

*Say not you know another entirely,
till you have divided an inheritance with him.*

—Johann Kasper Lavater

*This method is to define as the number of a class the
class of all classes similar to the given class.*

—Bertrand Russell

*Good as it is to inherit a library,
it is better to collect one.*

—Augustine Birrell

Save base authority from others' books.

—William Shakespeare

9

Object-Oriented Programming: Inheritance

Objectives

In this Chapter you'll learn:

- How inheritance promotes software reusability.
- The notions of superclasses and subclasses.
- To use keyword **extends** to create a class that inherits attributes and behaviors from another class.
- To use access modifier **protected** to give subclass methods access to superclass members.
- To access superclass members with **super**.
- How constructors are used in inheritance hierarchies.
- The methods of class **Object**, the direct or indirect superclass of all classes in Java.

Self-Review Exercises

- 9.1** Fill in the blanks in each of the following statements:
- _____ is a form of software reusability in which new classes acquire the members of existing classes and embellish those classes with new capabilities.
ANS: Inheritance.
 - A superclass's _____ members can be accessed in the superclass declaration *and in* subclass declarations.
ANS: public and protected.
 - In a(n) _____ relationship, an object of a subclass can also be treated as an object of its superclass.
ANS: *is-a* or inheritance.
 - In a(n) _____ relationship, a class object has references to objects of other classes as members.
ANS: *has-a* or composition.
 - In single inheritance, a class exists in a(n) _____ relationship with its subclasses.
ANS: hierarchical.
 - 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.
ANS: public.
 - When an object of a subclass is instantiated, a superclass _____ is called implicitly or explicitly.
ANS: constructor.
 - Subclass constructors can call superclass constructors via the _____ keyword.
ANS: super.
- 9.2** State whether each of the following is *true* or *false*. If a statement is *false*, explain why.
- Superclass constructors are not inherited by subclasses.
ANS: True.
 - A *has-a* relationship is implemented via inheritance.
ANS: False. A *has-a* relationship is implemented via composition. An *is-a* relationship is implemented via inheritance.
 - A Car class has an *is-a* relationship with the SteeringWheel and Brakes classes.
ANS: False. This is an example of a *has-a* relationship. Class Car has an *is-a* relationship with class Vehicle.
 - When a subclass redefines a superclass method by using the same signature, the subclass is said to overload that superclass method.
ANS: False. This is known as overriding, not overloading—an overloaded method has the same name, but a different signature.

Exercises

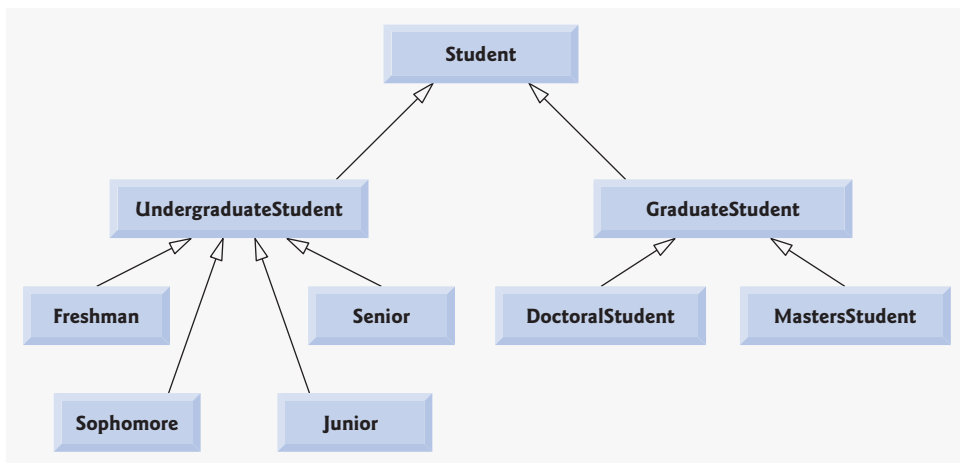
NOTE: Solutions to the programming exercises are located in the `ch09solutions` folder. Each exercise has its own folder named `ex09_##` where `##` is a two-digit number representing the exercise number. For example, exercise 9.3's solution is located in the folder `ex09_03`.

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

ANS: Inheritance allows developers to create subclasses that reuse code declared already in a superclass. Avoiding the duplication of common functionality between several classes by building a class inheritance hierarchy can save developers a considerable amount of time. Similarly, placing common functionality in a single superclass, rather than duplicating the code in multiple unrelated classes, helps prevent the same errors from appearing in multiple source-code files. If errors occur in the common functionality of the superclass, the software developer needs to modify only the superclass's.

9.5 Draw an inheritance hierarchy for students at a university similar to the hierarchy shown in Fig. 9.2. Use Student as the superclass of the hierarchy, then extend Student with classes UndergraduateStudent and GraduateStudent. Continue to extend the hierarchy as deep (i.e., as many levels) as possible. For example, Freshman, Sophomore, Junior and Senior might extend UndergraduateStudent, and DoctoralStudent 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.]

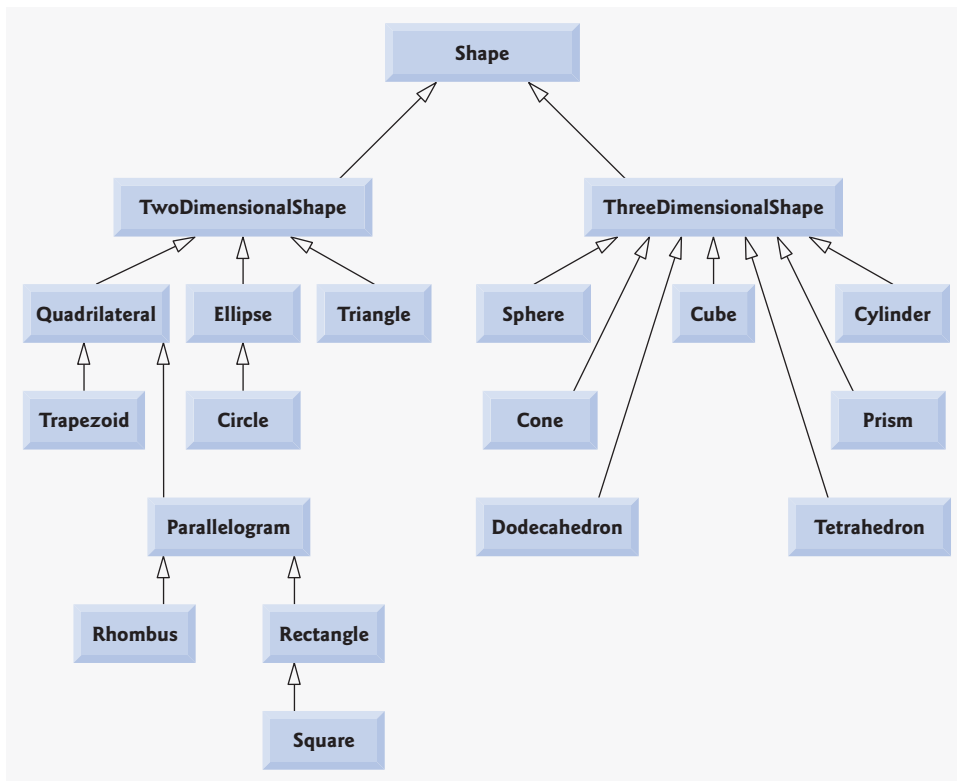
ANS:



This hierarchy contains many *is-a* (inheritance) relationships. An UndergraduateStudent *is a* Student. A GraduateStudent *is a* Student, too. Each of the classes Freshman, Sophomore, Junior and Senior *is an* UndergraduateStudent and *is a* Student. Each of the classes DoctoralStudent and MastersStudent *is a* GraduateStudent and *is a* Student. Note that there could be *many* more classes. For example, GraduateStudent could have subclasses like LawStudent, BusinessStudent, MedicalStudent, etc.

9.6 The world of shapes is much richer than the shapes included in the inheritance hierarchy of Fig. 9.3. 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 and ThreeDimensionalShape should extend Shape. Add additional subclasses, such as Quadrilateral and Sphere, at their correct locations in the hierarchy as necessary.

ANS: [Note: Solutions may vary.]



9.7 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.

ANS: private instance variables are hidden in the subclass and are accessible only through the public or protected methods of the superclass. Using protected access enables the subclass to manipulate the protected members without using the access methods of the superclass. This makes the code more brittle, because changes to the superclass might require changes to the subclasses. If the superclass instance variables are private, the methods of the superclass must be used to access the data. This encapsulation makes the code easier to maintain, modify and debug.