3.4 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 corresponding parameters of the method so that it can perform its task.

Exercises

3.5 *(Keyword* **new***)* What's the purpose of keyword `new`? Explain what happens when you use it.

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

3.7 *(Instance Variables)* Explain the purpose of an instance variable.

3.8 *(Using Classes without Importing Them)* Most classes need to be imported before they can be used in an app. Why is every app allowed to use classes `System` and `String` without first importing them?

3.9 *(Using a Class without Importing It)* Explain how a program could use class `Scanner` without importing it.

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

3.11 *(Modified* `Account` *Class)* Modify class `Account` (Fig. 3.8) to provide a method called `withdraw` that withdraws money from an `Account`. Ensure that the withdrawal 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 `"Withdrawal amount exceeded account balance."` Modify class `AccountTest` (Fig. 3.9) to test method `withdraw`.

3.12 *(*`Invoice` *Class)* 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 app named `InvoiceTest` that demonstrates class `Invoice`'s capabilities.

3.13 *(*`Employee` *Class)* Create a class called `Employee` that includes three instance variables—a first name (type `String`), a last name (type `String`) and a monthly salary (`double`). Provide 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, do not set its value. Write a test app 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.

3.14 *(*`Date` *Class)* Create a class called `Date` that includes three instance variables—a month (type `int`), a day (type `int`) and a year (type `int`). Provide 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 app named `DateTest` that demonstrates class `Date`'s capabilities.

3.15 *(Removing Duplicated Code in Method* `main`*)* In the `AccountTest` class of Fig. 3.9, method `main` contains six statements (lines 13–14, 15–16, 28–29, 30–31, 40–41 and 42–43) that each display an `Account` object's `name` and `balance`. Study these statements and you'll notice that they differ