cpphtp10.book Page 1 Tuesday, September 6, 2016 7:06 PM

11 Object-Oriented Programming: Inheritance

Objectives

In this chapter you'll:

- Learn what inheritance is.
- Understand the notions of base classes and derived classes and the relationships between them.
- Use the `protected` member access specifier.
- Use constructors and destructors in inheritance hierarchies.
- Understand the order in which constructors and destructors are called in inheritance hierarchies.
- Understand the differences between `public`, `protected` and `private` inheritance.
- Use inheritance to customize existing software.

cpphtp10.book Page 2 Tuesday, September 6, 2016 7:06 PM

Self-Review Exercises

11.1 Fill in the blanks in each of the following statements:

a) _____ enables new classes to absorb the data and behaviors of existing classes and embellish these classes with new capabilities.

ANS: Inheritance.

b) A base class's _____ and _____ members can be accessed in the base-class definition, in derived-class definitions and in `friend`s of the base class and derived classes.

ANS: `public`, `protected`.

c) In a(n) _____ relationship, an object of a derived class also can be treated as an object of its base class.

ANS: *is-a* or inheritance (for `public` inheritance).

d) In a(n) _____ relationship, a class object has one or more objects of other classes as members.

ANS: *has-a* or composition or aggregation.

e) In single inheritance, a class exists in a(n) _____ relationship with its derived classes.

ANS: hierarchical.

f) A base class's _____ members are accessible within that base class and anywhere that the program has a handle to an object of that class or one of its derived classes.

ANS: `public`.

g) A base class's `protected` access members have a level of protection between those of `public` and _____ access.

ANS: `private`.

h) C++ provides for _____, which allows a derived class to inherit from many base classes, even if the base classes are unrelated.

ANS: multiple inheritance.

i) When an object of a derived class is instantiated, the base class's _____ is called implicitly or explicitly to do any necessary initialization of the base-class data members in the derived-class object.

ANS: constructor.

j) When deriving a class with `public` inheritance, `public` members of the base class become _____ members of the derived class, and `protected` members of the base class become _____ members of the derived class.

ANS: `public`, `protected`.

k) When deriving from a class with `protected` inheritance, `public` members of the base class become _____ members of the derived class, and `protected` members of the base class become _____ members of the derived class.

ANS: `protected`, `protected`.

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

a) Base-class constructors are not automatically inherited by derived classes.

ANS: True.

b) 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.

c) 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`.

d) When a derived-class object is destroyed, the destructors are called in the reverse order of the constructors.

ANS: True.

cpphtp10.book Page 3 Tuesday, September 6, 2016 7:06 PM

Exercises

11.3 ***(Composition as an Alternative to Inheritance)*** Many programs written with inheritance can be written with composition instead, and vice versa. Rewrite class `BasePlusCommissionEmployee` of the `CommissionEmployee`–`BasePlusCommissionEmployee` hierarchy to use composition rather than inheritance. After you do this, assess the relative merits of the two approaches for designing classes `CommissionEmployee` and `BasePlusCommissionEmployee`, as well as for object-oriented programs in general. Which approach is more natural? Why?

ANS: For a relatively short program like this one, either approach is acceptable. Inheritance becomes preferable when it makes the program easier to modify and promotes the reuse of code. The inheritance approach is more natural because a base-salaried commission employee *is a* commission employee. Composition is defined by the "has-a" relationship, and clearly it would be strange to say that "a base-salaried commission employee *has a* commission employee."

11.4 ***(Inheritance Advantage)*** Discuss the ways in which inheritance saves time during program development and helps prevent errors.

ANS: Inheritance allows developers to create derived classes that reuse code declared already in a base class. Avoiding the duplication of common functionality between several classes by building an inheritance hierarchy to contain the classes can save developers a considerable amount of time. Placing common functionality in a single base class, rather than duplicating the code in multiple unrelated classes, helps prevent the same errors from appearing in multiple source-code files and makes debugging easier. If several classes each contain duplicate code containing an error, the software developer has to spend time correcting each source-code file with the error. However, if these classes take advantage of inheritance, and the error occurs in the common functionality of the base class, the software developer needs to modify only the base class's code.

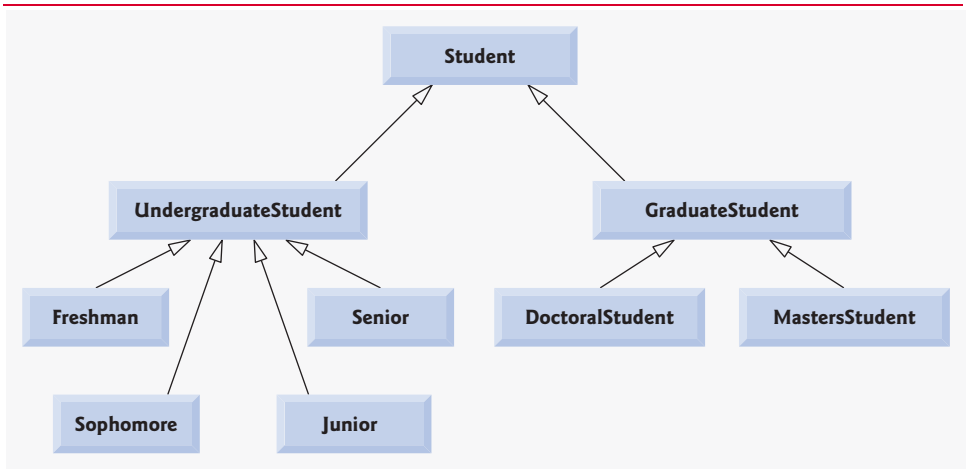
11.5 ***(Protected vs. Private Base Classes)*** Some programmers prefer not to use `protected` access because they believe it breaks the encapsulation of the base class. Discuss the relative merits of using `protected` access vs. using `private` access in base classes.

ANS: `private` data members are hidden in the base class and are accessible only through the `public` or `protected` member functions of the base class. Using `protected` access enables the derived class to manipulate the `protected` members without using the access functions of the base class. If the base-class members are `private`, the member functions of the base class must be used to access the data. This may result in a decrease in performance due to the extra function calls, yet accessing and modifying `private` data in this indirect manner helps ensure that the data in the base class remains consistent.

11.6 ***(Student Inheritance Hierarchy)*** Draw an inheritance hierarchy for students at a university similar to the hierarchy shown in Fig. 11.2. Use `Student` as the base class of the hierarchy, then include classes `UndergraduateStudent` and `GraduateStudent` that derive from `Student`. Continue to extend the hierarchy as deep (i.e., as many levels) as possible. For example, `Freshman`, `Sophomore`, `Junior` and `Senior` might derive from `UndergraduateStudent`, and `DoctoralStudent` and `MastersStudent` might derive from `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.]

cpphtp10.book Page 4 Tuesday, September 6, 2016 7:06 PM

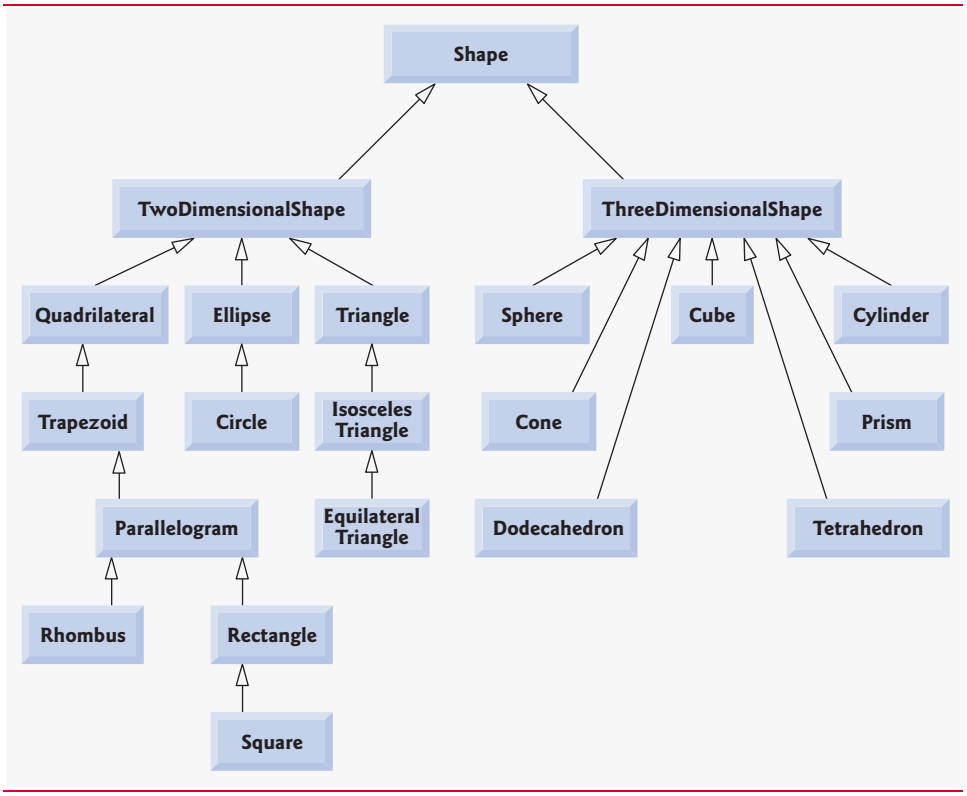
ANS: This hierarchy contains many "is-a" (inheritance) relationships. An `UndergraduateStudent` *is a* `Student`. A `GraduateStudent` *is a* `Student`, too. Each of the classes `Freshman`, `Sophomore`, `Junior` and `Senior` *is an* `UndergraduateStudent` and *is a* `Student`. Each of the classes `DoctoralStudent` and `MastersStudent` *is a* `GraduateStudent` and *is a* `Student`.



11.7 ***(Richer Shape Hierarchy)*** The world of shapes is much richer than the shapes included in the inheritance hierarchy of Fig. 11.3. 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 the base-class `Shape` from which class `TwoDimensionalShape` and class `ThreeDimensionalShape` are derived. [*Note:* You do not need to write any code for this exercise.] We'll use this hierarchy in the exercises of Chapter 12 to process a set of distinct shapes as objects of base-class `Shape`. (This technique, called polymorphism, is the subject of Chapter 12.)

cpphtp10.book Page 5 Tuesday, September 6, 2016 7:06 PM

ANS: [*Note:* Solutions may vary.]



11.8 *(Quadrilateral Inheritance Hierarchy)* Draw an inheritance hierarchy for classes `Quadrilateral`, `Trapezoid`, `Parallelogram`, `Rectangle` and `Square`. Use `Quadrilateral` as the base class of the hierarchy. Make the hierarchy as deep as possible.

cpphtp10.book Page 6 Tuesday, September 6, 2016 7:06 PM

ANS: