**Section 9.1 Introduction**

• Inheritance (p. [361](http://proquest.safaribooksonline.com/9780133813036/ch09_html#page_361)) reduces program-development time.

• The direct superclass (p. [361](http://proquest.safaribooksonline.com/9780133813036/ch09_html#page_361)) of a subclass is the one from which the subclass inherits. An indirect superclass (p. [361](http://proquest.safaribooksonline.com/9780133813036/ch09_html#page_361)) of a subclass is two or more levels up the class hierarchy from that subclass.

• In single inheritance (p. [361](http://proquest.safaribooksonline.com/9780133813036/ch09_html#page_361)), a class is derived from one superclass. In multiple inheritance, a class is derived from more than one direct superclass. Java does not support multiple inheritance.

• A subclass is more specific than its superclass and represents a smaller group of objects (p. [361](http://proquest.safaribooksonline.com/9780133813036/ch09_html#page_361)).

• Every object of a subclass is also an object of that class’s superclass. However, a superclass object is not an object of its class’s subclasses.

• An *is-a* relationship (p. [362](http://proquest.safaribooksonline.com/9780133813036/ch09lev1sec1_html#page_362)) represents inheritance. In an *is-a* relationship, an object of a subclass also can be treated as an object of its superclass.

• A *has-a* relationship (p. [362](http://proquest.safaribooksonline.com/9780133813036/ch09lev1sec1_html#page_362)) represents composition. In a *has-a* relationship, a class object contains references to objects of other classes.

#### Section 9.2 Superclasses and Subclasses

• Single-inheritance relationships form treelike hierarchical structures—a superclass exists in a hierarchical relationship with its subclasses.

#### Section 9.3 protected Members

• A superclass’s public members are accessible wherever the program has a reference to an object of that superclass or one of its subclasses.

• A superclass’s private members can be accessed directly only within the superclass’s declaration.

• A superclass’s protected members (p. [364](http://proquest.safaribooksonline.com/9780133813036/ch09lev2sec2_html#page_364)) have an intermediate level of protection between public andprivate access. They can be accessed by members of the superclass, by members of its subclasses and by members of other classes in the same package.

• A superclass’s private members are hidden in its subclasses and can be accessed only through thepublic or protected methods inherited from the superclass.

• An overridden superclass method can be accessed from a subclass if the superclass method name is preceded by super (p. [364](http://proquest.safaribooksonline.com/9780133813036/ch09lev2sec2_html#page_364)) and a dot (.) separator.

#### Section 9.4 Relationship Between Superclasses and Subclasses

• A subclass cannot access the private members of its superclass, but it can access the non-privatemembers.

• A subclass can invoke a constructor of its superclass by using the keyword super, followed by a set of parentheses containing the superclass constructor arguments. This must appear as the first statement in the subclass constructor’s body.

• A superclass method can be overridden in a subclass to declare an appropriate implementation for the subclass.

• The @Override annotation (p. [369](http://proquest.safaribooksonline.com/9780133813036/ch09lev2sec3_html#page_369)) indicates that a method should override a superclass method. When the compiler encounters a method declared with@Override, it compares the method’s signature with the superclass’s method signatures. If there isn’t an exact match, the compiler issues an error message, such as “method does not override or implement a method from a supertype.”

• Method toString takes no arguments and returns aString. The Object class’s toString method is normally overridden by a subclass.

• When an object is output using the %s format specifier, the object’s toString method is called implicitly to obtain its String representation.

#### Section 9.5 Constructors in Subclasses

• The first task of a subclass constructor is to call its direct superclass’s constructor (p. [378](http://proquest.safaribooksonline.com/9780133813036/ch09lev2sec5_html#page_378)) to ensure that the instance variables inherited from the superclass are initialized.

#### Section 9.6 Class Object

• See the table of class Object’s methods in [Fig. 9.12](http://proquest.safaribooksonline.com/9780133813036/ch09lev1sec6_html#ch09fig12).

### Self-Review Exercises

[**9.1**](http://proquest.safaribooksonline.com/9780133813036/ch09lev1sec11_html#ch09ans1) Fill in the blanks in each of the following statements:

a) \_\_\_\_\_\_ is a form of software reusability in which new classes acquire the members of existing classes and embellish those classes with new capabilities.

b) A superclass’s \_\_\_\_\_\_ members can be accessed in the superclass declaration and in subclass declarations.

c) In a(n) \_\_\_\_\_\_ relationship, an object of a subclass can also be treated as an object of its superclass.

d) In a(n) \_\_\_\_\_\_ relationship, a class object has references to objects of other classes as members.

e) In single inheritance, a class exists in a(n) \_\_\_\_\_\_ relationship with its subclasses.

f) A superclass’s \_\_\_\_\_\_ members are accessible anywhere that the program has a reference to an object of that superclass or to an object of one of its subclasses.

g) When an object of a subclass is instantiated, a superclass \_\_\_\_\_\_ is called implicitly or explicitly.

h) Subclass constructors can call superclass constructors via the \_\_\_\_\_\_ keyword.

[**9.2**](http://proquest.safaribooksonline.com/9780133813036/ch09lev1sec11_html#ch09ans2) State whether each of the following is true or false. If a statement is false, explain why.

a) Superclass constructors are not inherited by subclasses.

b) A has-a relationship is implemented via inheritance.

c) A Car class has an is-a relationship with theSteeringWheel and Brakes classes.

d) When a subclass redefines a superclass method by using the same signature, the subclass is said to overload that superclass method.

### Answers to Self-Review Exercises

[**9.1**](http://proquest.safaribooksonline.com/9780133813036/ch09lev1sec10_html#ch09que1)

a) Inheritance.

b) public and protected.

c) is-a or inheritance.

d) has-a or composition.

e) hierarchical.

f) public.

g) constructor.

h) super.

[**9.2**](http://proquest.safaribooksonline.com/9780133813036/ch09lev1sec10_html#ch09que2)

a) True.

b) False. A has-a relationship is implemented via composition. An is-a relationship is implemented via inheritance.

c) False. This is an example of a has-a relationship. ClassCar has an is-a relationship with class Vehicle.

d) False. This is known as overriding, not overloading—an overloaded method has the same name, but a different signature.

### Exercises

**9.3****(Using Composition Rather Than Inheritance)** Many programs written with inheritance could be written with composition instead, and vice versa. Rewrite classBasePlusCommissionEmployee ([Fig. 9.11](http://proquest.safaribooksonline.com/9780133813036/ch09lev2sec7_html#ch09fig11)) of theCommissionEmployee–BasePlusCommissionEmployeehierarchy to use composition rather than inheritance.

**9.4 (Software Reuse)** Discuss the ways in which inheritance promotes software reuse, saves time during program development and helps prevent errors.

**9.5 (*Student*** **Inheritance Hierarchy)** Draw an inheritance hierarchy for students at a university similar to the hierarchy shown in [Fig. 9.2](http://proquest.safaribooksonline.com/9780133813036/ch09lev2sec1_html#ch09fig02). Use Student as the superclass of the hierarchy, then extend Student with classesUndergraduateStudent and GraduateStudent. Continue to extend the hierarchy as deep (i.e., as many levels) as possible. For example, Freshman, Sophomore, Junior andSenior might extend UndergraduateStudent, andDoctoralStudent and MastersStudent might be subclasses of GraduateStudent. After drawing the hierarchy, discuss the relationships that exist between the classes. [Note: You do not need to write any code for this exercise.]

**9.6 (*Shape*** **Inheritance Hierarchy)** The world of shapes is much richer than the shapes included in the inheritance hierarchy of [Fig. 9.3](http://proquest.safaribooksonline.com/9780133813036/ch09lev2sec2_html#ch09fig03). Write down all the shapes you can think of—both two-dimensional and three-dimensional—and form them into a more complete Shape hierarchy with as many levels as possible. Your hierarchy should have class Shape at the top. Classes TwoDimensionalShape andThreeDimensionalShape should extend Shape. Add additional subclasses, such as Quadrilateral and Sphere, at their correct locations in the hierarchy as necessary.

**9.7 (*protected*** **vs.** ***private*)** Some programmers prefer not to use protected access, because they believe it breaks the encapsulation of the superclass. Discuss the relative merits of using protected access vs. using private access in superclasses.

**9.8 (*Quadrilateral*** **Inheritance Hierarchy)** Write an inheritance hierarchy for classes Quadrilateral, Trapezoid,Parallelogram, Rectangle and Square. Use Quadrilateralas the superclass of the hierarchy. Create and use a Pointclass to represent the points in each shape. Make the hierarchy as deep (i.e., as many levels) as possible. Specify the instance variables and methods for each class. Theprivate instance variables of Quadrilateral should be thex-y coordinate pairs for the four endpoints of theQuadrilateral. Write a program that instantiates objects of your classes and outputs each object’s area (exceptQuadrilateral).

**9.9 (What Does Each Code Snippet Do?)**

a) Assume that the following method call is located in an overridden earnings method in a subclass:

super.earnings()

b) Assume that the following line of code appears before a method declaration:

@Override

c) Assume that the following line of code appears as the first statement in a constructor’s body:

[**Click here to view code image**](http://proquest.safaribooksonline.com/9780133813036/app06_html#p394pro01a)

super(firstArgument, secondArgument);

**9.10 (Write a Line of Code)** Write a line of code that performs each of the following tasks:

a) Specify that class PieceWorker inherits from classEmployee.

b) Call superclass Employee’s toString method from subclass PieceWorker’s toString method.

c) Call superclass Employee’s constructor from subclassPieceWorker’s constructor—assume that the superclass constructor receives three Strings representing the first name, last name and social security number.

**9.11 (Using** ***super*** **in a Constructor’s Body)** Explain why you would use super in the first statement of a subclass constructor’s body.

**9.12 (Using** ***super*** **in an Instance Method’s Body)**Explain why you would use super in the body of a subclass’s instance method.

**9.13 (Calling get Methods in a Class’s Body)** In [Figs. 9.10](http://proquest.safaribooksonline.com/9780133813036/ch09lev2sec7_html#ch09fig10)–[9.11](http://proquest.safaribooksonline.com/9780133813036/ch09lev2sec7_html#ch09fig11) methods earnings and toString each call various get methods within the same class. Explain the benefits of calling these get methods within the classes.

**9.14 (*Employee*** **Hierarchy)** In this chapter, you studied an inheritance hierarchy in which classBasePlusCommissionEmployee inherited from classCommissionEmployee. However, not all types of employees are CommissionEmployees. In this exercise, you’ll create a more general Employee superclass that factors out the attributes and behaviors in class CommissionEmployee that are common to all Employees. The common attributes and behaviors for all Employees are firstName, lastName,socialSecurityNumber, getFirstName, getLastName,getSocialSecurityNumber and a portion of methodtoString. Create a new superclass Employee that contains these instance variables and methods and a constructor. Next, rewrite class CommissionEmployee from [Section 9.4.5](http://proquest.safaribooksonline.com/9780133813036/ch09lev2sec7_html#ch09lev2sec7)as a subclass of Employee. Class CommissionEmployeeshould contain only the instance variables and methods that are not declared in superclass Employee. ClassCommissionEmployee’s constructor should invoke classEmployee’s constructor and CommissionEmployee’s toStringmethod should invoke Employee’s toString method. Once you’ve completed these modifications, run theCommissionEmployeeTest andBasePlusCommissionEmployeeTest apps using these new classes to ensure that the apps still display the same results for a CommissionEmployee object andBasePlusCommissionEmployee object, respectively.

**9.15 (Creating a New Subclass of** ***Employee*)** Other types of Employees might include SalariedEmployees who get paid a fixed weekly salary, PieceWorkers who get paid by the number of pieces they produce or HourlyEmployees who get paid an hourly wage with time-and-a-half—1.5 times the hourly wage—for hours worked over 40 hours.

Create class HourlyEmployee that inherits from classEmployee ([Exercise 9.14](http://proquest.safaribooksonline.com/9780133813036/ch09lev1sec12_html#ch09que14)) and has instance variable hours (adouble) that represents the hours worked, instance variablewage (a double) that represents the wages per hour, a constructor that takes as arguments a first name, a last name, a social security number, an hourly wage and the number of hours worked, set and get methods for manipulating the hours and wage, an earnings method to calculate an HourlyEmployee’s earnings based on the hours worked and a toString method that returns theHourlyEmployee’s String representation. Method setWageshould ensure that wage is nonnegative, and setHoursshould ensure that the value of hours is between 0 and 168 (the total number of hours in a week). Use classHourlyEmployee in a test program that’s similar to the one in [Fig. 9.5](http://proquest.safaribooksonline.com/9780133813036/ch09lev2sec3_html#ch09fig05).