Chapter 11
Object-Oriented Programming: Inheritance

C++ How to Program, 9/e

OBJECTIVES

In this chapter you'll learn:

- What inheritance is and how it promotes software reuse.
- The notions of base classes and derived classes and the relationships between them.
- The `protected` member access specifier.
- The use of constructors and destructors in inheritance hierarchies.
- The order in which constructors and destructors are called in inheritance hierarchies.
- The differences between `public`, `protected` and `private` inheritance.
- To use inheritance to customize existing software.

11.1 Introduction

- Inheritance is a form of software reuse in which you create a class that absorbs an existing class's data and behaviors and enhances them with new capabilities.
- You can designate that the new class should inherit the members of an existing class.
- This existing class is called the base class, and the new class is referred to as the derived class.
- A derived class represents a *more specialized* group of objects.
- C++ offers `public`, `protected` and `private` inheritance.
- *With `public` inheritance, every object of a derived class is also an object of that derived class's base class.*
- However, base-class objects are not objects of their derived classes.

11.1 Introduction (cont.)

- With object-oriented programming, you focus on the commonalities among objects in the system rather than on the special cases.
- We distinguish between the *is-a* relationship and the *has-a* relationship.
- The *is-a* relationship represents inheritance.
- In an *is-a* relationship, an object of a derived class also can be treated as an object of its base class.
- By contrast, the *has-a* relationship represents

11.2 Base Classes and Derived Classes

- Figure 11.1 lists several simple examples of base classes and derived classes.
  - Base classes tend to be *more general* and derived classes tend to be *more specific*.
- Because every derived-class object *is an* object of its base class, and one base class can have *many* derived classes, the set of objects represented by a base class typically is *larger* than the set of objects represented by any of its derived classes.
- Inheritance relationships form class hierarchies.

11.2 Base Classes and Derived Classes (cont.)

- A base class exists in a hierarchical relationship with its derived classes.
- Although classes can exist independently, once they're employed in inheritance relationships, they become affiliated with other classes.
- A class becomes either a base class—supplying members to other classes, a derived class—inheriting its members from other classes, or *both*.

Base class	Derived classes
Student	GraduateStudent, UndergraduateStudent
Shape	Circle, Triangle, Rectangle, Sphere, Cube
Loan	CarLoan, HomeImprovementLoan, MortgageLoan
Employee	Faculty, Staff
Account	CheckingAccount, SavingsAccount

Fig. 11.1 | Inheritance examples.

11.2 Base Classes and Derived Classes (cont.)

`CommunityMember` Class Hierarchy

- Let's develop a simple inheritance hierarchy with five levels (represented by the UML class diagram in Fig. 11.2).
- A university community has thousands of `CommunityMember`s.
- `Employee`s are either `Faculty` or `Staff`.
- `Faculty` are either `Administrator`s or `Teacher`s.
- Some `Administrator`s, however, are also `Teacher`s.
- We've used *multiple inheritance* to form class `AdministratorTeacher`.

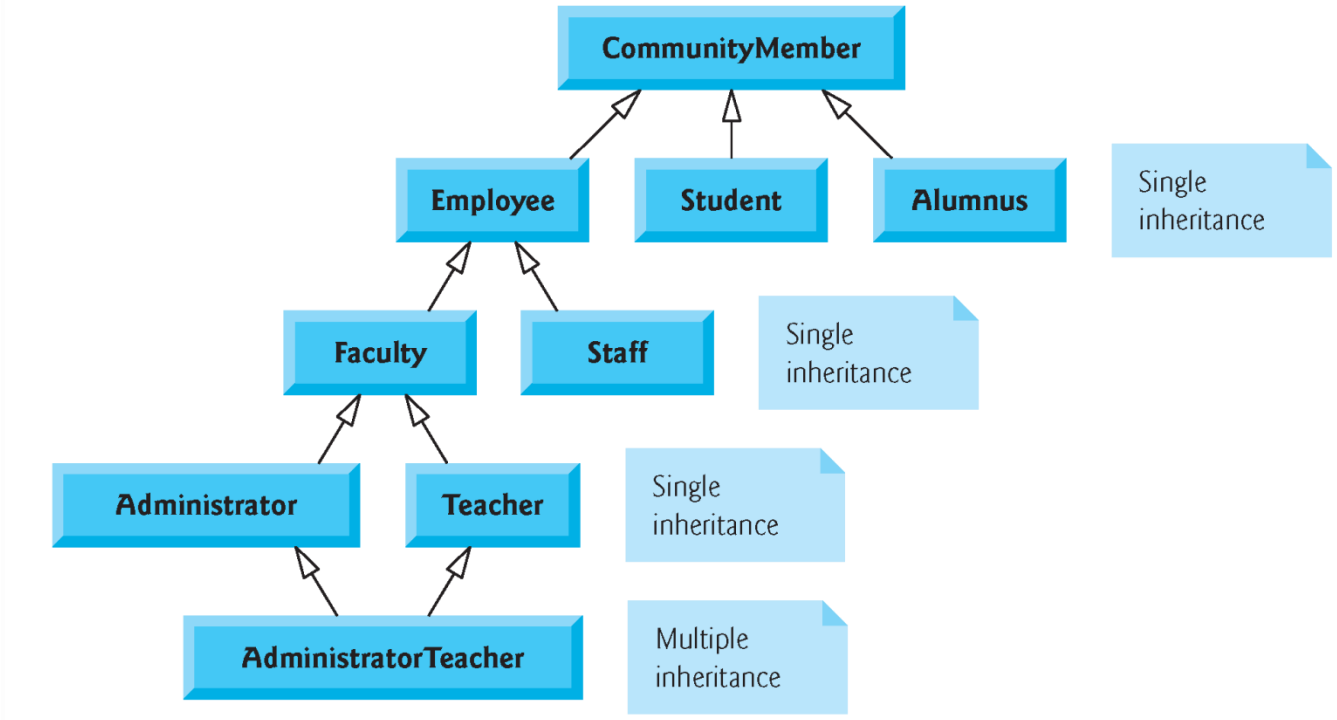

Fig. 11.2 | Inheritance hierarchy for university `CommunityMembers`.

11.2 Base Classes and Derived Classes (cont.)

- With single inheritance, a class is derived from *one* base class.
- With multiple inheritance, a derived class inherits simultaneously from *two or more* (possibly unrelated) base classes.
  - We discuss multiple inheritance in Chapter 23, Other Topics.

11.2 Base Classes and Derived Classes (cont.)

- Each arrow in the hierarchy (Fig. 11.2) represents an *is-a relationship.*
  - As we follow the arrows in this class hierarchy, we can state "an `Employee` *is a* `CommunityMember`" and "a `Teacher` *is a* `Faculty` member."
  - `CommunityMember` is the direct base class of `Employee`, `Student` and `Alumnus`.
  - `CommunityMember` is an indirect base class of all the other classes in the diagram.
- Starting from the bottom of the diagram, you can follow the arrows and apply the *is-a* relationship to the topmost base class.
  - An `AdministratorTeacher` *is an* `Administrator`, *is a* `Faculty` member, *is an* `Employee` and *is a* `CommunityMember`.

11.2 Base Classes and Derived Classes (cont.)

Shape Class Hierarchy

- Consider the `Shape` inheritance hierarchy in Fig. 11.3.
- Begins with base class `Shape`.
- Classes `TwoDimensionalShape` and `ThreeDimensionalShape` derive from base class `Shape`—`Shape`s are either `TwoDimensionalShape`s or `Three-DimensionalShape`s.
- The third level of this hierarchy contains some more specific types of `TwoDimensionalShape`s and `ThreeDimensionalShape`s.
- As in Fig. 11.2, we can follow the arrows from the bottom of the diagram to the topmost base class in this class hierarchy to identify several *is-a* relationships.

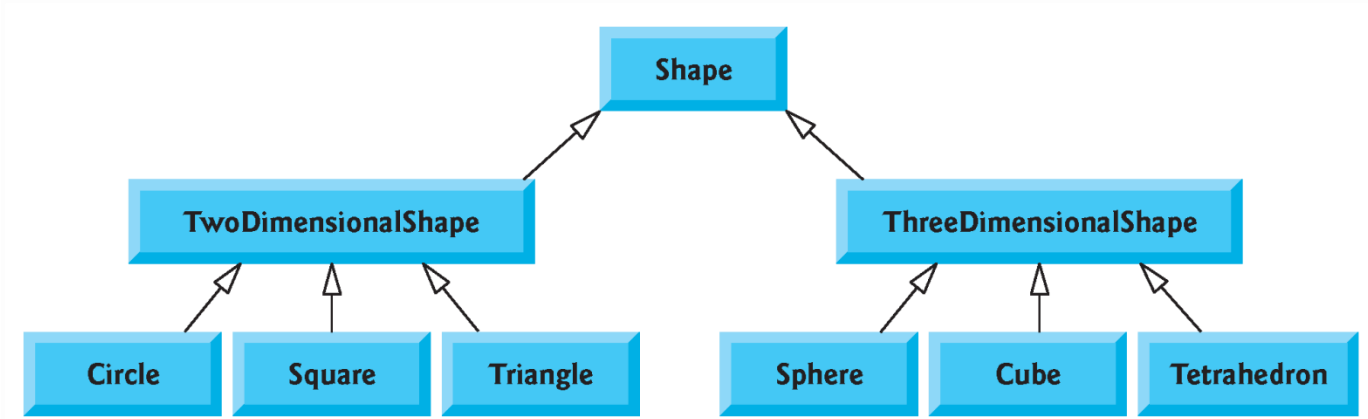

Fig. 11.3 | Inheritance hierarchy for Shapes.

11.3 Relationship between Base and Derived Classes

- In this section, we use an inheritance hierarchy containing types of employees in a company's payroll application to discuss the relationship between a base class and a derived class.
- Commission employees (who will be represented as objects of a base class) are paid a percentage of their sales, while base-salaried commission employees (who will be represented as objects of a derived class) receive a base salary plus a percentage of their sales.