9.7 Designing with Composition vs. Inheritance

There's much discussion in the software engineering community about the relative merits of composition and inheritance. Each has its own place, but inheritance is often overused and composition is more appropriate in many cases. A mix of composition and inheritance often is a reasonable design approach, as you'll see in Exercise 9.16.1

1. The concepts we present in this section are widely discussed in the software engineering community and derived from many sources, most notably the books: Gamma, Erich et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley, 1995, and Bloch, Joshua. *Effective Java*. Upper Saddle River, NJ: Addison-Wesley, 2008.

Software Engineering Observation 9.10

As a college student, you learn tools for creating solutions. In industry, problems are larger and more complex than you see in college courses. Often the demands of the problem you're solving will be unique and may require you to rethink the proper way to use the tools at your disposal. As a college student, you tend to work on problems yourself. In industry, problem solving often requires interaction among many

colleagues. Rarely will you be able to get everyone on a project to agree on the "right" approach to a solution. Also, rarely will any particular approach be "perfect." You'll often compare the relative merits of different approaches, as we do in this section.

Inheritance-Based Designs

Inheritance creates *tight coupling* among the classes in a hierarchy—each subclass typically depends on its direct or indirect superclasses' implementations. Changes in superclass implementation can affect the behavior of subclasses, often in subtle ways. Tightly coupled designs are more difficult to modify than those in loosely coupled, composition-based designs (discussed momentarily). Change is the rule rather than the exception—this encourages composition.

In general, you should use inheritance only for true *is-a* relationships in which you can assign a subclass object to a superclass reference. When you invoke a method via a superclass reference to a subclass object, the subclass's corresponding method executes. This is called polymorphic behavior, which we explore in Chapter 10.

Software Engineering Observation 9.11

Some of the difficulties with inheritance occur on large projects where different classes in the hierarchy are controlled by different people. An inheritance hierarchy is less problematic when it's entirely under one person's control.

Composition-Based Designs

Composition is loosely coupled. When you compose a reference as an instance variable of a class, it's part of the class's implementation details that are hidden from the class's client code. If the reference's class type changes, you may need to make changes to the composing class's internal details, but those changes do not affect the client code.

In addition, inheritance is done at compile time. Composition is more flexible—it, too, can be done at compile time, but it also can be done at execution time because non-`final` references to composed objects can be modified. We call this dynamic composition. This is another aspect of loose coupling—if the reference is of a superclass type, you can replace the referenced object with an object of *any* type that has an *is-a* relationship with the reference's class type.

When you use a composition approach instead of inheritance, you'll typically create a larger number of smaller classes, each focused on one responsibility. Smaller classes generally are easier to test, debug and modify.

Java does not offer multiple inheritance—each class in Java may extend only one class. However, a new class may reuse

the capabilities of one or more other classes by composition. As you'll see in Chapter 10, Object-Oriented Programming: Polymorphism and Interfaces, we also can get many of the benefits of multiple inheritance by implementing multiple interfaces.

Performance Tip 9.1

A potential disadvantage of composition is that it typically requires more objects at runtime, which might negatively impact garbage-collection and virtual-memory performance. Virtual-memory architectures and performance issues are typically discussed in operating systems courses.

Software Engineering Observation 9.12

A `public` *method of a composing class can call a method of a composed object to perform a task for the benefit of the composing class's clients. This is known as forwarding the method call and is a common way to reuse a class's capabilities via composition rather than inheritance.*

Software Engineering Observation 9.13

When implementing a new class and choosing whether you should reuse an existing class via inheritance or composition, use composition if the existing class's `public` *methods should not be part of the new class's* `public` *interface.*

Recommended Exercises

Exercise 9.3 asks you to reimplement this chapter's `CommissionEmployee–BasePlusCommissionEmployee` hierarchy using composition, rather than inheritance. Exercise 9.16 asks you to reimplement the hierarchy using a combination of composition and inheritance in which you'll see the benefits of composition's loose coupling.