**Section 10.1 Introduction**

• Polymorphism (p. [396](http://proquest.safaribooksonline.com/9780133813036/ch10_html#page_396)) enables us to write programs that process objects that share the same superclass as if they were all objects of the superclass; this can simplify programming.

• With polymorphism, we can design and implement systems that are easily extensible. The only parts of a program that must be altered to accommodate new classes are those that require direct knowledge of the new classes that you add to the hierarchy.

#### Section 10.3 Demonstrating Polymorphic Behavior

• When the compiler encounters a method call made through a variable, it determines if the method can be called by checking the variable’s class type. If that class contains the proper method declaration (or inherits one), the call is compiled. At execution time, the type of the object to which the variable refers determines the actual method to use.

#### Section 10.4 Abstract Classes and Methods

• Abstract classes (p. [401](http://proquest.safaribooksonline.com/9780133813036/ch10lev1sec3_html#page_401)) cannot be used to instantiate objects, because they’re incomplete.

• The primary purpose of an abstract class is to provide an appropriate superclass from which other classes can inherit and thus share a common design.

• Classes that can be used to instantiate objects are called concrete classes (p. [402](http://proquest.safaribooksonline.com/9780133813036/ch10lev2sec7_html#page_402)). They provide implementations of every method they declare (some of the implementations can be inherited).

• Programmers often write client code that uses only abstract superclasses (p. [402](http://proquest.safaribooksonline.com/9780133813036/ch10lev2sec7_html#page_402)) to reduce client code’s dependencies on specific subclass types.

• Abstract classes sometimes constitute several levels of a hierarchy.

• An abstract class normally contains one or more abstract methods (p. [402](http://proquest.safaribooksonline.com/9780133813036/ch10lev2sec7_html#page_402)).

• Abstract methods do not provide implementations.

• A class that contains any abstract methods must be declared as an abstract class (p. [402](http://proquest.safaribooksonline.com/9780133813036/ch10lev2sec7_html#page_402)). Each concrete subclass must provide implementations of each of the superclass’s abstract methods.

• Constructors and static methods cannot be declaredabstract.

• Abstract superclass variables can hold references to objects of any concrete class derived from the superclass. Programs typically use such variables to manipulate subclass objects polymorphically.

• Polymorphism is particularly effective for implementing layered software systems.

#### Section 10.5 Case Study: Payroll System Using Polymorphism

• A hierarchy designer can demand that each concrete subclass provide an appropriate method implementation by including an abstract method in a superclass.

• Most method calls are resolved at execution time, based on the type of the object being manipulated. This process is known as dynamic binding (p. [417](http://proquest.safaribooksonline.com/9780133813036/ch10lev2sec16_html#page_417)) or late binding.

• A superclass variable can be used to invoke only methods declared in the superclass.

• Operator instanceof (p. [417](http://proquest.safaribooksonline.com/9780133813036/ch10lev2sec16_html#page_417)) determines if an object has the is-a relationship with a specific type.

• Every object in Java knows its own class and can access it through Object method getClass (p. [418](http://proquest.safaribooksonline.com/9780133813036/ch10lev2sec16_html#page_418)), which returns an object of type Class (packagejava.lang).

• The is-a relationship applies only between the subclass and its superclasses, not vice versa.

#### Section 10.7 final Methods and Classes

• A method that’s declared final (p. [419](http://proquest.safaribooksonline.com/9780133813036/ch10lev2sec16_html#page_419)) in a superclass cannot be overridden in a subclass.

• Methods declared private are implicitly final, because you can’t override them in a subclass.

• Methods that are declared static are implicitly final.

• A final method’s declaration can never change, so all subclasses use the same implementation, and calls tofinal methods are resolved at compile time—this is known as static binding (p. [420](http://proquest.safaribooksonline.com/9780133813036/ch10lev2sec17_html#page_420)).

• The compiler can optimize programs by removing calls to final methods and inlining their expanded code at each method-call location.

• A class that’s declared final cannot be extended (p.[420](http://proquest.safaribooksonline.com/9780133813036/ch10lev2sec17_html#page_420)).

• All methods in a final class are implicitly final.

#### Section 10.9 Creating and Using Interfaces

• An interface (p. [421](http://proquest.safaribooksonline.com/9780133813036/ch10lev1sec9_html#page_421)) specifies what operations are allowed but not how they’re performed.

• A Java interface describes a set of methods that can be called on an object.

• An interface declaration begins with the keywordinterface (p. [421](http://proquest.safaribooksonline.com/9780133813036/ch10lev1sec9_html#page_421)).

• All interface members must be public, and interfaces may not specify any implementation details, such as concrete method declarations and instance variables.

• All methods declared in an interface are implicitlypublic abstract methods and all fields are implicitlypublic, static and final.

• To use an interface, a concrete class must specify that it implements (p. [421](http://proquest.safaribooksonline.com/9780133813036/ch10lev1sec9_html#page_421)) the interface and must declare each interface method with the signature specified in the interface declaration. A class that does not implement all the interface’s methods must be declaredabstract.

• Implementing an interface is like signing a contract with the compiler that states, “I will declare all the methods specified by the interface or I will declare my class abstract.”

• An interface is typically used when disparate (i.e., unrelated) classes need to share common methods and constants. This allows objects of unrelated classes to be processed polymorphically—objects of classes that implement the same interface can respond to the same method calls.

• You can create an interface that describes the desired functionality, then implement the interface in any classes that require that functionality.

• An interface is often used in place of an abstract class when there’s no default implementation to inherit—that is, no instance variables and no default method implementations.

• Like public abstract classes, interfaces are typicallypublic types, so they’re normally declared in files by themselves with the same name as the interface and the .java filename extension.

• Java does not allow subclasses to inherit from more than one superclass, but it does allow a class to inherit from a superclass and implement more than one interface.

• All objects of a class that implement multiple interfaces have the is-a relationship with each implemented interface type.

• An interface can declare constants. The constants are implicitly public, static and final.

#### Section 10.10 Java SE 8 Interface Enhancements

• In Java SE 8, an interface may declare defaultmethods—that is, public methods with concrete implementations that specify how an operation should be performed.

• When a class implements an interface, the class receives the interface’s default concrete implementations if it does not override them.

• To declare a default method in an interface, you simply place the keyword default before the method’s return type and provide a complete method body.

• When you enhance an existing an interface withdefault methods—any classes that implemented the original interface will not break—it’ll simply receive the default method implementations.

• With default methods, you can declare common method implementations in interfaces (rather than abstractclasses), which gives you more flexibility in designing your classes.

• As of Java SE 8, interfaces may now include public static methods.

• As of Java SE 8, any interface containing only one method is known as a functional interface. There are many such interfaces throughout the Java APIs.

• Functional interfaces are used extensively with Java SE 8’s new lambda capabilities. As you’ll see, lambdas provide a shorthand notation for creating anonymous methods.

### Self-Review Exercises

[**10.1**](http://proquest.safaribooksonline.com/9780133813036/ch10lev1sec15_html#ch10ans1) Fill in the blanks in each of the following statements:

a) If a class contains at least one abstract method, it’s a(n) \_\_\_\_\_\_\_\_ class.

b) Classes from which objects can be instantiated are called \_\_\_\_\_\_\_\_ classes.

c) \_\_\_\_\_\_\_\_ involves using a superclass variable to invoke methods on superclass and sub-class objects, enabling you to “program in the general.”

d) Methods that are not interface methods and that do not provide implementations must be declared using keyword \_\_\_\_\_\_\_.

e) Casting a reference stored in a superclass variable to a subclass type is called \_\_\_\_\_\_\_.

[**10.2**](http://proquest.safaribooksonline.com/9780133813036/ch10lev1sec15_html#ch10ans2) State whether each of the statements that follows istrue or false. If false, explain why.

a) All methods in an abstract class must be declared asabstract methods.

b) Invoking a subclass-only method through a subclass variable is not allowed.

c) If a superclass declares an abstract method, a subclass must implement that method.

d) An object of a class that implements an interface may be thought of as an object of that interface type.

[**10.3**](http://proquest.safaribooksonline.com/9780133813036/ch10lev1sec15_html#ch10ans3) **(Java SE 8 interfaces)** Fill in the blanks in each of the following statements:

a) In Java SE 8, an interface may declare \_\_\_\_\_\_—that is, public methods with concrete implementations that specify how an operation should be performed.

b) As of Java SE 8, interfaces can now include \_\_\_\_\_\_ helper methods.

c) As of Java SE 8, any interface containing only one method is known as a(n) \_\_\_\_\_\_.

### Answers to Self-Review Exercises

[**10.1**](http://proquest.safaribooksonline.com/9780133813036/ch10lev1sec14_html#ch10que1)

a) abstract.

b) concrete.

c) Polymorphism.

d) abstract.

e) downcasting.

[**10.2**](http://proquest.safaribooksonline.com/9780133813036/ch10lev1sec14_html#ch10que2)

a) False. An abstract class can include methods with implementations and abstract methods.

b) False. Trying to invoke a subclass-only method with a superclass variable is not allowed.

c) False. Only a concrete subclass must implement the method.

d) True.

[**10.3**](http://proquest.safaribooksonline.com/9780133813036/ch10lev1sec14_html#ch10que3)

a) default methods.

b) static.

c) functional interface.

### Exercises

**10.4** How does polymorphism enable you to program “in the general” rather than “in the specific”? Discuss the key advantages of programming “in the general.”

**10.5** What are abstract methods? Describe the circumstances in which an abstract method would be appropriate.

**10.6** How does polymorphism promote extensibility?

**10.7** Discuss three proper ways in which you can assign superclass and subclass references to variables of superclass and subclass types.

**10.8** Compare and contrast abstract classes and interfaces. Why would you use an abstract class? Why would you use an interface?

**10.9 (Java SE 8 Interfaces)** Explain how default methods enable you to add new methods to an existing interface without breaking the classes that implemented the original interface.

**10.10 (Java SE 8 Interfaces)** What is a functional interface?

**10.11 (Java SE 8 Interfaces)** Why is it useful to be able to add static methods to interfaces?

**10.12 (Payroll System Modification)** Modify the payroll system of [Figs. 10.4](http://proquest.safaribooksonline.com/9780133813036/ch10lev2sec11_html#ch10fig04)–[10.9](http://proquest.safaribooksonline.com/9780133813036/ch10lev2sec16_html#ch10fig09) to include private instance variable birthDate in class Employee. Use class Date of [Fig. 8.7](http://proquest.safaribooksonline.com/9780133813036/ch08lev2sec17_html#ch08fig07) to represent an employee’s birthday. Add get methods to class Date. Assume that payroll is processed once per month. Create an array of Employee variables to store references to the various employee objects. In a loop, calculate the payroll for each Employee (polymorphically), and add a $100.00 bonus to the person’s payroll amount if the current month is the one in which the Employee’s birthday occurs.

**10.13 (Project:** ***Shape*** **Hierarchy)** Implement the Shapehierarchy shown in [Fig. 9.3](http://proquest.safaribooksonline.com/9780133813036/ch09lev2sec2_html#ch09fig03). Each TwoDimensionalShapeshould contain method getArea to calculate the area of the two-dimensional shape. Each ThreeDimensionalShapeshould have methods getArea and getVolume to calculate the surface area and volume, respectively, of the three-dimensional shape. Create a program that uses an array ofShape references to objects of each concrete class in the hierarchy. The program should print a text description of the object to which each array element refers. Also, in the loop that processes all the shapes in the array, determine whether each shape is a TwoDimensionalShape or aThreeDimensionalShape. If it’s a TwoDimensionalShape, display its area. If it’s a ThreeDimensionalShape, display its area and volume.

**10.14 (Payroll System Modification)** Modify the payroll system of [Figs. 10.4](http://proquest.safaribooksonline.com/9780133813036/ch10lev2sec11_html#ch10fig04)–[10.9](http://proquest.safaribooksonline.com/9780133813036/ch10lev2sec16_html#ch10fig09) to include an additional Employeesubclass PieceWorker that represents an employee whose pay is based on the number of pieces of merchandise produced. Class PieceWorker should contain privateinstance variables wage (to store the employee’s wage per piece) and pieces (to store the number of pieces produced). Provide a concrete implementation of method earnings in class PieceWorker that calculates the employee’s earnings by multiplying the number of pieces produced by the wage per piece. Create an array of Employee variables to store references to objects of each concrete class in the newEmployee hierarchy. For each Employee, display its Stringrepresentation and earnings.

**10.15 (Accounts Payable System Modification)** In this exercise, we modify the accounts payable application of [Figs. 10.11](http://proquest.safaribooksonline.com/9780133813036/ch10lev2sec26_html#ch10fig11)–[10.15](http://proquest.safaribooksonline.com/9780133813036/ch10lev2sec30_html#ch10fig15) to include the complete functionality of the payroll application of [Figs. 10.4](http://proquest.safaribooksonline.com/9780133813036/ch10lev2sec11_html#ch10fig04)–[10.9](http://proquest.safaribooksonline.com/9780133813036/ch10lev2sec16_html#ch10fig09). The application should still process two Invoice objects, but now should process one object of each of the four Employee subclasses. If the object currently being processed is aBasePlusCommissionEmployee, the application should increase the BasePlusCommissionEmployee’s base salary by 10%. Finally, the application should output the payment amount for each object. Complete the following steps to create the new application:

a) Modify classes HourlyEmployee ([Fig. 10.6](http://proquest.safaribooksonline.com/9780133813036/ch10lev2sec13_html#ch10fig06)) andCommissionEmployee ([Fig. 10.7](http://proquest.safaribooksonline.com/9780133813036/ch10lev2sec14_html#ch10fig07)) to place them in thePayable hierarchy as subclasses of the version ofEmployee ([Fig. 10.13](http://proquest.safaribooksonline.com/9780133813036/ch10lev2sec28_html#ch10fig13)) that implements Payable. [Hint: Change the name of method earnings togetPaymentAmount in each subclass so that the class satisfies its inherited contract with interface Payable.]

b) Modify class BasePlusCommissionEmployee ([Fig. 10.8](http://proquest.safaribooksonline.com/9780133813036/ch10lev2sec15_html#ch10fig08)) such that it extends the version of classCommissionEmployee created in part (a).

c) Modify PayableInterfaceTest ([Fig. 10.15](http://proquest.safaribooksonline.com/9780133813036/ch10lev2sec30_html#ch10fig15)) to polymorphically process two Invoices, oneSalariedEmployee, one HourlyEmployee, oneCommissionEmployee and one Base PlusCommissionEmployee. First output a Stringrepresentation of each Payable object. Next, if an object is a BasePlusCommissionEmployee, increase its base salary by 10%. Finally, output the payment amount for each Payable object.

**10.16****(Accounts Payable System Modification)** It’s possible to include the functionality of the payroll application ([Figs. 10.4](http://proquest.safaribooksonline.com/9780133813036/ch10lev2sec11_html#ch10fig04)–[10.9](http://proquest.safaribooksonline.com/9780133813036/ch10lev2sec16_html#ch10fig09)) in the accounts payable application without modifying Employee subclasses SalariedEmployee,HourlyEmployee, CommissionEmployee orBasePlusCommissionEmployee. To do so, you can modify class Employee ([Fig. 10.4](http://proquest.safaribooksonline.com/9780133813036/ch10lev2sec11_html#ch10fig04)) to implement interface Payableand declare method getPaymentAmount to invoke methodearnings. Method getPaymentAmount would then be inherited by the subclasses in the Employee hierarchy. WhengetPaymentAmount is called for a particular subclass object, it polymorphically invokes the appropriate earnings method for that subclass. Reimplement [Exercise 10.15](http://proquest.safaribooksonline.com/9780133813036/ch10lev1sec16_html#ch10que15) using the original Employee hierarchy from the payroll application of[Figs. 10.4](http://proquest.safaribooksonline.com/9780133813036/ch10lev2sec11_html#ch10fig04)–[10.9](http://proquest.safaribooksonline.com/9780133813036/ch10lev2sec16_html#ch10fig09). Modify class Employee as described in this exercise, and do not modify any of class Employee’s subclasses.

### Making a Difference

**10.17 (*CarbonFootprint*** **Interface: Polymorphism)**Using interfaces, as you learned in this chapter, you can specify similar behaviors for possibly disparate classes. Governments and companies worldwide are becoming increasingly concerned with carbon footprints (annual releases of carbon dioxide into the atmosphere) from buildings burning various types of fuels for heat, vehicles burning fuels for power, and the like. Many scientists blame these greenhouse gases for the phenomenon called global warming. Create three small classes unrelated by inheritance—classes Building, Car and Bicycle. Give each class some unique appropriate attributes and behaviors that it does not have in common with other classes. Write an interfaceCarbonFootprint with a getCarbonFootprint method. Have each of your classes implement that interface, so that itsgetCarbonFootprint method calculates an appropriate carbon footprint for that class (check out a few websites that explain how to calculate carbon footprints). Write an application that creates objects of each of the three classes, places references to those objects inArrayList<CarbonFootprint>, then iterates through theArrayList, polymorphically invoking each object’sgetCarbonFootprint method. For each object, print some identifying information and the object’s carbon footprint.