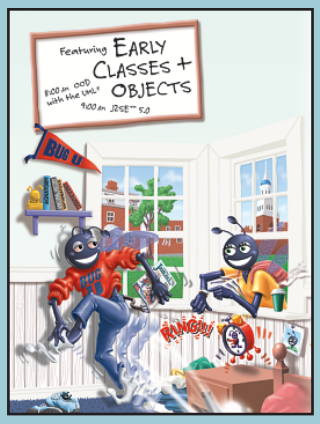
3

Introduction to Classes and Objects



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

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

OBJECTIVES

In this chapter you will learn:

- What classes, objects, methods and instance variables are.
- How to declare a class and use it to create an object.
- How to declare methods in a class to implement the class's behaviors.
- How to declare instance variables in a class to implement the class's attributes.
- How to call an object's method to make that method perform its task.
- The differences between instance variables of a class and local variables of a method.
- How to use a constructor to ensure that an object's data is initialized when the object is created.
- The differences between primitive and reference types.

You will see something new.
Two things. And I call them
Thing One and Thing Two.
—Dr. Theodor Seuss Geisel

Assignment Checklist

Name: _____ Date: _____

Section: _____

Exercises	Assigned: Circle assignments	Date Due
Prelab Activities		
Matching	YES NO	
Fill in the Blank	16, 17, 18, 19, 20, 21, 22, 23, 24, 25	
Short Answer	26, 27, 28, 29, 30	
Programming Output	31, 32, 33, 34, 35	
Correct the Code	36, 37, 38, 39	
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, 9	
Programming Challenges	1, 2	

Prelab Activities

Matching

Name: _____ **Date:** _____

Section: _____

After reading Chapter 3 of *Java How to Program: Sixth 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. field	a) Used in a class instance creation expression to create an instance of a class.
___	2. calling method	b) Primitive type that represents a single-precision floating-point number.
___	3. reference	c) Causes Java to execute a method.
___	4. `new` keyword	d) Also known as an instance variable.
___	5. `public` method	e) A method that assigns a value to a `private` instance variable.
___	6. class declaration	f) A variable that refers to an object contains one of these as its value.
___	7. fully qualified class name	g) A method that is accessible from outside of the class in which it is declared.
___	8. method call	h) Default initial value of a reference-type variable.
___	9. parameter	i) Additional information a method requires to help it perform its task.
___	10. *set* method	j) Primitive type that represents a double-precision floating-point number.
___	11. default constructor	k) The compiler provides one of these for a class that does not declare any.
___	12. client of an object or a class	l) Encompasses all of the attributes and behaviors of a class.
___	13. `double`	m) Can be used to access a class if the class is not imported.
___	14. `null`	n) A class that calls any of an object's or class's methods.
___	15. `float`	o) Receives the return value from a method.

Prelab Activities

Name:

Fill in the Blank

Name: _____ **Date:** _____

Section: _____

Fill in the blanks for each of the following statements:

16. Each method can specify _____ that represent additional information the method requires to perform its task correctly.

17. Declaring instance variables with access modifier _____ is known as information hiding.

18. Instance variables of the numeric primitive types are initialized to _____ and instance variables of type `boolean` are initialized to _____.

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

20. An `import` declaration is not required if you always refer to a class with its _____.

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

22. The format specifier `%f` is used to output values of type _____ or _____.

23. Programs use variables of _____ to store the location of objects in the computer's memory.

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

25. Classes often provide `public` methods to allows clients of the class to _____ or _____ the values of `private` instance variables.

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.

26. List the parts of a method header and why each one is important.

27. How are constructors and methods similar? How are they different?

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

Prelab Activities

Name:

Short Answer

29. What types of declarations are contained within a class declaration?

30. Distinguish between a primitive-type variable and a reference-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.]

Use the following class definition for *Programming Output Exercises 31–35*.

```
1  public class Account
2  {
3     private double balance; // instance variable that stores the balance
4
5     // constructor
6     public Account( double initialBalance )
7     {
8        // validate that initialBalance is greater than 0.0;
9        // if it is not, balance is initialized to the default value 0.0
10       if ( initialBalance > 0.0 )
11          balance = initialBalance;
12    } // end Account constructor
13
14    // credit (add) an amount to the account
15    public void credit( double amount )
16    {
17       balance = balance + amount; // add amount to balance
18    } // end method credit
19
20    // return the account balance
21    public double getBalance()
22    {
23       return balance; // gives the value of balance to the calling method
24    } // end method getBalance
25
26 } // end class Account
```

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

```
1  public static void main( String args[] )
2  {
3     Account account1 = new Account( 35.50 );
4
5     System.out.printf( "account1 balance: $%.2f\n", account1.getBalance() );
6  } // end main
```

Your answer:

Prelab Activities

Name:

Programming Output

32. What is output by the following `main` method?

```
1  public static void main( String args[] )
2  {
3     Account account1 = new Account( -20.17 );
4
5     System.out.printf( "account1 balance: $%.2f\n", account1.getBalance() );
6  } // end main
```

Your answer:

33. What is output by the following `main` method?

```
1  public static void main( String args[] )
2  {
3     Account account1 = new Account( 15.33 );
4
5     System.out.printf( "account1 balance: $%.2f\n", account1.getBalance() );
6     System.out.println( "adding $2.53 to account1 balance" );
7
8     account1.credit( 2.53 );
9     System.out.printf( "account1 balance: $%.2f\n", account1.getBalance() );
10 } // end main
```

Your answer:

Prelab Activities

Name:

Programming Output

34. What is output by the following `main` method?

```
1  public static void main( String args[] )
2  {
3     Account account1 = new Account( 27.70 );
4
5     System.out.printf( "account1 balance: $%.2f\n", account1.getBalance() );
6     System.out.println( "adding $3.75 to account1 balance" );
7
8     account1.credit( 3.757 );
9     System.out.printf( "account1 balance: $%.2f\n", account1.getBalance() );
10 } // end main
```

Your answer:

35. What is output by the following `main` method?

```
1  public static void main( String args[] )
2  {
3     Account account1 = new Account( 7.99 );
4
5     System.out.printf( "account1 balance: $%.2f\n", account1.getBalance() );
6     System.out.println( "adding -$1.14 to account1 balance" );
7
8     account1.credit( -1.14 );
9     System.out.printf( "account1 balance: $%.2f\n", account1.getBalance() );
10 } // end main
```

Your answer:

Prelab Activities

Name:

Correct the Code

Name: _____ **Date:** _____

Section: _____

Determine if there is an error in each of the following program segments. If there is an error, specify whether it is a logic error or a 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*: There may be more than one error in each program segment.]

Use the following class definitions for *Correct the Code Exercises 36–39*.

```
1  // Lab 2: GradeBook.java
2  // GradeBook class with a constructor to initialize the course name.
3
4  public class GradeBook
5  {
6     private String courseName; // course name for this GradeBook
7
8     // constructor initializes courseName with String supplied as argument
9     public GradeBook( String name )
10    {
11       courseName = name; // initializes courseName
12    } // end constructor
13
14    // method to set the course name
15    public void setCourseName( String name )
16    {
17       courseName = name; // store the course name
18    } // end method setCourseName
19
20    // method to retrieve the course name
21    public String getCourseName()
22    {
23       return courseName;
24    } // end method getCourseName
25
26    // display a welcome message to the GradeBook user
27    public void displayMessage()
28    {
29       // this statement calls getCourseName to get the
30       // name of the course this GradeBook represents
31       System.out.printf( "Welcome to the grade book for\n%s!\n",
32          getCourseName() );
33    } // end method displayMessage
34
35 } // end class GradeBook
```

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

```
1  GradeBook gradeBook = Gradebook( "Introduction to Java", 25 );
```

Prelab Activities

Name:

Correct the Code

Your answer:

37. The following code segment should set the `GradeBook`'s course name:

```
1  setCourseName( gradeBook, "Advanced Java" )
```

Your answer:

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

```
1  Scanner input = new Scanner( System.in );
2
3  System.out.println( "Please enter the course name:" );
4  inputName = Scanner.readLine();
5
6  gradeBook.setCourseName();
```

Your answer:

Prelab Activities

Name:

Correct the Code

39. The following code segment should output the grade book's current course name:

```
System.out.printf( "The grade book's course name is: \n", gradeBook.courseName );
```

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 and Fig. L 3.2)
5. Problem-Solving Tips

The program template represents a complete working Java 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 Java 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 3 of *Java How to Program: Sixth Edition*. In this lab, you will practice:

- Creating methods.
- Invoking methods and receiving return values from methods.
- Testing a condition using an `if` statement.
- Outputting variables with a `printf` statement.

Description of the Problem

Modify class `Account` (Fig. L 3.1) to provide a method 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 method should print a message indicating `"Debit amount exceeded account balance."` Modify class `AccountTest` (Fig. L 3.2) to test method `debit`.

Sample Output

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

Enter withdrawal amount for account1: 25.67

subtracting 25.67 from account1 balance
account1 balance: $24.33
account2 balance: $0.00

Enter withdrawal amount for account2: 10.00

subtracting 10.00 from account2 balance
Debit amount exceeded account balance.
account1 balance: $24.33
account2 balance: $0.00
```

Lab Exercises

Name:

Lab Exercise 1 — Modifying Class Account

Program Template

```
1  // Lab 1: Account.java
2  // Account class with a constructor to
3  // initialize instance variable balance.
4
5  public class Account
6  {
7     private double balance; // instance variable that stores the balance
8
9     // constructor
10    public Account( double initialBalance )
11    {
12       // validate that initialBalance is greater than 0.0;
13       // if it is not, balance is initialized to the default value 0.0
14       if ( initialBalance > 0.0 )
15          balance = initialBalance;
16    } // end Account constructor
17
18    // credit (add) an amount to the account
19    public void credit( double amount )
20    {
21       balance = balance + amount; // add amount to balance
22    } // end method credit
23
24    /* write code to declare method debit. */
25
26    // return the account balance
27    public double getBalance()
28    {
29       return balance; // gives the value of balance to the calling method
30    } // end method getBalance
31
32 } // end class Account
```

Fig. L 3.1 | Account.java.

```
1  // Lab 1: AccountTest.java
2  // Create and manipulate an Account object.
3  import java.util.Scanner;
4
5  public class AccountTest
6  {
7     // main method begins execution of Java application
8     public static void main( String args[] )
9     {
10       Account account1 = new Account( 50.00 ); // create Account object
11       Account account2 = new Account( -7.53 ); // create Account object
12
13       // display initial balance of each object
14       System.out.printf( "account1 balance: $%.2f\n",
15          account1.getBalance() );
16       System.out.printf( "account2 balance: $%.2f\n\n",
17          account2.getBalance() );
18
```

Fig. L 3.2 | AccountTest.java. (Part 1 of 2.)

Lab Exercises

Name:

Lab Exercise 1 — Modifying Class Account

```
19         // create Scanner to obtain input from command window
20         Scanner input = new Scanner( System.in );
21         double withdrawalAmount; // withdrawal amount read from user
22
23         System.out.print( "Enter withdrawal amount for account1: " );
24         withdrawalAmount = input.nextDouble(); // obtain user input
25         System.out.printf( "\nsubtracting %.2f from account1 balance\n",
26            withdrawalAmount );
27         /* write code to withdraw money from account */
28
29         // display balances
30         System.out.printf( "account1 balance: $%.2f\n",
31            account1.getBalance() );
32         System.out.printf( "account2 balance: $%.2f\n\n",
33            account2.getBalance() );
34
35         System.out.print( "Enter withdrawal amount for account2: " );
36         withdrawalAmount = input.nextDouble(); // obtain user input
37         System.out.printf( "\nsubtracting %.2f from account2 balance\n",
38            withdrawalAmount );
39         /* write code to withdraw from account */
40
41         // display balances
42         System.out.printf( "account1 balance: $%.2f\n",
43            account1.getBalance() );
44         System.out.printf( "account2 balance: $%.2f\n",
45            account2.getBalance() );
46      } // end main
47
48   } // end class AccountTest
```

Fig. L 3.2 | `AccountTest.java`. (Part 2 of 2.)

Problem-Solving Tips

1. Declare `public` method `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 method `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. 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 3.3 and Fig. L 3.4)
5. Problem-Solving Tips

The program template represents a complete working Java 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 Java 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 3 of *Java How to Program: Sixth Edition*. In this lab, you will practice:

- Declaring an instance variable.
- Providing a *set* method to modify an instance variable's value.
- Declaring methods with parameters.

Description of the Problem

Modify class `GradeBook` (Fig. L 3.3). Include a second `String` instance variable that represents the name of the course's instructor. Provide a *set* method to change the instructor's name and a *get* method to retrieve it. Modify the constructor to specify two parameters—one for the course name and one for the instructor's name. Modify method `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 3.4) to demonstrate the class's new capabilities.

Sample Output

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

Changing instructor name to Judy Jones

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

Lab Exercises

Name:

Lab Exercise 2 — Modifying Class GradeBook

Program Template

```
1  // Lab 2: GradeBook.java
2  // GradeBook class with a constructor to initialize the course name.
3
4  public class GradeBook
5  {
6     private String courseName; // course name for this GradeBook
7     /* write code to declare a second String instance variable */
8
9     // constructor initializes courseName with String supplied as argument
10    public GradeBook( String name )
11    {
12       courseName = name; // initializes courseName
13    } // end constructor
14
15    // method to set the course name
16    public void setCourseName( String name )
17    {
18       courseName = name; // store the course name
19    } // end method setCourseName
20
21    // method to retrieve the course name
22    public String getCourseName()
23    {
24       return courseName;
25    } // end method getCourseName
26
27    /* write code to declare a get and a set method for the instructor's name */
28
29    // display a welcome message to the GradeBook user
30    public void displayMessage()
31    {
32       // this statement calls getCourseName to get the
33       // name of the course this GradeBook represents
34       System.out.printf( "Welcome to the grade book for\n%s!\n",
35          getCourseName() );
36       /* write code to output the instructor's name */
37    } // end method displayMessage
38
39 } // end class GradeBook
```

Fig. L 3.3 | GradeBook.java.

```
1  // Lab 2: GradeBookTest.java
2  // GradeBook constructor used to specify the course name at the
3  // time each GradeBook object is created.
4
5  public class GradeBookTest
6  {
7     // main method begins program execution
8     public static void main( String args[] )
9     {
10       // create GradeBook object
11       GradeBook gradeBook1 = new GradeBook(
12          "CS101 Introduction to Java Programming" );
```

Fig. L 3.4 | GradeBookTest.java. (Part 1 of 2.)

Lab Exercises

Name:

Lab Exercise 2 — Modifying Class GradeBook

```
13
14        gradeBook1.displayMessage(); // display welcome message
15
16        /* write code to change instructor's name and output changes */
17
18     } // end main
19
20  } // end class GradeBookTest
```

Fig. L 3.4 | `GradeBookTest.java`. (Part 2 of 2.)

Problem-Solving Tips

1. In class `GradeBook`, declare a `String` instance variable to represent the instructor's name.
2. Declare a `public` *set* method for the instructor's name that does not return a value and takes a `String` as a parameter. In the body of the *set* method, assign the parameter's value to the variable that represents the instructor's name.
3. Declare a `public` *get* method that returns a `String` and takes no parameters. This method 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 instance variable.
5. Add a `System.out.printf` statement to method `displayMessage` to output the value of the instance variable you declared earlier.
6. 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 3.5 and Fig. L 3.6)
5. Problem-Solving Tips

The program template represents a complete working Java 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 Java 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 3 of *Java How to Program: Sixth Edition*. In this lab, you will practice:

- Creating a class declaration.
- Declaring instance variables.
- Declaring a constructor.
- Declaring *set* and *get* methods.
- Writing a test application to demonstrate the capabilities of another class.

Description of the Problem

Using only programming techniques from this chapter and Chapter 2 of *Java How to Program: Sixth Edition*, create a class called `Employee` that includes three pieces of information as instance variables—a first name (type `String`), a last name (type `String`) and a monthly salary (type `double`). Your class should have a constructor that initializes the three instance variables. Provide a *set* and a *get* method for each instance variable. If the monthly salary is not positive, set it to `0.0`. Write a test application named `EmployeeTest` that demonstrates class `Employee`'s capabilities. Create two `Employee` objects and display the yearly salary for each `Employee`. Then give each `Employee` a 10% raise and display each `Employee`'s yearly salary again.

Sample Output

```
Employee 1: Bob Jones; Yearly Salary: 34500.00
Employee 2: Susan Baker; Yearly Salary: 37809.00

Increasing employee salaries by 10%
Employee 1: Bob Jones; Yearly Salary: 37950.00
Employee 2: Susan Baker; Yearly Salary: 41589.90
```

Lab Exercises

Name:

Lab Exercise 3 — Creating an Employee Class

Program Template

```
1  // Lab 3: Employee.java
2  // Employee class.
3
4  /* Begin class declaration of Employee class. */
5
6     /* Declare three instance variables here. */
7
8     /* Add a constructor that declares a parameter for each instance variable. Assign
9        each parameter value to the appropriate instance variable. Write code that
10       validates the value of salary to ensure that it is not negative. */
11
12    /* Declare set and get methods for the first name instance variable. */
13
14    /* Declare set and get methods for the last name instance variable. */
15
16    /* Declare set and get methods for the monthly salary instance variable. Write code
17       that validates the salary to ensure that it is not negative. */
18
19 /* End class declaration of Employee class. */
```

Fig. L 3.5 | `Employee.java`.

```
1  // Lab 3: EmployeeTest.java
2  // Application to test class Employee.
3
4  /* Begin class declaration of EmployeeTest class. */
5
6     /* Begin main method declaration. */
7
8     / * Create two Employee objects and assign them to Employee variables. */
9
10    /* Output the first name, last name and salary for each Employee. */
11
12    /* Give each Employee a 10% raise. */
13
14    /* Output the first name, last name and salary of each Employee again. */
15
16    /* End main method declaration */
17
18 /* End class declaration of EmployeeTest class. */
```

Fig. L 3.6 | `EmployeeTest.java`.

Problem-Solving Tips

1. Class `Employee` should declare three instance variables.
2. The constructor must declare three parameters, one for each instance variable. The value for the salary should be validated to ensure it is not negative.
3. Declare a `public` *set* and *get* method for each instance variable. The *set* methods should not return values and should each specify a parameter of a type that matches the corresponding instance variable (`String` for first name and last name, `double` for the salary). The *get* methods should receive no parameters and should specify a return type that matches the corresponding instance variable.

Lab Exercises

Name:

Lab Exercise 3 — Creating an `Employee` Class

4. When you call the constructor from the test class, 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* method for the salary to obtain the current salary and a call to the *set* method for the salary to specify the new salary.
6. A salary is a dollar amount, so you should output the salary using the `%.2f` specifier to provide two digits of precision.
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 program's code is corrected. The source code is available at the Web sites `www.deitel.com` and `www.prenhall.com/deitel`.

Sample Output

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

Broken Code

```
1  // Person.java
2  // Creates and manipulates a person with a first name, last name and age
3
4  public class Person
5  {
6     private String firstName;
7     private String lastName;
8     private int age;
9
10    public void Person( String first, String last, int years )
11    {
12       firstName = first;
13       lastName = last;
14
15       if ( years < 0 )
16          age = years;
17    } // end Person constructor
18
19    public String getFirstName( String FirstName )
20    {
21       return firstName;
22    } // end method getFirstName
23
24    public setFirstName( String first )
25    {
26       firstName = first;
27    } // end method setFirstName
28
29    public String getLastName()
30    {
31       return;
32    } // end method getLastName
```

Fig. L 3.7 | `Person.java`. (Part 1 of 2.)

Lab Exercises

Name:

Debugging

```
33
34    public void setLastName( String last )
35    {
36       lastName = last;
37    } // end method setLastName
38
39    public int getAge()
40    {
41       return years;
42    } // end method getAge
43
44    public void setAge( int years )
45    {
46       if ( years > 0 )
47          age = years;
48    } // end method setAge
49 } // end class Person
```

Fig. L 3.7 | Person.java. (Part 2 of 2.)

```
1  // PersonTest.java
2  // Test application for the Person class
3
4  public class PersonTest
5  {
6     public static void main( String args[] )
7     {
8        Person person = Person( "John", "Smith", 19 );
9
10       System.out.printf( "Created %s %s, age %d\n",
11          getFirstName(), getLastName(), getAge() );
12
13       person.setAge = person.getAge() + 1;
14       System.out.printf( "Happy Birthday to %s %s\n",
15          person.getFirstName(), person.getLastName() );
16    } // end main
17 } // end class PersonTest
```

Fig. L 3.8 | PersonTest.java.

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 declaration for a class named `Student`.

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

3. In the class from *Coding Exercise 2*, declare a constructor that takes five parameters—two `String`s and three `double`s. Use these parameters to initialize the instance variables declared earlier.

Postlab Activities

Name:

Coding Exercises

4. Modify the class from *Coding Exercise 3* to include a *get* and a *set* method for each of the instance variables in the class.

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

6. Declare an empty test class to use the capabilities of your new `Student` class from *Coding Exercise 5*.

7. In the class from *Coding Exercise 6*, declare a `main` method that creates an instance of class `Student`.

Postlab Activities

Name:

Coding Exercises

8. Add statements to the main method of *Coding Exercise 7* to test class `Student`'s *get* methods. Output the name and average for the student.

9. Add statements to the `main` method of *Coding Exercise 8* that test the *set* methods of class `Student`, then output the new name and average of the `Student` object to show that the *set* methods 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 Java 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 or sample outputs are provided for each problem 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 instance variables—a part number (type `String`), a part description (type `String`), a quantity of the item being purchased (type `int`) and a price per item (`double`). Your class should have a constructor that initializes the four instance variables. Provide a *set* and a *get* method for each instance variable. In addition, provide a method named `getInvoiceAmount` that calculates the invoice amount (i.e., multiplies the quantity by the price per item), then returns the amount as a `double` 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 `0.0`. Write a test application named `InvoiceTest` that demonstrates class `Invoice`'s capabilities.

Hints:

- To solve this exercise, mimic your solutions to *Lab Exercises 1–3*.
- Validate the input values for the quantity and the price per item in the constructor and in the appropriate *set* methods.
- Your output should appear as follows:

```
Original invoice information
Part number: 1234
Description: Hammer
Quantity: 2
Price: 14.95
Invoice amount: 29.90

Updated invoice information
Part number: 001234
Description: Yellow Hammer
Quantity: 3
Price: 19.49
Invoice amount: 58.47

Original invoice information
Part number: 5678
Description: Paint Brush
Quantity: 0
Price: 0.00
Invoice amount: 0.00

Updated invoice information
Part number: 5678
Description: Paint Brush
Quantity: 3
Price: 9.49
Invoice amount: 28.47
```

Postlab Activities

Name:

Programming Challenges

2. Create a class called `Date` that includes three pieces of information as instance variables—a month (type `int`), a day (type `int`) and a year (type `int`). Your class should have a constructor that initializes the three instance variables and assumes that the values provided are correct. Provide a *set* and a *get* method for each instance variable. Provide a method `displayDate` that displays the month, day and year separated by forward slashes (/). Write a test application named `DateTest` that demonstrates class `Date`'s capabilities.

Hints:

- To solve this exercise, mimic your solutions to *Lab Exercises 1–3*.
- For the purpose of this chapter, it is not necessary to validate the values passed to the constructor or the *set* methods.
- Your output should appear as follows:

```
The initial date is: 7/4/2004
Date with new values is: 11/1/2003
```