3

# Introduction to Classes and Objects

*Your public servants serve you right.*
—Adlai E. Stevenson

*Knowing how to answer one who speaks,*
*To reply to one who sends a message.*
—Amenemope

*You'll see something new. Two things. And I call them Thing One and Thing Two.*

## OBJECTIVES

In this chapter you'll learn:

- What classes, objects, member functions and data members are.
- How to define a class and use it to create an object.
- How to define member functions in a class to implement the class's behaviors.
- How to declare data members in a class to implement the class's attributes.
- How to call a member function of an object to make that member function perform its task.
- The differences between data members of a class and local variables of a function.
- How to use a constructor to ensure that an object's data is initialized when the object is created.
- How to engineer a class to separate its interface from its implementation and encourage reuse.

*Nothing can have value without being an object of utility.*
—Karl Marx

# Assignment Checklist

Name: _____ Date: _____

Section: _____

| Exercises | Assigned: Circle assignments | Date Due |
|---|---|---|
| **Prelab Activities** | | |
| Matching | YES NO | |
| Fill in the Blank | 13, 14, 15, 16, 17, 18, 19, 20, 21 | |
| Short Answer | 22, 23, 24, 25, 26 | |
| Programming Output | 27, 28, 29, 30, 31 | |
| Correct the Code | 32, 33, 34, 35 | |
| **Lab Exercises** | | |
| Exercise 1 — Modifying Class `Account` | YES NO | |
| Exercise 2 — Modifying Class `GradeBook` | YES NO | |
| Exercise 3 — Creating an `Employee` Class | YES NO | |
| Debugging | YES NO | |
| **Labs Provided by Instructor** | | |
| 1. | | |
| 2. | | |
| 3. | | |
| **Postlab Activities** | | |
| Coding Exercises | 1, 2, 3, 4, 5, 6, 7, 8 | |
| Programming Challenges | 1, 2 | |

# Prelab Activities

## Matching

**Name:** _____ **Date:** _____

**Section:** _____

After reading Chapter 3 of *C++ How to Program, Seventh Edition*, answer the given questions. The questions are intended to test and reinforce your understanding of key concepts. You may answer the questions either before or during the lab.

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

| Term | Description |
|---|---|
| ___ 1. data members | a) Primitive type that represents a single-precision floating-point number. |
| ___ 2. calling function | b) Causes C++ to execute a function. |
| ___ 3. object | c) Defines a class's attributes. |
| ___ 4. `public` member function | d) A function that assigns a value to a `private` data member. |
| ___ 5. class definition | e) An instance of a class. |
| ___ 6. function call | f) A function that is accessible from outside of the class in which it is declared. |
| ___ 7. parameter | g) Additional information a function requires to help it perform its task. |
| ___ 8. *set* function | h) Primitive type that represents a double-precision floating-point number. |
| ___ 9. default constructor | i) The compiler provides one of these for a class that does not declare any. |
| ___ 10. client | j) Encompasses all of the attributes and behaviors of a class. |
| ___ 11. `double` | k) Uses an object's or class's functions. |
| ___ 12. `float` | l) Receives the return value from a function. |

## Prelab Activities

Name:

### Fill in the Blank

**Name:** _____ **Date:** _____

**Section:** _____

Fill in the blanks in each of the following statements:

13. Each function definition can specify _____ that represent additional information the function requires to perform its task correctly.

14. Declaring data members with access modifier _____ is known as information hiding.

15. The initial value of a `string` is the _____ which does not contain any characters.

16. Variables declared in the body of a particular function are known as _____ and can be used only in that function.

17. Each parameter must specify both a(n) _____ and a(n) _____.

18. Function _____ reads characters until a newline character is encountered.

19. It is customary to define a class in a(n) _____ file that has a `.h` filename extension.

20. A(n) _____ normally consists of one or more member functions that manipulate the attributes that belong to a particular object.

21. Classes often provide `public` member functions to allow clients of the class to _____ or _____ the values of `private` data members.

## Prelab Activities

Name:

### Short Answer

**Name:** _____ **Date:** _____

**Section:** _____

Answer the given questions in the spaces provided. Your answers should be as concise as possible; aim for two or three sentences.

22. List the parts of a function header and why each one is important.

23. How are constructors and functions similar? How are they different?

24. What is the relationship between a client of an object and the object's `public` members?

25. What types of declarations are contained within a class definition?

26. Distinguish between a primitive-type variable and a class-type variable.

## 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.]

For *Programming Output Exercises 27–31*, use the following class definition.

```
1  // Definition of Account class.
2  class Account
3  {
4  public:
5     Account( int ); // constructor initializes balance
6     void credit( int ); // add an amount to the account balance
7     int getBalance(); // return the account balance
8  private:
9     int balance; // data member that stores the balance
10 }; // end class Account
```

```
1  // Member-function definitions for class Account.
2  #include <iostream>
3  using namespace std;
4
5  #include "Account.h" // include definition of class Account
6
7  // Account constructor initializes data member balance
8  Account::Account( int initialBalance )
9  {
10    balance = 0; // assume that the balance begins at 0
11
12    // if initialBalance is greater than 0, set this value as the
13    // balance of the Account; otherwise, balance remains 0
14    if ( initialBalance > 0 )
15       balance = initialBalance;
16
17    // if initialBalance is negative, print error message
18    if ( initialBalance < 0 )
19       cout << "Error: Initial balance cannot be negative.\n" << endl;
20 } // end Account constructor
21
22 // credit (add) an amount to the account balance
23 void Account::credit( int amount )
24 {
25    balance = balance + amount; // add amount to balance
26 } // end function credit
27
28 // return the account balance
29 int Account::getBalance()
30 {
31    return balance; // gives the value of balance to the calling function
32 } // end function getBalance
```

## Prelab Activities

Name:

### Programming Output

27. What is output by the following `main` function?

```
1  int main()
2  {
3     Account account1( 3550 );
4
5     cout << "account1 balance: $" << account1.getBalance() << endl;
6  } // end main
```

*Your answer:*

28. What is output by the following `main` function?

```
1  int main()
2  {
3     Account account1( -2017 );
4
5     cout << "account1 balance: $" << account1.getBalance() << endl;
6  } // end main
```

*Your answer:*

29. What is output by the following `main` function?

```
1  int main()
2  {
3     Account account1( 1533 );
4
5     cout << "account1 balance: $" << account1.getBalance() << endl;
6     cout << "adding $253 to account1 balance" << endl;
7
8     account1.credit( 253 );
9     cout << "account1 balance: $" << account1.getBalance() << endl;
10 } // end main
```

## Prelab Activities

Name:

### Programming Output

*Your answer:*

30. What is output by the following `main` function?

```
1  int main()
2  {
3     Account account1( 2770 );
4
5     cout << "account1 balance: $" << account1.getBalance() << endl;
6     cout << "adding $375 to account1 balance" << endl;
7
8     account1.credit( 375 );
9     cout << "account1 balance: $" << account1.getBalance() << endl;
10 } // end main
```

*Your answer:*

31. What is output by the following `main` function?

```
1  int main()
2  {
3     Account account1( 799 );
4
5     cout << "account1 balance: $" << account1.getBalance() << endl;
6     cout << "adding -$114 to account1 balance" << endl;
7
8     account1.credit( -114 );
9     cout << "account1 balance: $" << account1.getBalance() << endl;
10 } // 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 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.]

For *Correct the Code Exercises 32–35*, use the following class definition.

```
1  // Definition of GradeBook class that stores the course name.
2  #include <string> // program uses C++ standard string class
3  using namespace std;
4
5  // GradeBook class definition
6  class GradeBook
7  {
8  public:
9     // constructor initializes course name
10    GradeBook( string );
11    void setCourseName( string ); // function to set the course name
12    string getCourseName(); // function to retrieve the course name
13    void displayMessage(); // display welcome message and course name
14 private:
15    string courseName; // course name for this GradeBook
16 }; // end class GradeBook
```

```
1  // Member-function definitions for class GradeBook.
2  #include <iostream>
3  using namespace std;
4
5  // include definition of class GradeBook from GradeBook.h
6  #include "GradeBook.h"
7
8  // constructor initializes courseName
9  // with string supplied as argument
10 GradeBook::GradeBook( string course )
11 {
12    setCourseName( course ); // initializes courseName
13 } // end GradeBook constructor
14
15 // function to set the course name
16 void GradeBook::setCourseName( string name )
17 {
18    courseName = name; // store the course name
19 } // end function setCourseName
20
21 // function to retrieve the course name
22 string GradeBook::getCourseName()
23 {
```

## Prelab Activities

Name:

### Correct the Code

```
24     return courseName;
25  } // end function getCourseName
26
27  // display a welcome message and the course name
28  void GradeBook::displayMessage()
29  {
30     // display a welcome message containing the course name
31     cout << "Welcome to the grade book for\n" << getCourseName() << "!"
32        << endl;
33  } // end function displayMessage
```

32. The following code segment should create a new `GradeBook` object:

```
1  Gradebook gradeBook( "Introduction to C++", 25 );
```

*Your answer:*

33. The following code segment should set the course name for the `gradeBook` object:

```
1  setCourseName( gradeBook, "Advanced C++" )
```

*Your answer:*

34. The following code segment should ask the user to input a course name. That name should then be set as the course name of your `gradeBook`.

```
1  cout << "Please enter the course name:" << endl;
2  cin.getline( inputName );
3
4  gradeBook.setCourseName();
```

## Prelab Activities

Name:

### Correct the Code

*Your answer:*

35. The following code segment should output `gradeBook`'s current course name:

```
cout << "The grade book's course name is: " << gradeBook.courseName << endl;
```

*Your answer:*

# Lab Exercises

## Lab Exercise 1 — Modifying Class Account

**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 3.1–Fig. L 3.3)
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 3 of *C++ How to Program, Seventh Edition*. In this lab, you will practice:

- Creating member functions.
- Invoking functions and receiving return values from functions.
- Testing a condition using an `if` statement.
- Outputting variables with stream insertion and the `cout` object.

### Description of the Problem

Modify class `Account` (Fig. L 3.1 and Fig. L 3.2) to provide a member function called `debit` that withdraws money from an `Account`. 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 a message indicating `"Debit amount exceeded account balance."` Modify class `AccountTest` (Fig. L 3.3) to test member function `debit`.

## Lab Exercises

Name:

### Lab Exercise 1 — Modifying Class Account

### Sample Output

```
account1 balance: $50.00
account2 balance: $0.00

Enter withdrawal amount for account1: 25

subtracting 25 from account1 balance

account1 balance: $25
account2 balance: $0.00

Enter withdrawal amount for account2: 10

subtracting 10 from account2 balance

Debit amount exceeded account balance.

account1 balance: $25
account2 balance: $0
```

### Program Template

```
1  // Lab 1: Account.h
2  // Definition of Account class.
3
4  class Account
5  {
6  public:
7     Account( int ); // constructor initializes balance
8     void credit( int ); // add an amount to the account balance
9     /* write code to declare member function debit. */
10    int getBalance(); // return the account balance
11 private:
12    int balance; // data member that stores the balance
13 }; // end class Account
```

**Fig. L 3.1** | 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( int initialBalance )
10 {
11    balance = 0; // assume that the balance begins at 0
12
13    // if initialBalance is greater than 0, set this value as the
14    // balance of the Account; otherwise, balance remains 0
```

**Fig. L 3.2** | Account.cpp. (Part 1 of 2.)

## Lab Exercises

Name:

### Lab Exercise 1 — Modifying Class Account

```
15     if ( initialBalance > 0 )
16        balance = initialBalance;
17
18     // if initialBalance is negative, print error message
19     if ( initialBalance < 0 )
20        cout << "Error: Initial balance cannot be negative.\n" << endl;
21  } // end Account constructor
22
23  // credit (add) an amount to the account balance
24  void Account::credit( int amount )
25  {
26     balance = balance + amount; // add amount to balance
27  } // end function credit
28
29  /* write code to define member function debit. */
30
31  // return the account balance
32  int Account::getBalance()
33  {
34     return balance; // gives the value of balance to the calling function
35  } // end function getBalance
```

**Fig. L 3.2** | Account.cpp. (Part 2 of 2.)

```
1   // Lab 1: AccountTest.cpp
2   // Create and manipulate Account objects.
3   #include <iostream>
4   using namespace std;
5
6   // include definition of class Account from Account.h
7   #include "Account.h"
8
9   // function main begins program execution
10  int main()
11  {
12     Account account1( 50 ); // create Account object
13     Account account2( 0 ); // create Account object
14
15     // display initial balance of each object
16     cout << "account1 balance: $" << account1.getBalance() << endl;
17     cout << "account2 balance: $" << account2.getBalance() << endl;
18
19     int withdrawalAmount; // stores withdrawal amount read from user
20
21     cout << "\nEnter withdrawal amount for account1: "; // prompt
22     cin >> withdrawalAmount; // obtain user input
23     cout << "\nsubtracting " << withdrawalAmount
24        << " from account1 balance\n\n";
25     /* write code to withdraw money from account1 */
26
27     // display balances
28     cout << "account1 balance: $" << account1.getBalance() << endl;
29     cout << "account2 balance: $" << account2.getBalance() << endl;
30
```

**Fig. L 3.3** | AccountTest.cpp (Part 1 of 2.)

## Lab Exercises

Name:

### Lab Exercise 1 — Modifying Class Account

```
31      cout << "\nEnter withdrawal amount for account2: "; // prompt
32      cin >> withdrawalAmount; // obtain user input
33      cout << "\nsubtracting " << withdrawalAmount
34         << " from account2 balance\n\n";
35      /* write code to withdraw money from account2 */
36
37      // display balances
38      cout << "account1 balance: $" << account1.getBalance() << endl;
39      cout << "account2 balance: $" << account2.getBalance() << endl;
40   } // end main
```

**Fig. L 3.3** | `AccountTest.cpp`. (Part 2 of 2.)

### Problem-Solving Tips

1. Declare `public` member function `debit` with a return type of `void`.
2. Use a parameter to enable the program to specify the amount the user wishes to withdraw.
3. In the body of member function `debit`, use an `if` statement to test whether the withdrawal amount is more than the balance. Output an appropriate message if the condition is `true`.
4. Use another `if` statement to test whether the withdrawal amount is less than or equal to the balance. Decrement the balance appropriately.
5. Be sure to follow the spacing and indentation conventions mentioned in the text.
6. If you have any questions as you proceed, ask your lab instructor for help.

## Lab Exercises

Name:

# Lab Exercise 2 — Modifying class GradeBook

**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. Problem of the Description
3. Sample Output
4. Program Template (Fig. L 3.4, Fig. L 3.5 and Fig. L 3.6)
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 3 of *C++ How to Program, Seventh Edition*. In this lab, you will practice:

- Declaring data members.
- Providing *set* and *get* functions to manipulate a data member's value.
- Declaring member functions with parameters.

## Description of the Problem

Modify class `GradeBook` (Fig. L 3.4 and Fig. L 3.6). Include a second `string` data member that represents the name of the course's instructor. Provide a *set* function to change the instructor's name and a *get* function to retrieve it. Modify the constructor to specify *two* parameters—one for the course name and one for the instructor's name. Modify member function `displayMessage` such that it first outputs the welcome message and course name, then outputs `"This course is presented by: "` followed by the instructor's name. Modify the test application (Fig. L 3.6) to demonstrate the class's new capabilities.

## Sample Output

```
Welcome to the grade book for
CS101 Introduction to C++ Programming!
This course is presented by: Sam Smith

Changing instructor name to Judy Jones

Welcome to the grade book for
CS101 Introduction to C++ Programming!
This course is presented by: Judy Jones
```

## Lab Exercises

Name:

### Lab Exercise 2 — Modifying class GradeBook

### Program Template

```
// Lab 2: GradeBook.h
// Definition of GradeBook class that stores an instructor's name.
#include <string> // program uses C++ standard string class
using namespace std;

// GradeBook class definition
class GradeBook
{
public:
   // constructor initializes course name and instructor name
   GradeBook( string, string );
   void setCourseName( string ); // function to set the course name
   string getCourseName(); // function to retrieve the course name
   /* write code to declare a get function for the instructor's name */
   /* write code to declare a set function for the instructor's name */
   void displayMessage(); // display welcome message and instructor name
private:
   string courseName; // course name for this GradeBook
   string instructorName; // instructor name for this GradeBook
}; // end class GradeBook
```

**Fig. L 3.4** | GradeBook.h.

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

// include definition of class GradeBook from GradeBook.h
#include "GradeBook.h"

// constructor initializes courseName and instructorName
// with strings supplied as arguments
GradeBook::GradeBook( string course, string instructor )
{
   setCourseName( course ); // initializes courseName
   setInstructorName( instructor ); // initializes instructorName
} // end GradeBook constructor

// function to set the course name
void GradeBook::setCourseName( string name )
{
   courseName = name; // store the course name
} // end function setCourseName

// function to retrieve the course name
string GradeBook::getCourseName()
{
   return courseName;
} // end function getCourseName

/* write code to define a get member function for the instructor's name */

```

**Fig. L 3.5** | GradeBook.cpp. (Part 1 of 2.)

## Lab Exercises

Name:

### Lab Exercise 2 — Modifying class GradeBook

```
31  /* write code to define a set member function for the instructor's name */
32
33  // display a welcome message and the instructor's name
34  void GradeBook::displayMessage()
35  {
36     // display a welcome message containing the course name
37     cout << "Welcome to the grade book for\n" << getCourseName() << "!"
38        << endl;
39     /* write code to output the instructor's name */
40  } // end function displayMessage
```

**Fig. L 3.5** | `GradeBook.cpp`. (Part 2 of 2.)

```
1  // Lab 2: GradeBookTest.cpp
2  // Test program for modified GradeBook class.
3  #include <iostream>
4  using namespace std;
5
6  // include definition of class GradeBook from GradeBook.h
7  #include "GradeBook.h"
8
9  // function main begins program execution
10 int main()
11 {
12    // create a GradeBook object; pass a course name and instructor name
13    GradeBook gradeBook(
14       "CS101 Introduction to C++ Programming" );
15
16    // display welcome message and instructor's name
17    gradeBook.displayMessage();
18
19    /* write code to change instructor's name and output changes */
20 } // end main
```

**Fig. L 3.6** | `GradeBookTest.cpp`.

### Problem-Solving Tips

1. In class `GradeBook`, declare a `string` data member to represent the instructor's name.
2. Declare a `public` *set* function for the instructor's name that does not return a value and takes a `string` as a parameter. In the body of the *set* function, assign the parameter's value to the data member that represents the instructor's name.
3. Declare a `public` *get* function that returns a `string` and takes no parameters. This member function should return the instructor's name.
4. Modify the constructor to take two `string` parameters. Assign the parameter that represents the instructor's name to the appropriate data member.
5. Add an output statement to member function `displayMessage` to output the value of the data member you declared earlier.
6. Be sure to follow the spacing and indentation conventions mentioned in the text.
7. If you have any questions as you proceed, ask your lab instructor for help.

## Lab Exercises

Name:

# Lab Exercise 3 — Creating an `Employee` Class

**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 3.7, Fig. L 3.8 and Fig. L 3.9)
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 3 of *C++ How to Program, Seventh Edition*. In this lab, you will practice:

- Creating a class definition.
- Declaring data members.
- Defining a constructor.
- Defining *set* and *get* functions.
- Writing a test application to demonstrate the capabilities of another class.

## Description of the Problem

Create a class called `Employee` that includes three pieces of information as data members—a first name (type `string`), a last name (type `string`) and a monthly salary (type `int`). [*Note:* In subsequent chapters, we'll use numbers that contain decimal points (e.g., 2.75)—called floating-point values—to represent dollar amounts.] Your class should have a constructor that initializes the three data members. Provide a *set* and a *get* function for each data member. If the monthly salary is not positive, set it to `0`. Write a test program that demonstrates class `Employee`'s capabilities. Create two `Employee` objects and display each object's yearly salary. Then give each `Employee` a 10 percent raise and display each `Employee`'s yearly salary again.

## Sample Output

```
Employee 1: Bob Jones; Yearly Salary: 34500
Employee 2: Susan Baker; Yearly Salary: 37800

Increasing employee salaries by 10%
Employee 1: Bob Jones; Yearly Salary: 37944
Employee 2: Susan Baker; Yearly Salary: 41580
```

## Lab Exercises

Name:

### Lab Exercise 3 — Creating an Employee Class

### Program Template

```
// Lab 3: Employee.h
// Employee class definition.

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

// Employee class definition
class Employee
{
public:
   /* Declare a constructor that has one parameter for each data member */
   /* Declare a set method for the employee's first name */
   /* Declare a get method for the employee's first name */
   /* Declare a set method for the employee's last name */
   /* Declare a get method for the employee's last name */
   /* Declare a set method for the employee's monthly salary */
   /* Declare a get method for the employee's monthly salary */
private:
   /* Declare a string data member for the employee's first name */
   /* Declare a string data member for the employee's last name */
   /* Declare an int data member for the employee's monthly salary */
}; // end class Employee
```

**Fig. L 3.7** | Employee.h.

```
// Lab 3: Employee.cpp
// Employee class member-function definitions.
#include <iostream>
using namespace std;

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

/* Define the constructor. Assign each parameter value to the appropriate data
   member. Write code that validates the value of salary to ensure that it is
   not negative. */

/* Define a set function for the first name data member. */

/* Define a get function for the first name data member. */

/* Define a set function for the last name data member. */

/* Define a get function for the last name data member. */

/* Define a set function for the monthly salary data member. Write code
   that validates the salary to ensure that it is not negative. */

/* Define a get function for the monthly salary data member. */
```

**Fig. L 3.8** | Employee.cpp.

## Lab Exercises

Name:

### Lab Exercise 3 — Creating an Employee Class

```
1  // Lab 3: EmployeeTest.cpp
2  // Create and manipulate two Employee objects.
3  #include <iostream>
4  using namespace std;
5
6  #include "Employee.h" // include definition of class Employee
7
8  // function main begins program execution
9  int main()
10 {
11    /* Create two Employee objects and assign them to Employee variables. */
12
13    /* Output the first name, last name and salary for each Employee. */
14
15    /* Give each Employee a 10% raise. */
16
17    /* Output the first name, last name and salary of each Employee again. */
18 } // end main
```

**Fig. L 3.9** | `EmployeeTest.cpp` .

### Problem-Solving Tips

1. Class `Employee` should declare three data members.
2. The constructor must declare three parameters, one for each data member. The value for the salary should be validated to ensure it is not negative.
3. Declare a `public` *set* and *get* member functions for each data member. The *set* functions should not return values and should each specify a parameter of a type that matches the corresponding data member (`string` for first name and last name, `int` for the salary). The *get* functions should receive no parameters and should specify a return type that matches the corresponding data member.
4. When you call the constructor from the `main` function, you must pass it three arguments that match the parameters declared by the constructor.
5. Giving each employee a raise will require a call to the *get* function for the salary to obtain the current salary and a call to the *set* function for the salary to specify the new salary.
6. Be sure to follow the spacing and indentation conventions mentioned in the text.
7. If you have any questions as you proceed, ask your lab instructor for help.

## Lab Exercises

Name:

### Debugging

**Name:** _____ **Date:** _____

**Section:** _____

The program in this section does not compile. Fix all the compilation errors so that the program will compile successfully. Once the program compiles, execute the program, and compare its output with the sample output; then eliminate any logic errors that may exist. The sample output demonstrates what the program's output should be once the code is corrected. The source code is available from the Companion Website for *C++ How to Program, Seventh Edition* at `www.pearsonhighered.com/deitel/`.

### Sample Output

```
Created John Smith, age 19
Happy Birthday to John Smith
```

### Broken Code

```
1  // Person.h
2  // Creates and manipulates a person with a first name, last name and age
3
4  #include <string>
5  using namespace std;
6
7  class Person
8  {
9  public:
10    void Person( string, string, int )
11    string getFirstName( string )
12    setFirstName( string )
13    string getLastName()
14    void setLastName( string )
15    int getAge()
16    void setAge( int )
17 private:
18    string firstName;
19    string lastName;
20    int age;
21 }; // end class Person
```

**Fig. L 3.10** | `Person.h`.

```
1  // Person.cpp
2  // Creates and manipulates a person with a first name, last name and age
3
4  #include "Person.h"
5
6  void Person::Person( string first, string last, int years )
7  {
```

**Fig. L 3.11** | `Person.cpp` (Part 1 of 2).

## Lab Exercises

Name:

### Debugging

```
8     firstName = first;
9     lastName = last;
10    if ( years < 0 )
11       age = years;
12 } // end Person constructor
13
14 String Person::getFirstName( string FirstName )
15 {
16    return firstName;
17 } // end function getFirstName
18
19 Person::setFirstName( string first )
20 {
21    firstName = first;
22 } // end function setFirstName
23
24 String Person::getLastName()
25 {
26    return;
27 } // end function getLastName
28
29 void Person::setLastName( string last )
30 {
31    lastName = last;
32 } // end function setLastName
33
34 int Person::getAge()
35 {
36    return years;
37 } // end function getAge
38
39 void Person::setAge( int years )
40 {
41    if ( years > 0 )
42       age = years;
43 } // end function setAge
```

**Fig. L 3.11** | Person.cpp. (Part 2 of 2.)

```
1  // PersonTest.cpp
2  // Test application for the Person class
3
4  #include <iostream>
5  using namespace std;
6
7  #include "Person.h"
8
9  int main()
10 {
11    Person person = ( "John", "Smith", 19 );
12
13    cout << "Created " << getFirstName() << " " << getLastName() << ", age "
14      << getAge() << endl;
15
```

**Fig. L 3.12** | PersonTest.cpp. (Part 1 of 2.)

## Lab Exercises

Name:

### Debugging

```
16     person.setAge = person.getAge() + 1;
17     cout << "Happy Birthday to " << person.getFirstName() << " "
18        << person.getLastName() << endl;
19  } // end main
```

**Fig. L 3.12** | `PersonTest.cpp`. (Part 2 of 2.)

# 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 successfully completed the *Prelab Activities* and *Lab Exercises*.

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

1. Write an empty class definition for a class named `Student` and include header file `<string>` so this class can use `string` objects. Also add a `using` directive so that `string` will not need to be preceded by `std::` everytime it is used.

2. Declare five data members in the class from *Coding Exercise 1*: A `string` variable for the first name, a `string` variable for the last name and three `int` variables that are used to store a student's exam grades.

## Postlab Activities

Name:

### Coding Exercises

3. In the class from *Coding Exercise 2*, declare a constructor that takes five parameters—two `Strings` and three `ints`. Implement this constructor in a separate source file and be sure to include the `Student` header file in the `Student` source file.

4. Modify the class from *Coding Exercise 3* to include a *get* and a *set* function for each of the data members in the class. Declare each *get* or *set* function in the header file and implement it in the source file.

## Postlab Activities

Name:

### Coding Exercises

5. Modify the class from *Coding Exercise 4* to include a `getAverage` member function that calculates and returns the average of the three exam grades.

6. Declare a `main` function to test the capabilities of your new `Student` class from *Coding Exercise 5*. Begin by including the appropriate header files, `using` directives and a return statement.

## Postlab Activities

Name:

### Coding Exercises

7. Add statements to the `main` function of *Coding Exercise 6* to test class `Student`'s *get* functions. Create a `student` and output the name and average for the `student`.

8. Add statements to the `main` function of *Coding Exercise 7* that test the *set* functions of class `Student`, then output the new name and average of the `Student` object to show that the *set* functions worked correctly.

## 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 or sample output is provided for each problem in order to aid you in your programming.

1. Create a class called `Invoice` that a hardware store might use to represent an invoice for an item sold at the store. An `Invoice` should include four pieces of information as data members—a part number (type `string`), a part description (type `string`), a quantity of the item being purchased (type `int`) and a price per item (`int`). [*Note:* In subsequent chapters, we'll use numbers that contain decimal points (e.g., 2.75)—called floating-point values—to represent dollar amounts.] Your class should have a constructor that initializes the four data members. Provide a *set* and a *get* method for each data member. In addition, provide a member function named `getInvoiceAmount` that calculates the invoice amount (i.e., multiplies the quantity by the price per item), then returns the amount as an `int` value. If the quantity is not positive, it should be set to `0`. If the price per item is not positive, it should be set to `0`. Write a test program that demonstrates class `Invoice`'s capabilities.

**Hints:**

- To solve this exercise, mimic your solutions to *Lab Exercises 1–3*.
- The input values for the quantity and the price per item must be validated before they can be used to set the corresponding data members. This should be done both in the constructor and in the appropriate *set* functions.
- The function header for `getInvoiceAmount` should be `int getInvoiceAmount()`.
- Your output should appear as follows:

```
Part number: 12345
Description: Hammer
Quantity: 100
Price per item: $5
Invoice amount: $500

quantity cannot be negative. quantity set to 0.

Invoice data members modified.

Part number: 123456
Description: Saw
Quantity: 0
Price per item: $10
Invoice amount: $0
```

## Postlab Activities

Name:

### Programming Challenges

2. Create a class called `Date` that includes three pieces of information as data members—a month (type `int`), a day (type `int`) and a year (type `int`). Your class should have a constructor with three parameters that uses the parameters to initialize the three data members. For the purpose of this programming challenge, assume that the values provided for the year and day are correct, but ensure that the month value is in the range 1–12; if it is not, set the month to 1. Provide a *set* and a *get* function for each data member. Provide a member function `displayDate` that displays the month, day and year separated by forward slashes (/). Write a test program that demonstrates class `Date`'s capabilities.

**Hints:**

- To solve this exercise, mimic your solutions to *Lab Exercises 1–3*.
- For the purpose of this programming challenge, it is not necessary to validate the year and day values passed to the constructor or the *set* function, but should still validate the month values.
- Your output should appear as follows:

```
Month: 5
Day: 6
Year: 1981

Original date:
5/6/1981

New date:
1/1/2005
```