9.1 Introduction

This chapter continues our discussion of object-oriented programming (OOP) by introducing **inheritance**, in which a new class is created by acquiring an existing class's members and possibly embellishing them with new or modified capabilities. With inheritance, you can save time during program development by basing new classes on existing proven and debugged high-quality software. This also increases the likelihood that a system will be implemented and maintained effectively.

When creating a class, rather than declaring completely new members, you can designate that the new class should *inherit* the members of an existing class. The existing class is called the **superclass**, and the new class is the **subclass.** (The C++ programming language refers to the superclass as the **base class** and the subclass as the **derived class**.) A subclass can become a superclass for future subclasses.

A subclass can add its own fields and methods. Therefore, a subclass is *more specific* than its superclass and represents a more specialized group of objects. The subclass exhibits the behaviors of its superclass and can modify those behaviors so that they operate appropriately for the subclass. This is why inheritance is sometimes referred to as **specialization**.

The **direct superclass** is the superclass from which the

subclass explicitly inherits. An **indirect superclass** is any class above the direct superclass in the **class hierarchy**, which defines the inheritance relationships among classes—as you'll see in Section 9.2, diagrams help you understand these relationships. In Java, the class hierarchy begins with class `Object` (in package `java.lang`), which *every* class in Java directly or indirectly **extends** (or "inherits from"). Section 9.6 lists the methods of class `Object` that are inherited by all other Java classes. Java supports only **single inheritance**, in which each class is derived from exactly *one* direct superclass. Unlike C++, Java does *not* support multiple inheritance (which occurs when a class is derived from more than one direct superclass). Chapter 10, Object-Oriented Programming: Polymorphism and Interfaces, explains how to use Java *interfaces* to realize many of the benefits of multiple inheritance while avoiding the associated problems.

We distinguish between the ***is-a relationship*** and the ***has-a relationship***. *Is-a* represents inheritance. In an *is-a* relationship, *an object of a subclass can also be treated as an object of its superclass*—e.g., a car *is a* vehicle. By contrast, *has-a* represents composition (see Chapter 8). In a *has-a* relationship, *an object contains as members references to other objects*—e.g., a car *has a* steering wheel (and a car object has a reference to a steering-wheel object).

New classes can inherit from classes in **class libraries**. Organizations develop their own class libraries and can take advantage of others available worldwide. Some day, most new software likely will be constructed from **standardized reusable components**, just as automobiles and most computer

hardware are constructed today. This will facilitate the rapid development of more powerful, abundant and economical software.