:
   // constructor initializes balance and interest rate
   /* Declare a two-parameter constructor for SavingsAccount */

   /* Declare member function calculateInterest */
private:
   /* Declare data member interestRate */
}; // end class SavingsAccount

#endif
```

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

Lab Exercises

Name:

Lab Exercise 1 — Account Hierarchy

```cpp
// 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`.

```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`.

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

#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 */

```

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

Lab Exercises

Name:

Lab Exercise 1 — Account Hierarchy

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

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

```
// Lab 1: bankAccounts.cpp
// Test program for Account hierarchy.
#include <iostream>
using std::cout;
using std::endl;

#include <iomanip>
using std::setprecision;
using std::fixed;

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

int main()
{
   Account account1( 50.0 ); // create Account object
   SavingsAccount account2( 25.0, .03 ); // create SavingsAccount object
   CheckingAccount account3( 80.0, 1.0 ); // create CheckingAccount object

   cout << fixed << setprecision( 2 );

   // display initial balance of each object
   cout << "account1 balance: $" << account1.getBalance() << endl;
   cout << "account2 balance: $" << account2.getBalance() << endl;
   cout << "account3 balance: $" << account3.getBalance() << endl;

   cout << "\nAttempting to debit $25.00 from account1." << endl;
   account1.debit( 25.0 ); // try to debit $25.00 from account1
   cout << "\nAttempting to debit $30.00 from account2." << endl;
   account2.debit( 30.0 ); // try to debit $30.00 from account2
   cout << "\nAttempting to debit $40.00 from account3." << endl;
   account3.debit( 40.0 ); // try to debit $40.00 from account3

   // display balances
   cout << "\naccount1 balance: $" << account1.getBalance() << endl;
   cout << "account2 balance: $" << account2.getBalance() << endl;
   cout << "account3 balance: $" << account3.getBalance() << endl;

   cout << "\nCrediting $40.00 to account1." << endl;
   account1.credit( 40.0 ); // credit $40.00 to account1
```

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

Lab Exercises

Name:

Lab Exercise 1 — Account Hierarchy

```
   cout << "\nCrediting $65.00 to account2." << endl;
   account2.credit( 65.0 ); // credit $65.00 to account2
   cout << "\nCrediting $20.00 to account3." << endl;
   account3.credit( 20.0 ); // credit $20.00 to account3

   // display balances
   cout << "\naccount1 balance: $" << account1.getBalance() << endl;
   cout << "account2 balance: $" << account2.getBalance() << endl;
   cout << "account3 balance: $" << account3.getBalance() << endl;

   // add interest to SavingsAccount object account2
   /* Declare a variable interestEarned and assign it the interest
      account2 should earn */
   cout << "\nAdding $" << interestEarned << " interest to account2."
      << endl;
   /* Write a statement to credit the interest to account2's balance */

   cout << "\nNew account2 balance: $" << account2.getBalance() << endl;
   return 0;
} // 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 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 at `www.deitel.com` and `www.prenhall.com/deitel`.

Lab Objectives

This lab was designed to reinforce programming concepts from Chapter 12 of *C++ How To Program: Fifth 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

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

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

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

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

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

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

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

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

   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 std::cout;

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

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

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

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

// 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.)

Lab Exercises

Name:

Lab Exercise 2 — Composition

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

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

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

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

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

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

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

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

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

Lab Exercises

Name:

Lab Exercise 2 — Composition

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

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

#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`.

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

// 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

```
// constructor
BasePlusCommissionEmployee::BasePlusCommissionEmployee(
   const string &first, const string &last, const string &ssn,
   double sales, double rate, double salary )
   // initialize composed object
   : /* Initialize the commissionEmployee data member,
        pass (first, last, ssn, sales, rate) to its constructor */
{
   setBaseSalary( salary ); // validate and store base salary
} // end BasePlusCommissionEmployee constructor

// set commission employee's first name
void BasePlusCommissionEmployee::setFirstName( const string &first )
{
   /* Call commissionEmployee's setFirstName function */
} // end function setFirstName

// return commission employee's first name
string BasePlusCommissionEmployee::getFirstName() const
{
   /* Call commissionEmployee's getFirstName function */
} // end function getFirstName

// set commission employee's last name
void BasePlusCommissionEmployee::setLastName( const string &last )
{
   /* Call commissionEmployee's setLastName function */
} // end function setLastName

// return commission employee's last name
string BasePlusCommissionEmployee::getLastName() const
{
   /* Call commissionEmployee's getLastName function */
} // end function getLastName

// set commission employee's social security number
void BasePlusCommissionEmployee::setSocialSecurityNumber(
   const string &ssn )
{
   /* Call commissionEmployee's setSocialSecurity function */
} // end function setSocialSecurityNumber

// return commission employee's social security number
string BasePlusCommissionEmployee::getSocialSecurityNumber() const
{
   /* Call commissionEmployee's getSocialSecurity function */
} // end function getSocialSecurityNumber

// set commission employee's gross sales amount
void BasePlusCommissionEmployee::setGrossSales( double sales )
{
   /* Call commissionEmployee's setGrossSales function */
} // end function setGrossSales

```

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

Lab Exercises

Name:

Lab Exercise 2 — Composition

```cpp
// return commission employee's gross sales amount
double BasePlusCommissionEmployee::getGrossSales() const
{
   /* Call commissionEmployee's getGrossSales function */
} // end function getGrossSales

// set commission employee's commission rate
void BasePlusCommissionEmployee::setCommissionRate( double rate )
{
   /* Call commissionEmployee's setCommissionRate function */
} // end function setCommissionRate

// return commission employee's commission rate
double BasePlusCommissionEmployee::getCommissionRate() const
{
   /* Call commissionEmployee's getCommissionRate function */
} // end function getCommissionRate

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

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

// calculate earnings
double BasePlusCommissionEmployee::earnings() const
{
   return getBaseSalary() +
      /* Call commissionEmployee's earnings function */;
} // end function earnings

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

   // invoke composed CommissionEmployee object's print function
   /* Call commissionEmployee's print function */

   cout << "\nbase salary: " << getBaseSalary();
} // 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>
using std::cout;
using std::endl;
using std::fixed;

#include <iomanip>
using std::setprecision;

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

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

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

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

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

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

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

   return 0;
} // end main
```

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 `BasePlusCommissionEm-ployee`, 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:

Lab Exercise 2 — Composition

Follow-Up Question and Activity

1. Assess the relative merits of the two approaches for designing classes `CommissionEmployee` and `BasePlus-CommissionEmployee`, as well as for object-oriented programs in general. Which approach is more natural? Why?

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

```
// Debugging: Animal.h
#ifndef ANIMAL_H
#define ANIMAL_H

#include <string>
using std::string;

// class Animal definition
class Animal
{
public:
   Animal( const int = 0, const int = 0 );
```

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

Lab Exercises

Name:

Debugging

```

   void setHeight( int );
   int getHeight() const;

   void setWeight( int );
   int getWeight() const;

   string getName() const;
   void print() const;
private:
   int height;
   int weight;
}; // end class Animal

#endif // ANIMAL_H
```

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

```
// Debugging: Animal.cpp
#include <iostream>
using std::cout;
using std::endl;

#include "Animal.h"

// default constructor
Animal::Animal( const int h, const int w )
{
   height = h;
   weight = w;
} // end class Animal constructor

// function print definition
void Animal::print() const
{
   cout << "This animal's height and weight are as follows\n"
        << "Height: " << height << "\tWeight: " << weight
        << endl << endl;
} // end function print

// return height
int Animal::getHeight() const
{
   return height;
} // end function getHeight

// return weight
int Animal::getWeight() const
{
   return weight;
} // end function getWeight

// function print definition
void Animal::setHeight( const int h )
{
```

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

Lab Exercises

Name:

Debugging

```
   height = h;
} // end function setHeight

// function print definition
void Animal::setWeight( const int w )
{
   weight = w;
} // end function setWeight

// return name
string Animal::getName() const
{
   return name;
} // end function getName
```

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

```
// Debugging: Lion.h

#ifndef LION_H
#define LION_H

#include "Animal.h"

// class Lion definition
class Lion
{
public:
   Lion( const int = 0, const int = 0 );

   void print() const;
}; // end class Lion

#endif // LION_H
```

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

```
// Debugging: Lion.cpp
#include <iostream>

using std::cout;
using std::endl;
#include "Lion.h"

// default constructor
Lion::Lion( const int h, const int w )
   : Animal( h, w )
{
   // empty
} // end class Lion constructor

```

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

Lab Exercises

Name:

Debugging

```cpp
// function print definition
void Lion::print() const
{
   cout << "This animal is a lion\n";
   print();
} // end function print
```

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

```cpp
// Debugging: Dog.h
#ifndef DOG_H
#define DOG_H

#include "Animal.h"

// class Dog definition
class Dog : public Animal
{
public:
   Dog( const int, const int, string = "Toto" );

   void Print() const;
   void setName( string );
private:
   string name;
}; // end class Dog

#endif // DOG_H
```

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

```cpp
// Debugging: Dog.cpp
#include <iostream>

using std::cout;
using std::endl;

#include "Dog.h"

// constructor
Dog::Dog( const int h, const int w, string n )
   : Animal( h, w )
{
   setName( n );
} // end class Dog constructor

// function setName definition
void Dog::setName( const char * n )
{
   n = name;
} // end function setName

```

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

Lab Exercises

Name:

Debugging

```cpp
// function print definition
void Dog::Print() const
{
   cout << "This animal is a dog, its name is: " << name << endl;

   print();
} // end function print
```

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

```cpp
// Debugging: debugging.cpp
#include <iostream>
using std::cout;
using std::endl;

#include "Animal.h"
#include "Lion.h"

int main()
{
   Animal a1( 0, 0 );
   Dog d1( 60, 120, "Fido" );
   Dog d2;
   Lion lion1( 45, 300 );

   a1.print();
   d1.print();
   d2.print();
   lion1.print();

   a1 = d1;
   cout << "Animal 1 now has the same height and weight "
        << "as dog 1\n";
   a1.print();

   d2 = a1;
   cout << "Dog 2 now has the same height and weight as animal 1\n"
   d2.print();

   return 0;
} // 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 `Base1`, then, write the header file for class `Derived`, which inherits `publicly` from class `Base1`. 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 at `www.deitel.com` and `www.prenhall.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
```