9.3 `protected` Members

Chapter 8 discussed access modifiers `public` and `private`. A class's `public` members are accessible wherever the program has a *reference* to an *object* of that class or one of its *subclasses*. A class's `private` members are accessible only within the class itself. In this section, we introduce the access modifier `protected`. Using `protected` access offers an intermediate level of access between `public` and `private`. A superclass's `protected` members can be accessed by members of that superclass, by members of its subclasses and by members of other classes in the *same package*—`protected` members also have *package access*.

All `public` and `protected` superclass members retain their original access modifier when they become members of the subclass—`public` members of the superclass become `public` members of the subclass, and `protected` members of the superclass become `protected` members of the subclass. A superclass's `private` members are *not* accessible outside the class itself. Rather, they're *hidden* from its subclasses and can be accessed only through the `public` or `protected` methods inherited from the superclass.

Subclass methods can refer to `public` and `protected` members inherited from the superclass simply by using the member names. When a subclass method *overrides* an

inherited superclass method, the *superclass* version of the method can be accessed from the *subclass* by preceding the superclass method name with keyword `super` and a dot (`.`) separator. We discuss accessing overridden members of the superclass in Section 9.4.

Software Engineering Observation 9.1

Methods of a subclass cannot directly access `private` *members of their superclass. A subclass can change the state of* `private` *superclass instance variables only through non-*`private` *methods provided in the superclass and inherited by the subclass.*

Software Engineering Observation 9.2

Declaring `private` *instance variables helps you test, debug and correctly modify systems. If a subclass could access its superclass's* `private` *instance variables, classes that inherit from that subclass could access the instance variables as well. This would propagate access to what should be* `private` *instance variables, and the benefits of information hiding would be lost.*