3

Introduction to Classes and Objects

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

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 methods to make those methods perform their tasks.
- 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.

© Copyright 1992-2007 Pearson Education, Inc., Upper Saddle River, NJ. All rights reserved.

Self-Review Exercises

3.1 Fill in the blanks in each of the following:

a) A house is to a blueprint as a(n) _____ is to a class.

ANS: object.

b) Each class declaration that begins with keyword _____ must be stored in a file that has exactly the same name as the class and ends with the `.java` file-name extension.

ANS: `public`.

c) Every class declaration contains keyword _____ followed immediately by the class's name.

ANS: `class`.

d) Keyword _____ creates an object of the class specified to the right of the keyword.

ANS: `new`.

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

ANS: type, name.

f) By default, classes that are compiled in the same directory are considered to be in the same package—known as the _____.

ANS: default package.

g) When each object of a class maintains its own copy of an attribute, the field that represents the attribute is also known as a(n) _____.

ANS: instance variable.

h) Java provides two primitive types for storing floating-point numbers in memory—_____ and _____.

ANS: `float`, `double`.

i) Variables of type `double` represent _____ floating-point numbers.

ANS: double-precision.

j) `Scanner` method _____ returns a `double` value.

ANS: `nextDouble`.

k) Keyword `public` is a(n) _____.

ANS: access modifier.

l) Return type _____ indicates that a method will perform a task but will not return any information when it completes its task.

ANS: `void`.

m) `Scanner` method _____ reads characters until a newline character is encountered, then returns those characters as a `String`.

ANS: `nextLine`.

n) Class `String` is in package _____.

ANS: `java.lang`.

o) A(n) _____ is not required if you always refer to a class with its fully qualified class name.

ANS: `import` declaration.

p) A(n) _____ is a number with a decimal point, such as 7.33, 0.0975 or 1000.12345.

ANS: floating-point number.

q) Variables of type `float` represent _____ floating-point numbers.

ANS: single-precision.

r) The format specifier _____ is used to output values of type `float` or `double`.

ANS: `%f`.

s) Types in Java are divided into two categories—_____ types and _____ types.

ANS: primitive types, reference types.

3.2 State whether each of the following is *true* or *false*. If *false*, explain why.

© Copyright 1992-2007 Pearson Education, Inc., Upper Saddle River, NJ. All rights reserved.

a) By convention, method names begin with an uppercase first letter and all subsequent words in the name begin with a capital first letter.

ANS: False. By convention, method names begin with a lowercase first letter and all subsequent words in the name begin with a capital first letter.

b) An `import` declaration is not required when one class in a package uses another in the same package.

ANS: True.

c) Empty parentheses following a method name in a method declaration indicate that the method does not require any parameters to perform its task.

ANS: True.

d) Variables or methods declared with access modifier `private` are accessible only to methods of the class in which they are declared.

ANS: True.

e) A primitive-type variable can be used to invoke a method.

ANS: False. A primitive-type variable cannot be used to invoke a method—a reference to an object is required to invoke the object's methods.

f) Variables declared in the body of a particular method are known as instance variables and can be used in all methods of the class.

ANS: False. Such variables are called local variables and can be used only in the method in which they are declared.

g) Every method's body is delimited by left and right braces (`{` and `}`).

ANS: True.

h) Primitive-type local variables are initialized by default.

ANS: False. Primitive-type instance variables are initialized by default. Each local variable must explicitly be assigned a value.

i) Reference-type instance variables are initialized by default to the value `null`.

ANS: True.

j) Any class that contains `public static void main( String args[] )` can be used to execute an application.

ANS: True.

k) The number of arguments in the method call must match the number of parameters in the method declaration's parameter list.

ANS: True.

l) Floating-point values that appear in source code are known as floating-point literals and are type `float` by default.

ANS: False. Such literals are of type `double` by default.

3.3 What is the difference between a local variable and a field?

ANS: A local variable is declared in the body of a method and can be used only from the point at which it is declared through the end of the method declaration. A field is declared in a class, but not in the body of any of the class's methods. Also, fields are accessible to all methods of the class. (We will see an exception to this in Chapter 8, Classes and Objects: A Deeper Look.)

3.4 Explain the purpose of a method parameter. What is the difference between a parameter and an argument?

ANS: A parameter represents additional information that a method requires to perform its task. Each parameter required by a method is specified in the method's declaration. An argument is the actual value for a method parameter. When a method is called, the argument values are passed to the method so that it can perform its task.

© Copyright 1992-2007 Pearson Education, Inc., Upper Saddle River, NJ. All rights reserved.

Exercises

3.5 What is the purpose of keyword `new`? Explain what happens when this keyword is used in an application.

ANS: The purpose of keyword `new` is to create an object of a class. When keyword `new` is used in an application, first a new object of the class to the right of `new` is created, then the class's constructor is called to ensure that the object is initialized properly.

3.6 What is a default constructor? How are an object's instance variables initialized if a class has only a default constructor?

ANS: A default constructor is a constructor provided by the compiler when the programmer does not specify any constructors in the class. When a class has only the default constructor, its instance variables are initialized to their default values. Variables of types `char`, `byte`, `short`, `int`, `long`, `float` and `double` are initialized to `0`, variables of type `boolean` are initialized to `false`, and reference-type variables are initialized to `null`.

3.7 Explain the purpose of an instance variable.

ANS: A class provides an instance variable (or several instance variables) when each object of the class must maintain information separately from all other objects of the class. For example, a class called `Account` that represents a bank account provides an instance variable to represent the balance of the account. Each `Account` object maintains its own balance, but does not know the balances of the bank's other accounts.

3.8 Most classes need to be imported before they can be used in an application. Why is every application allowed to use classes `System` and `String` without first importing them?

ANS: Classes `System` and `String` are both in package `java.lang`, which is implicitly imported into every Java source-code file.

3.9 Explain how a program could use class `Scanner` without importing the class from package `java.util`.

ANS: If every use of a class's name in a program is fully qualified, there is no need to import the class. A class's fully qualified name consists of the class's package followed by the class name. For example, a program could use class `Scanner` without importing it if every use of `Scanner` in the program is specified as `java.util.Scanner`.

3.10 Explain why a class might provide a *set* method and a *get* method for an instance variable.

ANS: An instance variable is typically declared `private` in a class so that only the methods of the class in which the instance variable is declared can manipulate the variable. In some cases, it may be necessary for an application to modify the `private` data. For example, the owner of a bank account should be able to deposit or withdraw funds and check the account's balance. A class's designer can provide `public` *set* and *get* methods that enable an application to specify the value for, or retrieve the value of, a particular object's `private` instance variable.

3.11 Modify class `GradeBook` (Fig. 3.10) as follows:

a) Include a second `String` instance variable that represents the name of the course's instructor.

b) Provide a *set* method to change the instructor's name and a *get* method to retrieve it.

c) Modify the constructor to specify two parameters—one for the course name and one for the instructor's name.

d) 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.

© Copyright 1992-2007 Pearson Education, Inc., Upper Saddle River, NJ. All rights reserved.

Use your modified class in a test application that demonstrates the class's new capabilities.

ANS:

```
1  // Exercise 3.11 Solution: 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     private String instructorName; // name of course's instructor
8
9     // constructor initializes courseName with String supplied as argument
10    public GradeBook( String course, String instructor )
11    {
12       courseName = course; // initializes courseName
13       instructorName = instructor; // initializes instructorName
14    } // end constructor
15
16    // method to set the course name
17    public void setCourseName( String name )
18    {
19       courseName = name; // store the course name
20    } // end method setCourseName
21
22    // method to retrieve the course name
23    public String getCourseName()
24    {
25       return courseName;
26    } // end method getCourseName
27
28    // method to set the instructor name
29    public void setInstructorName( String name )
30    {
31       instructorName = name; // store the course name
32    } // end method setInstructorName
33
34    // method to retrieve the instructor name
35    public String getInstructorName()
36    {
37       return instructorName;
38    } // end method getInstructorName
39
40    // display a welcome message to the GradeBook user
41    public void displayMessage()
42    {
43       // this statement calls getCourseName to get the
44       // name of the course this GradeBook represents
45       System.out.printf( "Welcome to the grade book for\n%s!\n",
46          getCourseName() );
47       System.out.printf( "This course is presented by: %s\n",
48          getInstructorName() );
49    } // end method displayMessage
50
51 } // end class GradeBook
```

© Copyright 1992-2007 Pearson Education, Inc., Upper Saddle River, NJ. All rights reserved.

```
1  // Exercise 3.11 Solution: 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", "Sam Smith" );
13
14       gradeBook1.displayMessage(); // display welcome message
15
16       System.out.println( "\nChanging instructor name to Judy Jones\n" );
17       gradeBook1.setInstructorName( "Judy Jones" );
18
19       gradeBook1.displayMessage(); // display welcome message
20    } // end main
21
22 } // end class GradeBookTest
```

```
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
```

3.12 Modify class `Account` (Fig. 3.13) 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. 3.14) to test method `debit`.

ANS:

```
1  // Exercise 3.12 Solution: 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       // if initialBalance is not greater than 0.0,
13       // balance is still initialized to 0.0 by default
```

© Copyright 1992-2007 Pearson Education, Inc., Upper Saddle River, NJ. All rights reserved.

```
14          if ( initialBalance > 0.0 )
15             balance = initialBalance;
16       } // end Account constructor
17
18       // credits (adds) 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       // debits (subtracts) an amount from the account
25       public void debit( double amount )
26       {
27          if ( amount > balance )
28             System.out.println( "Debit amount exceeded account balance." );
29
30          if ( amount <= balance )
31             balance = balance - amount; // subtract amount to balance
32       } // end method credit
33
34       // returns the account balance
35       public double getBalance()
36       {
37          return balance; // gives the value of balance to the calling method
38       } // end method getBalance
39
40    } // end class Account
```

```
1   // Exercise 3.12 Solution: 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 ); v
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
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",
```

© Copyright 1992-2007 Pearson Education, Inc., Upper Saddle River, NJ. All rights reserved.

```
26           withdrawalAmount );
27        account1.debit( withdrawalAmount ); // subtract amount from account1
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        account2.debit( withdrawalAmount ); // subtract amount from account2
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
```

```
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
```

3.13 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.

ANS:

```
1  // Exercises 3.13 Solution: Invoice.java
2  // Invoice class.
```

© Copyright 1992-2007 Pearson Education, Inc., Upper Saddle River, NJ. All rights reserved.

```
3
4  public class Invoice
5  {
6     private String partNumber;
7     private String partDescription;
8     private int quantity;
9     private double pricePerItem;
10
11    // four-argument constructor
12    public Invoice( String part, String description, int count,
13       double price )
14    {
15       partNumber = part;
16       partDescription = description;
17
18       if ( count > 0 ) // determine whether count is positive
19          quantity = count; // valid count assigned to quantity
20
21       if ( price > 0.0 ) // determine whether price is positive
22          pricePerItem = price; // valid price assigned to pricePerItem
23    } // end four-argument Invoice constructor
24
25    // set part number
26    public void setPartNumber( String part )
27    {
28       partNumber = part;
29    } // end method setPartNumber
30
31    // get part number
32    public String getPartNumber()
33    {
34       return partNumber;
35    } // end method getPartNumber
36
37    // set description
38    public void setPartDescription( String description )
39    {
40       partDescription = description;
41    } // end method setPartDescription
42
43    // get description
44    public String getPartDescription()
45    {
46       return partDescription;
47    } // end method getPartDescription
48
49    // set quantity
50    public void setQuantity( int count )
51    {
52       if ( count > 0 ) // determine whether count is positive
53          quantity = count; // valid count assigned to quantity
54
55       if ( count <= 0 ) // determine whether count is zero or negative
56          quantity = 0; // invalid count; quantity set to 0
```

© Copyright 1992-2007 Pearson Education, Inc., Upper Saddle River, NJ. All rights reserved.

```
57      } // end method setQuantity
58
59      // get quantity
60      public int getQuantity()
61      {
62         return quantity;
63      } // end method getQuantity
64
65      // set price per item
66      public void setPricePerItem( double price )
67      {
68         if ( price > 0.0 ) // determine whether price is positive
69            pricePerItem = price; // valid price assigned to pricePerItem
70
71         if ( price <= 0.0 ) // determine whether price is zero or negative
72            pricePerItem = 0.0; // invalid price; pricePerItem set to 0.0
73      } // end method setPricePerItem
74
75      // get price per item
76      public double getPricePerItem()
77      {
78         return pricePerItem;
79      } // end method getPricePerItem
80
81      // calculates and returns the invoice amount
82      public double getInvoiceAmount()
83      {
84         return getQuantity() * getPricePerItem(); // calculate total cost
85      } // end method getPaymentAmount
86   } // end class Invoice
```

```
1   // Exercises 3.13 Solution: InvoiceTest.java
2   // Application to test class Invoice.
3
4   public class InvoiceTest
5   {
6      public static void main( String args[] )
7      {
8         Invoice invoice1 = new Invoice( "1234", "Hammer", 2, 14.95 );
9
10        // display invoice1
11        System.out.println( "Original invoice information" );
12        System.out.printf( "Part number: %s\n", invoice1.getPartNumber() );
13        System.out.printf( "Description: %s\n",
14           invoice1.getPartDescription() );
15        System.out.printf( "Quantity: %d\n", invoice1.getQuantity() );
16        System.out.printf( "Price: %.2f\n", invoice1.getPricePerItem() );
17        System.out.printf( "Invoice amount: %.2f\n",
18           invoice1.getInvoiceAmount() );
19
20        // change invoice1's data
21        invoice1.setPartNumber( "001234" );
22        invoice1.setPartDescription( "Yellow Hammer" );
```

© Copyright 1992-2007 Pearson Education, Inc., Upper Saddle River, NJ. All rights reserved.

```
23         invoice1.setQuantity( 3 );
24         invoice1.setPricePerItem( 19.49 );
25
26         // display invoice1 with new data
27         System.out.println( "\nUpdated invoice information" );
28         System.out.printf( "Part number: %s\n", invoice1.getPartNumber() );
29         System.out.printf( "Description: %s\n",
30            invoice1.getPartDescription() );
31         System.out.printf( "Quantity: %d\n", invoice1.getQuantity() );
32         System.out.printf( "Price: %.2f\n", invoice1.getPricePerItem() );
33         System.out.printf( "Invoice amount: %.2f\n",
34            invoice1.getInvoiceAmount() );
35
36         Invoice invoice2 = new Invoice( "5678", "Paint Brush", -5, -9.99 );
37
38         // display invoice2
39         System.out.println( "\nOriginal invoice information" );
40         System.out.printf( "Part number: %s\n", invoice2.getPartNumber() );
41         System.out.printf( "Description: %s\n",
42            invoice2.getPartDescription() );
43         System.out.printf( "Quantity: %d\n", invoice2.getQuantity() );
44         System.out.printf( "Price: %.2f\n", invoice2.getPricePerItem() );
45         System.out.printf( "Invoice amount: %.2f\n",
46            invoice2.getInvoiceAmount() );
47
48         // change invoice2's data
49         invoice2.setQuantity( 3 );
50         invoice2.setPricePerItem( 9.49 );
51
52         // display invoice2 with new data
53         System.out.println( "\nUpdated invoice information" );
54         System.out.printf( "Part number: %s\n", invoice2.getPartNumber() );
55         System.out.printf( "Description: %s\n",
56            invoice2.getPartDescription() );
57         System.out.printf( "Quantity: %d\n", invoice2.getQuantity() );
58         System.out.printf( "Price: %.2f\n", invoice2.getPricePerItem() );
59         System.out.printf( "Invoice amount: %.2f\n",
60            invoice2.getInvoiceAmount() );
61
62      } // end main
63
64   } // end class InvoiceTest
```

© Copyright 1992-2007 Pearson Education, Inc., Upper Saddle River, NJ. All rights reserved.

```
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
```

3.14 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 (`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 each object's *yearly* salary. Then give each `Employee` a 10% raise and display each `Employee`'s yearly salary again.

ANS:

```
1  // Exercise 3.14 Solution: Employee.java
2  // Employee class.
3
4  public class Employee
5  {
6     private String firstName;
7     private String lastName;
8     private double monthlySalary;
9
10    // constructor to initialize first name, last name and monthly salary
11    public Employee( String first, String last, double salary )
12    {
13       firstName = first;
14       lastName = last;
15
16       if ( salary >= 0.0 ) // determine whether salary is positive
```

© Copyright 1992-2007 Pearson Education, Inc., Upper Saddle River, NJ. All rights reserved.

```
17            monthlySalary = salary;
18      } // end three-argument Employee constructor
19
20      // set Employee's first name
21      public void setFirstName( String first )
22      {
23         firstName = first;
24      } // end method setFirstName
25
26      // get Employee's first name
27      public String getFirstName()
28      {
29         return firstName;
30      } // end method getFirstName
31
32      // set Employee's last name
33      public void setLastName( String last )
34      {
35         lastName = last;
36      } // end method setLastName
37
38      // get Employee's last name
39      public String getLastName()
40      {
41         return lastName;
42      } // end method getLastName
43
44      // set Employee's monthly salary
45      public void setMonthlySalary( double salary )
46      {
47         if ( salary >= 0.0 ) // determine whether salary is positive
48            monthlySalary = salary;
49      } // end method setMonthlySalary
50
51      // get Employee's monthly salary
52      public double getMonthlySalary()
53      {
54         return monthlySalary;
55      } // end method getMonthlySalary
56
57   } // end class Employee
```

```
1   // Exercise 3.14 Solution: EmployeeTest.java
2   // Application to test class Employee.
3
4   public class EmployeeTest
5   {
6      public static void main( String args[] )
7      {
8         Employee employee1 = new Employee( "Bob", "Jones", 2875.00 );
9         Employee employee2 = new Employee( "Susan", "Baker", 3150.75 );
10
11        // display employees
```

© Copyright 1992-2007 Pearson Education, Inc., Upper Saddle River, NJ. All rights reserved.

```
12         System.out.printf( "Employee 1: %s %s; Yearly Salary: %.2f\n",
13            employee1.getFirstName(), employee1.getLastName(),
14            12 * employee1.getMonthlySalary() );
15         System.out.printf( "Employee 2: %s %s; Yearly Salary: %.2f\n",
16            employee2.getFirstName(), employee2.getLastName(),
17            12 * employee2.getMonthlySalary() );
18
19         // increase employee salaries by 10%
20         System.out.println( "\nIncreasing employee salaries by 10%" );
21         employee1.setMonthlySalary( employee1.getMonthlySalary() * 1.10 );
22         employee2.setMonthlySalary( employee2.getMonthlySalary() * 1.10 );
23
24         // display employees with new yearly salary
25         System.out.printf( "Employee 1: %s %s; Yearly Salary: %.2f\n",
26            employee1.getFirstName(), employee1.getLastName(),
27            12 * employee1.getMonthlySalary() );
28         System.out.printf( "Employee 2: %s %s; Yearly Salary: %.2f\n",
29            employee2.getFirstName(), employee2.getLastName(),
30            12 * employee2.getMonthlySalary() );
31      } // end main
32
33   } // end class EmployeeTest
```

```
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
```

3.15 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.

ANS:

```
1  // Exercise 3.15 Solution: Date.java
2  // Date class with instance variables for the month, day and year.
3
4  public class Date
5  {
6     private int month;
7     private int day;
8     private int year;
9
10    // constructor
11    public Date( int monthValue, int dayValue, int yearValue )
12    {
13       month = monthValue;
```

© Copyright 1992-2007 Pearson Education, Inc., Upper Saddle River, NJ. All rights reserved.

```
14         day = dayValue;
15         year = yearValue;
16      } // end three-argument constructor
17
18      // set the month
19      public void setMonth( int monthValue )
20      {
21         month = monthValue;
22      } // end method setMonth
23
24      // return the month
25      public int getMonth()
26      {
27         return month;
28      } // return month
29
30      // set the day
31      public void setDay( int dayValue )
32      {
33         day = dayValue;
34      } // end method setDay
35
36      // return the day
37      public int getDay()
38      {
39         return day;
40      } // return day
41
42      // set the year
43      public void setYear( int yearValue )
44      {
45         year = yearValue;
46      } // end method setYear
47
48      // return the year
49      public int getYear()
50      {
51         return year;
52      } // return year
53
54      // display the date
55      public void displayDate()
56      {
57         System.out.printf( "%d/%d/%d", getMonth(), getDay(), getYear() );
58      } // end method displayDate
59   } // end class Date
```

```
1   // Exercise 3.15 Solution: DateTest.java
2   // Application to test class Date.
3
4   public class DateTest
5   {
6      public static void main( String args[] )
```

© Copyright 1992-2007 Pearson Education, Inc., Upper Saddle River, NJ. All rights reserved.

```
7      {
8         Date date1 = new Date( 7, 4, 2004 );
9
10        System.out.print( "The initial date is: " );
11        date1.displayDate();
12
13        // change date values
14        date1.setMonth( 11 );
15        date1.setDay( 1 );
16        date1.setYear( 2003 );
17
18        System.out.print( "\nDate with new values is: " );
19        date1.displayDate();
20
21        System.out.println(); // output a newline
22     } // end main
23  } // end class DateTest
```

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

GUI and Graphics Case Study Exercise Solution

3.1 Modify the addition program in Fig. 2.7 to use dialog-based input and output with the methods of class `JOptionPane`. Since method `showInputDialog` returns a `String`, you must convert the `String` the user enters to an `int` for use in calculations. The method

```
Integer.parseInt( String s )
```

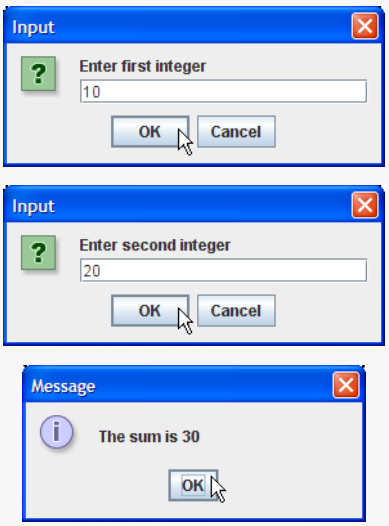
takes a `String` argument representing an integer (e.g., the result of `JOptionPane.showInputDialog`) and returns the value as an `int`. Method `parseInt` is a `static` method of class `Integer` (from package `java.lang`). Note that if the `String` does not contain a valid integer, the program will terminate with an error.

ANS:

```
1  // GCS Exercise 3.1: Addition.java
2  // Modification of addition example from Chapter 2
3  import javax.swing.JOptionPane; // uses JOptionPane instead of Scanner
4
5  public class Addition
6  {
7     // main method begins execution of Java application
8     public static void main( String args[] )
9     {
10       String firstNumber; // first string entered by user
11       String secondNumber; // second string entered by user
12
13       int number1; // first number to add
14       int number2; // second number to add
15       int sum; // sum of number1 and number2
16
17       // read in first number from user as a String
```

© Copyright 1992-2007 Pearson Education, Inc., Upper Saddle River, NJ. All rights reserved.

```
18         firstNumber = JOptionPane.showInputDialog( "Enter first integer" );
19
20         // read in second number from user as a String
21         secondNumber =
22            JOptionPane.showInputDialog( "Enter second integer" );
23
24         // convert numbers from type String to type int
25         number1 = Integer.parseInt( firstNumber );
26         number2 = Integer.parseInt( secondNumber );
27
28         sum = number1 + number2; // add numbers
29
30         // create the message
31         String message = String.format( "The sum is %d", sum );
32
33         // display result
34         JOptionPane.showMessageDialog( null, message );
35      } // end main
36   } // end class Addition
```



© Copyright 1992-2007 Pearson Education, Inc., Upper Saddle River, NJ. All rights reserved.